DIWAKAR EDUCATION HUB

System Software and Operating System
Unit – 5

As per updated syllabus

DIWAKAR EDUCATION HUB

2020

THE LEARN WITH EXPERTIES

System software

System software is a type of computer program that is designed to run a computer's hardware and application programs. If we think of the computer system as a layered model, the system software is the interface between the hardware and user applications. The operating system (OS) is the best-known example of system software. The OS manages all the other programs in a computer.

Other examples of system software include:

- The BIOS (basic input/output system) gets the computer system started after you turn it on and manages the data flow between the operating system and attached devices such as the hard disk, video adapter, keyboard, mouse and printer.
- The boot program loads the operating system into the computer's main memory or random access memory (RAM).
- An assembler takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations.
- A device driver controls a particular type of device that is attached to your computer, such as a keyboard or a mouse. The driver program converts the more general input/output instructions of the operating system to messages that the device type can understand.

Additionally, system software can also include system utilities, such as the disk defragmenter and System Restore, and development tools, such as compilers and debuggers.

System software and application programs are the two main types of computer software. Unlike system software, an application program (often just called an application or app) performs a particular function for the user. Examples include browsers, email clients, word processors and spreadsheets.

System Software is a set of programs that control and manage the operations of computer hardware. It also helps application programs to execute correctly.

System Software are designed to control the operation and extend the processing functionalities of a computer system. System software makes the operation of a computer more fast, effective, and secure. Example: Operating system, programming language, Communication software, etc.

Types of System Software

Important types of System Software:

- **Operating systems:-** Operating system software helps you for the effective utilization of all hardware and software components of a computer system.
- **Programming language translators:-** Transforms the instructions prepared by developers in a programming language into a form that can be interpreted or compiled and executed by a computer system.
- **Communication Software : -** Communication software allows us to transfer data and programs from one computer system to another.
- **Utility programs:** - Utility programs are a set of programs that help users in system maintenance tasks, and in performing tasks of routine nature.

Features of System Software

An important feature of System Software are:

- System Software is closer to the system
- Generally written in a low-level language
- The system software is difficult to design and understand
- Fast in speed
- Less interactive
- Smaller in size

- Hard to manipulate

Machine language

Machine language, or machine code, is a low-level language comprised of binary digits (ones and zeros). High-level languages, such as Swift and C++ must be compiled into machine language before the code is run on a computer.

Since computers are digital devices, they only recognize binary data. Every program, video, image, and character of text is represented in binary. This binary data, or machine code, is processed as input by the CPU. The resulting output is sent to the operating system or an application, which displays the data visually. For example, the ASCII value for the letter "A" is 01000001 in machine code, but this data is displayed as "A" on the screen. An image may have thousands or even millions of binary values that determine the color of each pixel.

While machine code is comprised of 1s and 0s, different processor architectures use different machine code. For example, a PowerPC processor, which has a RISC architecture, requires different code than an Intel x86 processor, which has a CISC architecture. A compiler must compile high-level source code for the correct processor architecture in order for a program to run correctly.

Assembly Language

Programming in Machine language is tedious (you have to program every command from scratch) and hard to read & modify (the 1s and 0s are kind of hard to work with…). For these reasons, Assembly language was developed as an alternative to Machine language.

Assembly Language uses short descriptive words (mnemonic) to represent each of the Machine Language instructions.

For example the mnemonic **add** means to add numbers together, and **sub** means to subtract the numbers. So if you want to add the numbers 2 and 3 in assembly language, it would look like this:

add 2, 3, result

So Assembly Languages were developed to make programming easier. However, the computer cannot directly execute the assembly language. First another program called the **assembler** is used to translate the Assembly Language into machine code.



Machine Language vs Assembly Language

Machine language and assembly language are both low-level languages, but machine code is below assembly in the hierarchy of computer languages. Assembly language includes human-readable commands, such as mov, add, and sub, while machine language does not contain any words or even letters. Some developers manually write assembly language to optimize a program, but they do not write machine code. Only developers who write software compilers need to worry about machine language.

NOTE: While machine code is technically comprised of binary data, it may also be represented in hexadecimal values. For example, the letter "Z," which is 01011010 in binary, may be displayed as 5A in hexadecimal code.

High Level Language

The high level language is simple and easy to understand and it is similar to English language. For example, COBOL, FORTRAN, BASIC, C, C+, Python, etc.

High-level languages are very important, as they help in developing complex software and they have the following advantages −

- Unlike assembly language or machine language, users do not need to learn the high-level language in order to work with it.
- High-level languages are similar to natural languages, therefore, easy to learn and understand.

- High-level language is designed in such a way that it detects the errors immediately.
- High-level language is easy to maintain and it can be easily modified.
- High-level language makes development faster.
- High-level language is comparatively cheaper to develop.
- High-level language is easier to document.

Although a high-level language has many benefits, yet it also has a drawback. It has poor control on machine/hardware.

The following table lists down the frequently used languages –

SQL
Java
Javascript
C#
Python
C++
PHP
IOS
Ruby/Rails
.Net

A **high-level language** is a programming language that uses English and mathematical symbols, like +, -, % and many others, in its instructions. When using the term 'programming languages,' most people are actually referring to high-level languages. High-level languages are the languages most often used by programmers to write programs. Examples of high-level languages are C++, Fortran, Java and Python.

To get a flavor of what a high-level language actually looks like, consider an ATM machine where someone wants to make a withdrawal of $100. This amount needs to be compared to the account balance to make sure there are enough funds. The instruction in a high-level computer language would look something like this:

```
x = 100
if balance  x:
 print 'Insufficient balance'
else:
 print 'Please take your money'
```

This is not exactly how real people communicate, but it is much easier to follow than a series of 1s and 0s in binary code.

There are a number of advantages to high-level languages. The first advantage is that high-level languages are much closer to the logic of a human language. A high-level language uses a set of rules that dictate how words and symbols can be put together to form a program. Learning a high-level language is not unlike learning another human language - you need to learn vocabulary and grammar so you can make sentences. To learn a programming language, you need to learn commands, syntax and logic, which correspond closely to vocabulary and grammar.

The second advantage is that the code of most high-level languages is portable and the same code can run on different hardware. Both machine code and assembly languages are hardware specific and not portable. This means that the machine code used to run a program on one specific computer needs to be modified to run on another computer. Portable code in a high-level language can run on multiple computer systems without modification. However, modifications to code in high-level languages may be necessary because of the operating system. For example, programs written for Windows typically don't run on a Mac.

A high-level language cannot be understood directly by a computer, and it needs to be translated into machine code. There are two ways to do this, and they are

related to how the program is executed: a high-level language can be compiled or interpreted.

Types of Programming Languages

- **Data-oriented Language**: These programming languages are designed for searching and manipulating relation that have been described as entity relationship tables which map one set of things into other sets. Example: SQL
- Imperative Language: ?
- Object-oriented programming (OOP) support objects defined by their class. … Focuses on objects over action, data over logic.

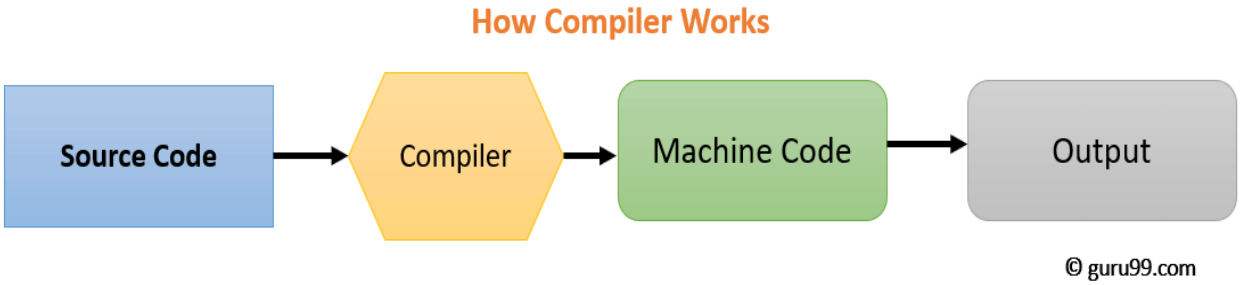
Compiler

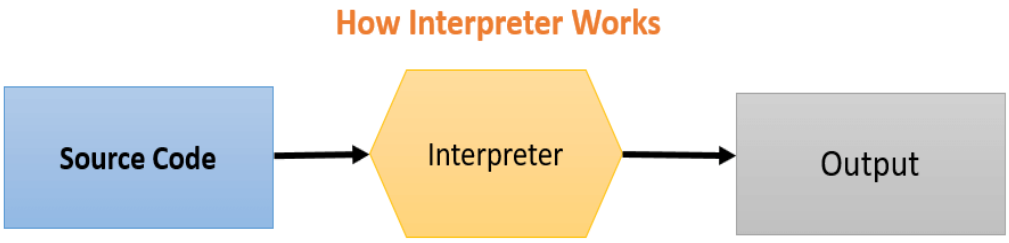
A compiler is a computer program that transforms code written in a high-level programming language into the machine code. It is a program which translates the human-readable code to a language a computer processor understands (binary 1 and 0 bits). The computer processes the machine code to perform the corresponding tasks.

A compiler should comply with the syntax rule of that programming language in which it is written. However, the compiler is only a program and cannot fix errors found in that program. So, if you make a mistake, you need to make changes in the syntax of your program. Otherwise, it will not compile.

Interpreter

An interpreter is a computer program, which coverts each high-level program statement into the machine code. This includes source code, pre-compiled code, and scripts. Both compiler and interpreters do the same job which is converting higher level programming language to machine code. However, a compiler will convert the code into machine code (create an exe) before program run. Interpreters convert code into machine code when the program is run.





Difference Between Compiler and Interpreter

Basis of difference	Compiler	Interpreter
Programming Steps	• Create the program. • Compile will parse or analyses all of the language statements for its correctness. If incorrect, throws an error • If no error, the compiler will convert source code to machine code. • It links different code files into a runnable program(know as exe)	• Create the Program • No linking of files or machine code generation • Source statements executed line by line DURING Execution

Basis of difference	Compiler	Interpreter
	• Run the Program	
Advantage	The program code is already translated into machine code. Thus, it code execution time is less.	Interpreters are easier to use, especially for beginners.
Disadvantage	You can't change the program without going back to the source code.	Interpreted programs can run on computers that have the corresponding interpreter.
Machine code	Store machine language as machine code on the disk	Not saving machine code at all.
Running time	Compiled code run faster	Interpreted code run slower
Model	It is based on language translationlinking-loading model.	It is based on Interpretation Method.
Program generation	Generates output program (in the form of exe) which can be run independently from the original program.	Do not generate output program. So they evaluate the source program at every time during execution.
Execution	Program execution is separate from the compilation. It performed only after the entire output program is compiled.	Program Execution is a part ofInterpretation process, so it is performed line by line.
Memory	Target program executeindependently and	The interpreter exists in the memory

Basis of difference	Compiler	Interpreter
requirement	do not require the compiler in the memory.	during interpretation.
Best suited for	Bounded to the specific target machine and cannot be ported. C and C++ are a most popular a programming language which uses compilation model.	For web environments, where load times are important. Due to all the exhaustive analysis is done, compiles take relatively larger time to compile even small code that may not be run multiple times. In such cases, interpreters are better.
Code Optimization	The compiler sees the entire code upfront. Hence, they perform lots of optimizations that make code run faster	Interpreters see code line by line, and thus optimizations are not as robust as compilers
Dynamic Typing	Difficult to implement as compilers cannot predict what happens at turn time.	Interpreted languages support Dynamic Typing
Usage	It is best suited for the Production Environment	It is best suited for the program and developmentenvironment.
Error execution	Compiler displays all errors and warning at the compilation time. Therefore, you can't run the program without fixing errors	The interpreter reads a single statement and shows the error if any. You must correct the error to interpret next line.
Input	It takes an entire program	It takes a single line of code.

Basis of difference	Compiler	Interpreter
Output	Compliers generates intermediate machnie code.	Interpreter never generate any intermediate machnie code.
Errors	Display all errors after, compilation, all at the same time.	Displays all errors of each line one by one.
Pertaining Programming languages	C,C++,C#, Scala, Java all use complier.	PHP, Perl, Ruby uses an interpreter.

Role of Compiler

- Compliers reads the source code, outputs executable code
- Translates software written in a higher-level language into instructions that computer can understand. It converts the text that a programmer writes into a format the CPU can understand.
- The process of compilation is relatively complicated. It spends a lot of time analyzing and processing the program.
- The executable result is some form of machine-specific binary code.

Role of Interpreter

- The interpreter converts the source code line-by-line during RUN Time.
- Interpret completely translates a program written in a high-level language into machine level language.
- Interpreter allows evaluation and modification of the program while it is executing.

- Relatively less time spent for analyzing and processing the program
- Program execution is relatively slow compared to compiler

HIGH-LEVEL LANGUAGES

High-level languages, like C, C++, JAVA, etc., are very near to English. It makes programming process easy. However, it must be translated into machine language before execution. This translation process is either conducted by either a compiler or an interpreter. Also known as source code.

MACHINE CODE

Machine languages are very close to the hardware. Every computer has its machine language. A machine language programs are made up of series of binary pattern. (Eg. 110110) It represents the simple operations which should be performed by the computer. Machine language programs are executable so that they can be run directly.

OBJECT CODE

On compilation of source code, the machine code generated for different processors like Intel, AMD, an ARM is different. tTo make code portable, the source code is first converted to Object Code. It is an intermediary code (similar to machine code) that no processor will understand. At run time, the object code is converted to the machine code of the underlying platform.

Java is both Compiled and Interpreted.

To exploit relative advantages of compilers are interpreters some programming language like Java are both compiled and interpreted. The Java code itself is compiled into Object Code. At run time, the JVM interprets the Object code into machine code of the target computer.

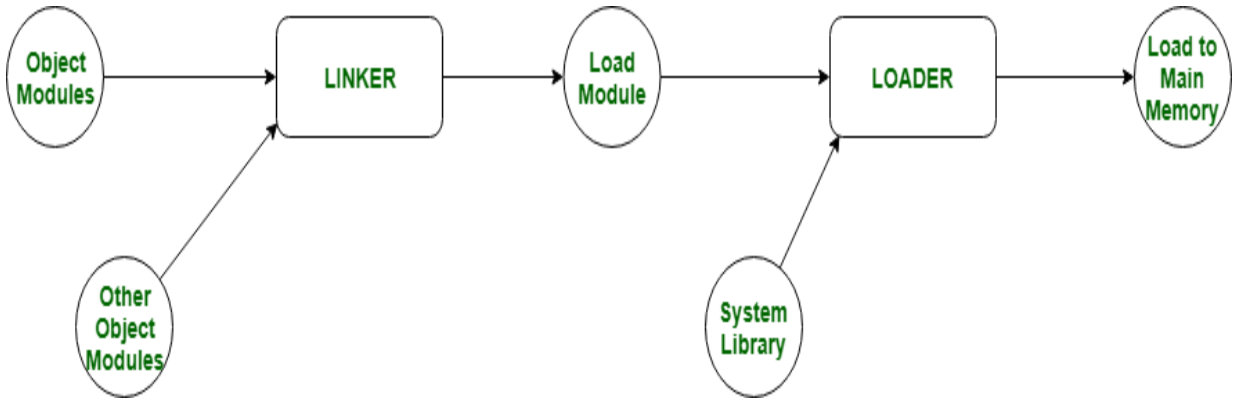
Linking and Loading

Linking and **Loading** are the utility programs that play a important role in the execution of a program. Linking intakes the object codes generated by the

assembler and combines them to generate the executable module. On the other hand, the loading loads this executable module to the main memory for execution.

Loading:
Bringing the program from secondary memory to main memory is called Loading.

Linking:
Establishing the linking between all the modules or all the functions of the program in order to continue the program execution is called linking.



Linker is a program in a system which helps to link a object modules of program into a single object file. It performs the process of linking. Linker are also called link editors. Linking is process of collecting and maintaining piece of code and data into a single file. Linker also link a particular module into system library. It takes object modules from assembler as input and forms an executable file as output for loader.

Linking is performed at both compile time, when the source code is translated into machine code and load time, when the program is loaded into memory by the loader. Linking is performed at the last step in compiling a program.

Source code -> compiler -> Assembler -> Object code -> Linker -> Executable file -> Loader

Linking is of two types:
1. Static Linking –

It is performed during the compilation of source program. Linking is performed before execution in static linking. It takes collection of relocatable object file and command-line argument and generate fully linked object file that can be loaded and run.

Static linker perform two major task:

- **Symbol resolution –** It associates each symbol reference with exactly one symbol definition .Every symbol have predefined task.
- **Relocation –** It relocate code and data section and modify symbol references to the relocated memory location.

The linker copy all library routines used in the program into executable image. As a result, it require more memory space. As it does not require the presence of library on the system when it is run . so, it is faster and more portable. No failure chance and less error chance.

2. Dynamic linking – Dynamic linking is performed during the run time. This linking is accomplished by placing the name of a shareable library in the executable image. There is more chances of error and failure chances. It require less memory space as multiple program can share a single copy of the library.

Here we can perform code sharing. it means we are using a same object a number of times in the program. Instead of linking same object again and again into the library, each module share information of a object with other module having same object. The shared library needed in the linking is stored in virtual memory to save RAM. In this linking we can also relocate the code for the smooth running of code but all the code is not relocatable.It fixes the address at run time.

Differences between Linking and Loading:

1. The key difference between linking and loading is that the linking generates the executable file of a program whereas, the loading loads the executable file obtained from the linking into main memory for execution.

2. The linking intakes the object module of a program generated by the assembler. However, the loading intakes the executable module generated by the linking.
3. The linking combines all object modules of a program to generate executable modules it also links the library function in the object module to built-in libraries of the high-level programming language. On the other hand, loading allocates space to an executable module in main memory.

Loading and Linking are further categorized into 2 types:

STATIC	DYNAMIC
Loading the entire program into the main memory before start of the program execution is called as static loading.	Loading the program into the main memory on demand is called as dynamic loading.
Inefficent utilization of memory because whether it is required or not required entire program is brought into the main memory.	Efficent utilization of memory.
Program execution will be faster.	Program execution will be slower.
Statically linked program takes constant load time every time it is loaded into the memory for execution.	Dynamic linking is performed at run time by the operating system.
If the static loading is used then accordingly static linking is	If the dynamic loading is used then

STATIC	DYNAMIC
applied.	accordingly dynamic linking is applied.
Static linking is performed by programs called linkers as the last step in compiling a program. Linkers are also called link editors.	In dynamic linking this is not the case and individual shared modules can be updated and recompiled.This is one of the greatest advantages dynamic linking offers.
In static linking if any of the external programs has changed then they have to be recompiled and re-linked again else the changes won't reflect in existing executable file.	In dynamic linking load time might be reduced if the shared library code is already present in memory.

Macros

Writing a macro is another way of ensuring modular programming in assembly language.

- A macro is a sequence of instructions, assigned by a name and could be used anywhere in the program.
- In NASM, macros are defined with **%macro** and **%endmacro** directives.
- The macro begins with the %macro directive and ends with the %endmacro directive.

The Syntax for macro definition −

```
%macro macro_name  number_of_params
<macro body>
%endmacro
```

Where, number_of_params specifies the number parameters, macro_name specifies the name of the macro.

The macro is invoked by using the macro name along with the necessary parameters. When you need to use some sequence of instructions many times in a program, you can put those instructions in a macro and use it instead of writing the instructions all the time.

For example, a very common need for programs is to write a string of characters in the screen. For displaying a string of characters, you need the following sequence of instructions −

```
mov     edx,len     ;message length
mov     ecx,msg     ;message to write
mov     ebx,1       ;file descriptor (stdout)
mov     eax,4       ;system call number (sys_write)
int     0x80        ;call kernel
```

In the above example of displaying a character string, the registers EAX, EBX, ECX and EDX have been used by the INT 80H function call. So, each time you need to display on screen, you need to save these registers on the stack, invoke INT 80H and then restore the original value of the registers from the stack. So, it could be useful to write two macros for saving and restoring data.

We have observed that, some instructions like IMUL, IDIV, INT, etc., need some of the information to be stored in some particular registers and even return values in some specific register(s). If the program was already using those registers for keeping important data, then the existing data from these registers should be saved in the stack and restored after the instruction is executed.

Example

Following example shows defining and using macros −

```
; A macro with two parameters
; Implements the write system call
  %macro write_string 2
     mov   eax, 4
     mov   ebx, 1
     mov   ecx, %1
     mov   edx, %2
```

```
    int   80h
  %endmacro

section  .text
  global _start          ;must be declared for using gcc

_start:                  ;tell linker entry point
  write_string msg1, len1
  write_string msg2, len2
  write_string msg3, len3

  mov eax,1              ;system call number (sys_exit)
  int 0x80               ;call kernel

section  .data
msg1 db            'Hello, programmers!',0xA,0xD
len1 equ $ - msg1

msg2 db 'Welcome to the world of,', 0xA,0xD
len2 equ $- msg2

msg3 db 'Linux assembly programming! '
len3 equ $- msg3
```

When the above code is compiled and executed, it produces the following result –

```
Hello, programmers!
Welcome to the world of,
Linux assembly programming!
```

Debugger

A debugger is a software program used to test and find bugs (errors) in other programs.

A debugger is also known as a debugging tool.

A debugger is a computer program used by programmers to test and debug a target program. Debuggers may use instruction-set simulators, rather than running a program directly on the processor to achieve a higher level of control over its execution. This allows debuggers to stop or halt the program according to specific conditions. However, use of simulators decreases execution speed.

When a program crashes, debuggers show the position of the error in the target program. Most debuggers also are capable of running programs in a step-by-step mode, besides stopping on specific points. They also can often modify the state of programs while they are running.

Even the most experienced software programmers usually don't get it right on their first try. Certain errors, often called bugs, can occur in programs, causing them to not function as the programmer expected. Sometimes these errors are easy to fix, while some bugs are very difficult to trace. This is especially true for large programs that consist of several thousand lines of code.

Fortunately, there are programs called debuggers that help software developers find and eliminate bugs while they are writing programs. A debugger tells the programmer what types of errors it finds and often marks the exact lines of code where the bugs are found. Debuggers also allow programmers to run a program step by step so that they can determine exactly when and why a program crashes. Advanced debuggers provide detailed information about threads and memory being used by the program during each step of execution. You could say a powerful debugger program is like OFF! with 100% deet.

Operating System

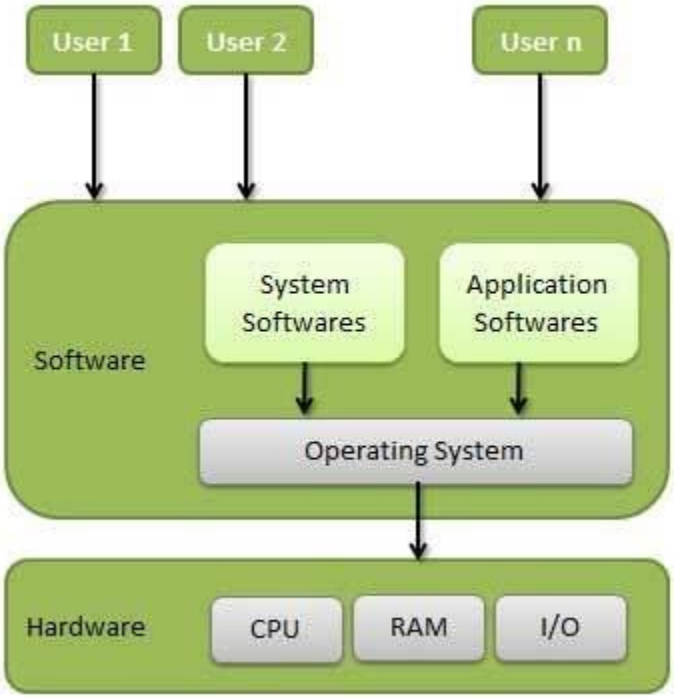
An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is a vital component of the system software in a computer system. This tutorial will take you through step by step approach while learning Operating System concepts.

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Why to Learn Operating System?

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Applications of Operating System

Following are some of the important activities that an Operating System performs –

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** − Recording delays between request for a service and response from the system.
- **Job accounting** − Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** − Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other softwares and users** − Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

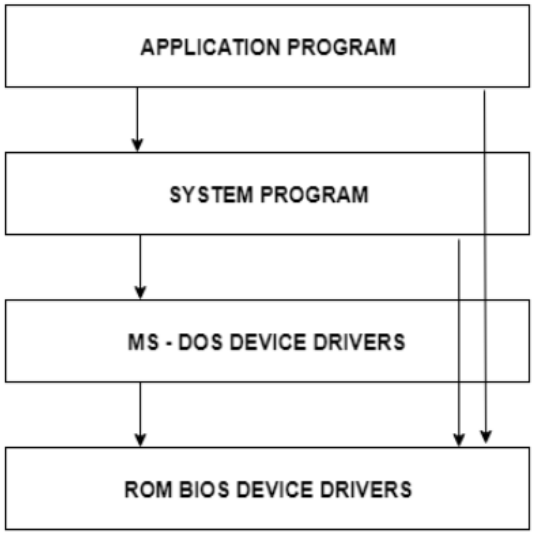
Operating System Structure

An operating system is a construct that allows the user application programs to interact with the system hardware. Since the operating system is such a complex structure, it should be created with utmost care so it can be used and modified easily. An easy way to do this is to create the operating system in parts. Each of these parts should be well defined with clear inputs, outputs and functions.

Simple Structure

There are many operating systems that have a rather simple structure. These started as small systems and rapidly expanded much further than their scope. A common example of this is MS-DOS. It was designed simply for a niche amount for people. There was no indication that it would become so popular.

An image to illustrate the structure of MS-DOS is as follows –



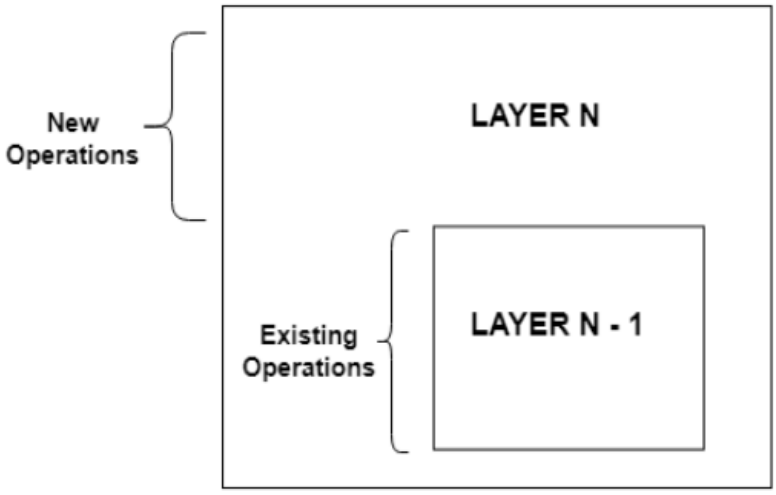
MS-DOS STRUCTURE

It is better that operating systems have a modular structure, unlike MS-DOS. That would lead to greater control over the computer system and its various applications. The modular structure would also allow the programmers to hide information as required and implement internal routines as they see fit without changing the outer specifications.

Layered Structure

One way to achieve modularity in the operating system is the layered approach. In this, the bottom layer is the hardware and the topmost layer is the user interface.

An image demonstrating the layered approach is as follows –



Layered Structure of Operating System

As seen from the image, each upper layer is built on the bottom layer. All the layers hide some structures, operations etc from their upper layers.

One problem with the layered structure is that each layer needs to be carefully defined. This is necessary because the upper layers can only use the functionalities of the layers below them.

Operating System Services

Operating system services are responsible for the management of platform resources, including the processor, memory, files, and input and output. They generally shield applications from the implementation details of the machine. Operating system services include:

- **Kernel operations** provide low-level services necessary to:
    - create and manage processes and threads of execution

  - execute programs
  - define and communicate asynchronous events
  - define and process system clock operations
  - implement security features
  - manage files and directories, and
  - control input/output processing to and from peripheral devices.

- Some kernel services have analogues described in the paragraph on Object Services, such as concurrency control services.
- **Command interpreter and utility services** include mechanisms for services at the operator level, such as:
  - comparing, printing, and displaying file contents
  - editing files
  - searching patterns
  - evaluating expressions
  - logging messages
  - moving files between directories
  - sorting data
  - executing command scripts
  - local print spooling
  - scheduling signal execution processes, and
  - accessing environment information.
- **Batch processing** services support the capability to queue work (jobs) and manage the sequencing of processing based on job control commands and lists of data. These services also include support for the management of the output of batch processing, which frequently includes updated files or

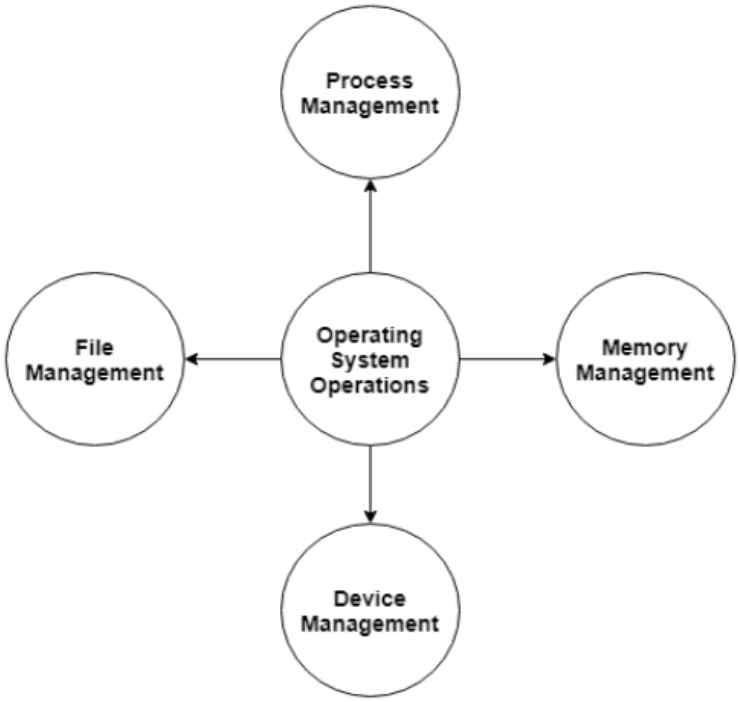
databases and information products such as printed reports or electronic documents. Batch processing is performed asynchronously from the user requesting the job.

- **File and directory synchronization** services allow local and remote copies of files and directories to be made identical. Synchronization services are usually used to update files after periods of off line working on a portable system.

Operating System Operations

An operating system is a construct that allows the user application programs to interact with the system hardware. Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.

The major operations of the operating system are process management, memory management, device management and file management. These are given in detail as follows:



Process Management

The operating system is responsible for managing the processes i.e assigning the processor to a process at a time. This is known as process scheduling. The different algorithms used for process scheduling are FCFS (first come first served), SJF (shortest job first), priority scheduling, round robin scheduling etc.

There are many scheduling queues that are used to handle processes in process management. When the processes enter the system, they are put into the job queue. The processes that are ready to execute in the main memory are kept in the ready queue. The processes that are waiting for the I/O device are kept in the device queue.

Memory Management

Memory management plays an important part in operating system. It deals with memory and the moving of processes from disk to primary memory for execution and back again.

The activities performed by the operating system for memory management are −

- The operating system assigns memory to the processes as required. This can be done using best fit, first fit and worst fit algorithms.
- All the memory is tracked by the operating system i.e. it nodes what memory parts are in use by the processes and which are empty.
- The operating system deallocated memory from processes as required. This may happen when a process has been terminated or if it no longer needs the memory.

Device Management

There are many I/O devices handled by the operating system such as mouse, keyboard, disk drive etc. There are different device drivers that can be connected to the operating system to handle a specific device. The device controller is an

interface between the device and the device driver. The user applications can access all the I/O devices using the device drivers, which are device specific codes.

File Management

Files are used to provide a uniform view of data storage by the operating system. All the files are mapped onto physical devices that are usually non volatile so data is safe in the case of system failure.

The files can be accessed by the system in two ways i.e. sequential access and direct access −

- **Sequential Access**

The information in a file is processed in order using sequential access. The files records are accessed on after another. Most of the file systems such as editors, compilers etc. use sequential access.

- **Direct Access**

In direct access or relative access, the files can be accessed in random for read and write operations. The direct access model is based on the disk model of a file, since it allows random accesses.

System Calls

In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system's kernel. System call **provides** the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Services Provided by System Calls :

1. Process creation and management
2. Main memory management
3. File Access, Directory and File system management
4. Device handling(I/O)
5. Protection
6. Networking, etc.

Types of System Calls : There are 5 different categories of system calls –

1. **Process control:** end, abort, create, terminate, allocate and free memory.
2. **File management:** create, open, close, delete, read file etc.
3. Device management
4. Information maintenance
5. Communication

Examples of Windows and Unix System Calls –

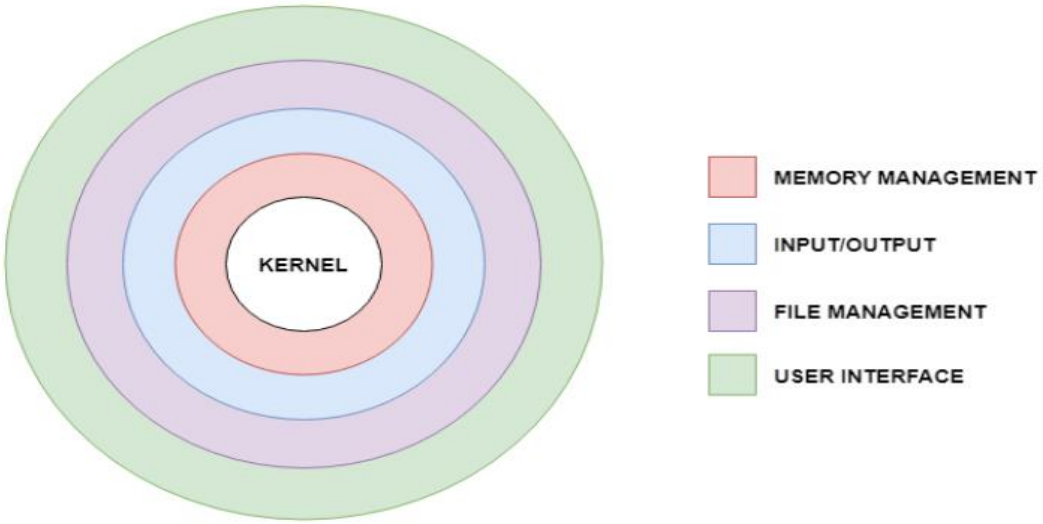
	WINDOWS	UNIX
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()

Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Operating System Design and Implementation

An operating system is a construct that allows the user application programs to interact with the system hardware. Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.

There are many problems that can occur while designing and implementing an operating system. These are covered in operating system design and implementation.



Layered Operating System Design

Operating System Design Goals

It is quite complicated to define all the goals and specifications of the operating system while designing it.The design changes depending on the type of the operating system i.e if it is batch system, time shared system, single user system, multi user system, distributed system etc.

There are basically two types of goals while designing an operating system. These are –

User Goals

The operating system should be convenient, easy to use, reliable, safe and fast according to the users. However, these specifications are not very useful as there is no set method to achieve these goals.

System Goals

The operating system should be easy to design, implement and maintain. These are specifications required by those who create, maintain and operate the operating system. But there is not specific method to achieve these goals as well.

Operating System Mechanisms and Policies

There is no specific way to design an operating system as it is a highly creative task. However, there are general software principles that are applicable to all operating systems.

A subtle difference between mechanism and policy is that mechanism shows how to do something and policy shows what to do. Policies may change over time and this would lead to changes in mechanism. So, it is better to have a general mechanism that would require few changes even when a policy change occurs.

For example - If the mechanism and policy are independent, then few changes are required in mechanism if policy changes. If a policy favours I/O intensive processes over CPU intensive processes, then a policy change to preference of CPU intensive processes will not change the mechanism.

Operating System Implementation

The operating system needs to be implemented after it is designed. Earlier they were written in assembly language but now higher level languages are used. The first system not written in assembly language was the Master Control Program (MCP) for Burroughs Computers.

Advantages of Higher Level Language

There are multiple advantages to implementing an operating system using a higher level language such as: the code is written more fast, it is compact and also easier to debug and understand. Also, the operating system can be easily moved from one hardware to another if it is written in a high level language.

Disadvantages of Higher Level Language

Using high level language for implementing an operating system leads to a loss in speed and increase in storage requirements. However in modern systems only a small amount of code is needed for high performance, such as the CPU scheduler and memory manager. Also, the bottleneck routines in the system can be replaced by assembly language equivalents if required.

System Boot

The BIOS, operating system and hardware components of a computer system should all be working correctly for it to boot. If any of these elements fail, it leads to a failed boot sequence.

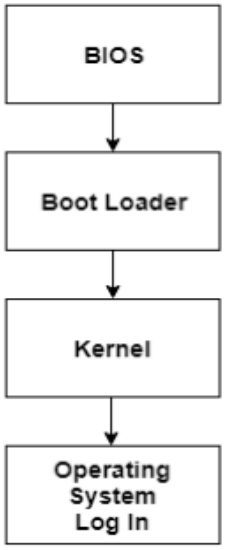
Booting the system is done by loading the kernel into main memory, and starting its execution.

The CPU is given a reset event, and the instruction register is loaded with a predefined memory location, where execution starts.

- The initial bootstrap program is found in the BIOS read-only memory.
- This program can run diagnostics, initialize all components of the system, loads and starts the Operating System loader. (Called **boot strapping**)
- The loader program loads and starts the operating system.
- When the Operating system starts, it sets up needed data structures in memory, sets several registers in the CPU, and then creates and starts the first user level program. From this point, the operating system only runs in response to interrupts.

System Boot Process

The following diagram demonstrates the steps involved in a system boot process –



Here are the steps −

- The CPU initializes itself after the power in the computer is first turned on. This is done by triggering a series of clock ticks that are generated by the system clock.
- After this, the CPU looks for the system's ROM BIOS to obtain the first instruction in the start-up program. This first instruction is stored in the ROM BIOS and it instructs the system to run POST (Power On Self Test) in a memory address that is predetermined.
- POST first checks the BIOS chip and then the CMOS RAM. If there is no battery failure detected by POST, then it continues to initialize the CPU.
- POST also checks the hardware devices, secondary storage devices such as hard drives, ports etc. And other hardware devices such as the mouse and keyboard. This is done to make sure they are working properly.
- After POST makes sure that all the components are working properly, then the BIOS finds an operating system to load.

- In most computer system's, the operating system loads from the C drive onto the hard drive. The CMOS chip typically tells the BIOS where the operating system is found.
- The order of the different drives that CMOS looks at while finding the operating system is known as the boot sequence. This sequence can be changed by changing the CMOS setup.
- After finding the appropriate boot drive, the BIOS first finds the boot record which tells it to find the beginning of the operating system.
- After the initialization of the operating system, the BIOS copies the files into the memory. Then the operating system controls the boot process.
- In the end, the operating system does a final inventory of the system memory and loads the device drivers needed to control the peripheral devices.
- The users can access the system applications to perform various tasks.

Without the system boot process, the computer users would have to download all the software components, including the ones not frequently required. With the system boot, only those software components need to be downloaded that are legitimately required and all extraneous components are not required. This process frees up a lot of space in the memory and consequently saves a lot of time.

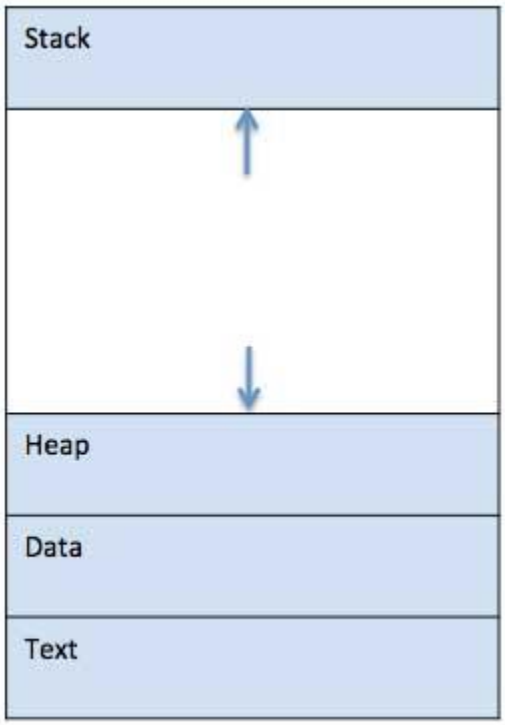
Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.

2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```
#include <stdio.h>

int main() {

   printf("Hello, World! \n");

   return 0;

}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

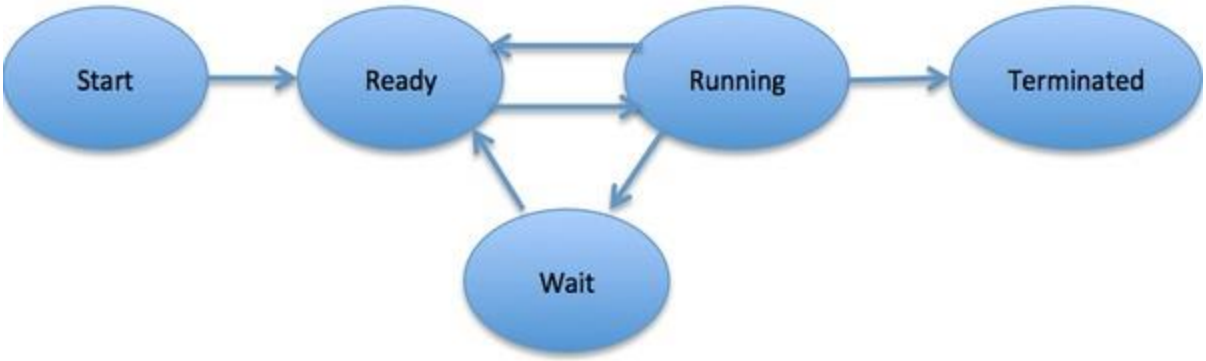
Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start** This is the initial state when a process is first started/created.
2	**Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

4	**Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



Process Control Block (PCB)

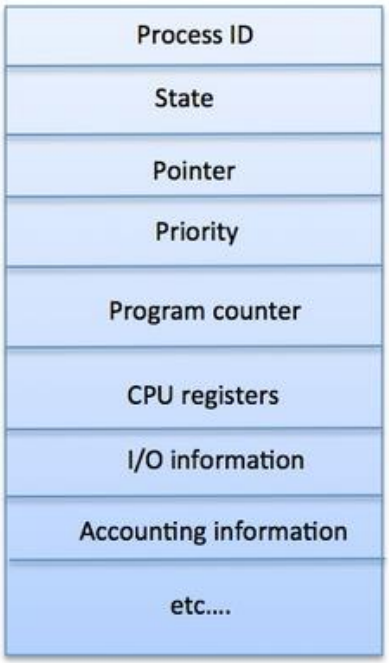
A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.

2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.

9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB –



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Process Scheduling

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

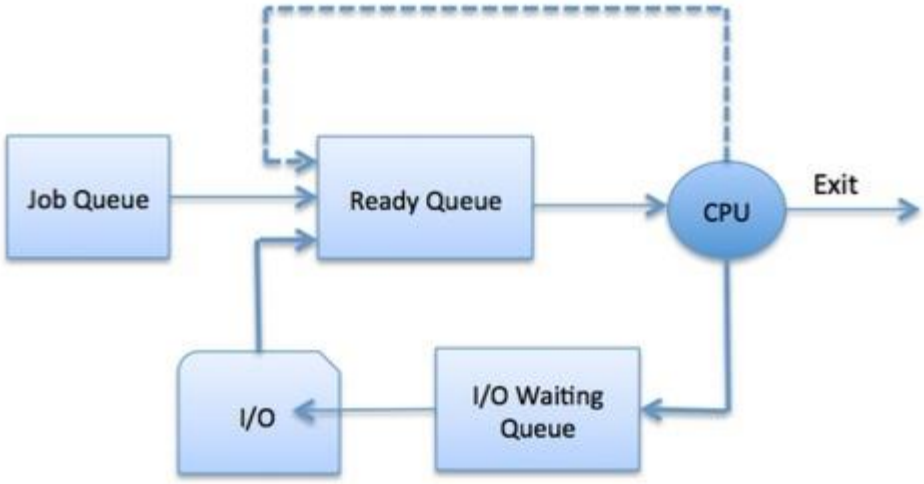
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other

processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Comparison among Scheduler

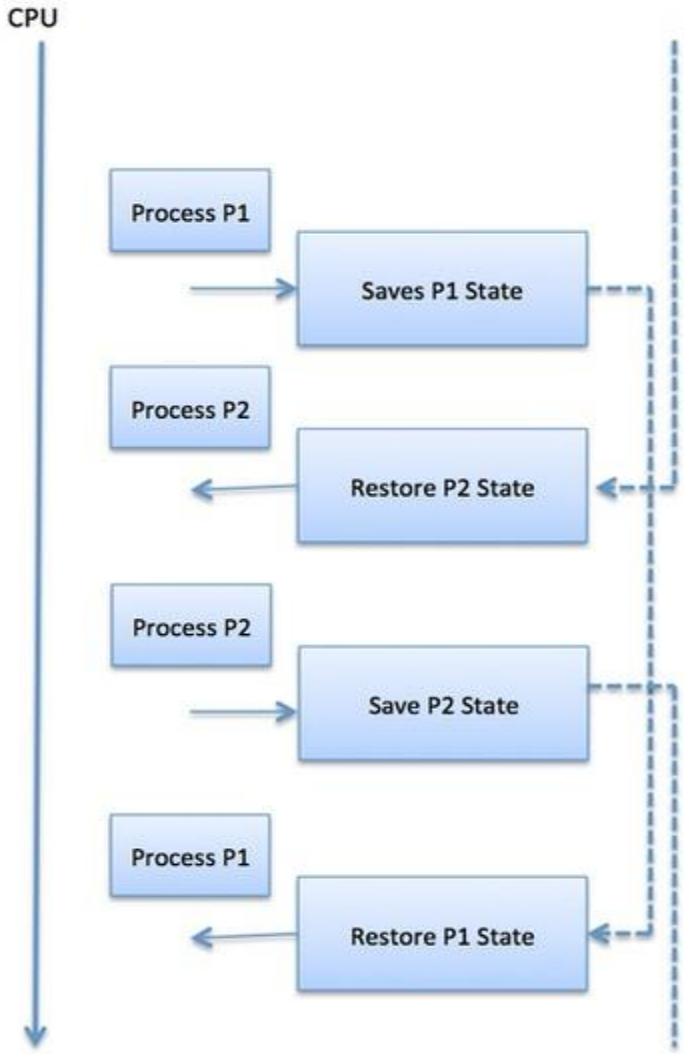
S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from

the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.



Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time,

some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information
- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

Process Operations

Process operations, also called process manufacturing or process production, is the mass production method of producing products in a continuous flow. In other words, this is a conveyer belt system that produces identical, standardized items at a high rate of speed.

Different Operations on Processes

There are many operations that can be performed on processes. Some of these are process creation, process preemption, process blocking, and process termination. These are given in detail as follows −

Process Creation

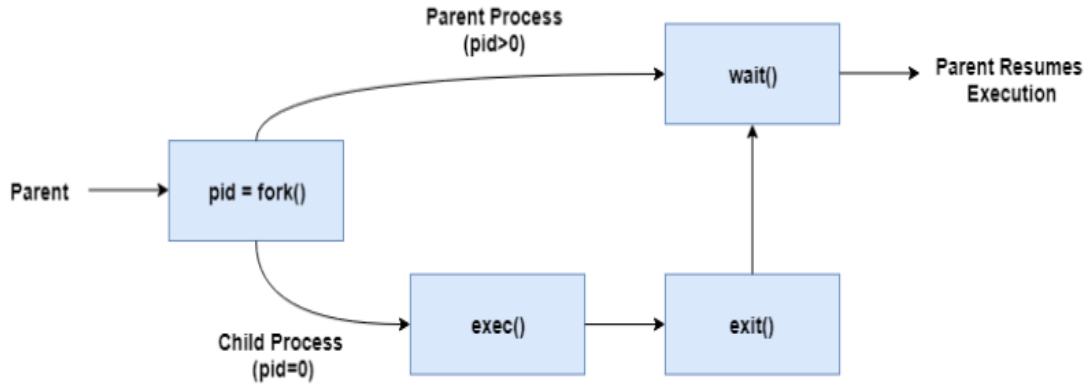
Processes need to be created in the system for different operations. This can be done by the following events −

- User request for process creation
- System initialization
- Execution of a process creation system call by a running process

- Batch job initialization

A process may be created by another process using fork(). The creating process is called the parent process and the created process is the child process. A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files, and environment strings. However, they have distinct address spaces.

A diagram that demonstrates process creation using fork() is as follows –

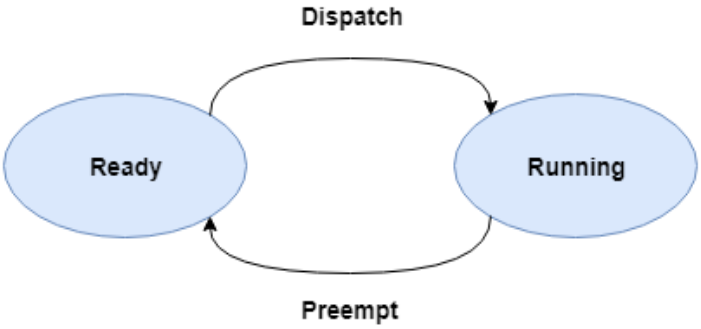


Process Creation Using fork()

Process Preemption

An interrupt mechanism is used in preemption that suspends the process executing currently and the next process to execute is determined by the short-term scheduler. Preemption makes sure that all processes get some CPU time for execution.

A diagram that demonstrates process preemption is as follows –

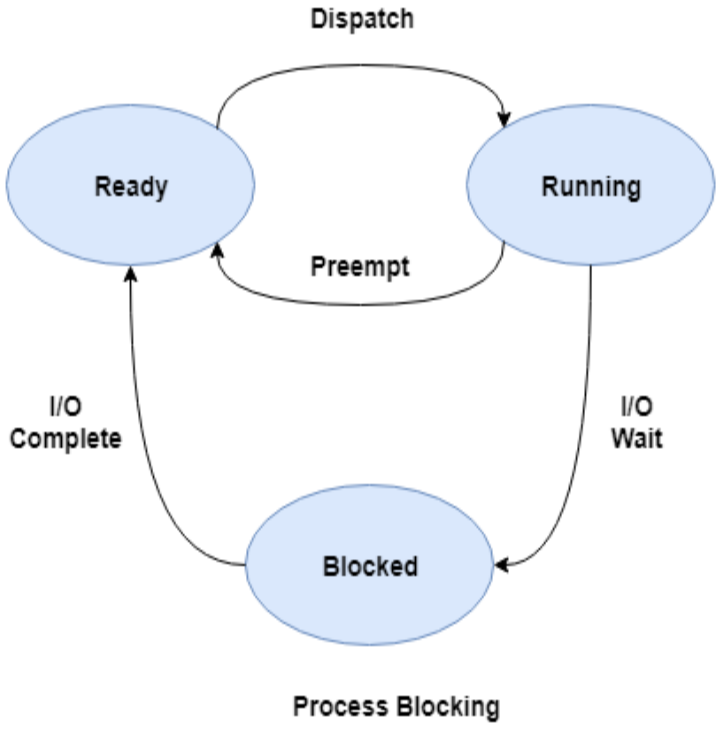


Process Preemption

Process Blocking

The process is blocked if it is waiting for some event to occur. This event may be I/O as the I/O events are executed in the main memory and don't require the processor. After the event is complete, the process again goes to the ready state.

A diagram that demonstrates process blocking is as follows –



Process Termination

After the process has completed the execution of its last instruction, it is terminated. The resources held by a process are released after it is terminated.

A child process can be terminated by its parent process if its task is no longer relevant. The child process sends its status information to the parent process before it terminates. Also, when a parent process is terminated, its child processes are terminated as well as the child processes cannot run if the parent processes are terminated.

Inter Process Communication (IPC)

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

1. Shared Memory
2. Message passing

The Figure 1 below shows a basic structure of communication between processes via the shared memory method and via the message passing method.

An operating system can implement both method of communication. First, we will discuss the shared memory methods of communication and then message passing. Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from another process. Process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

an example of communication between processes using shared memory method.

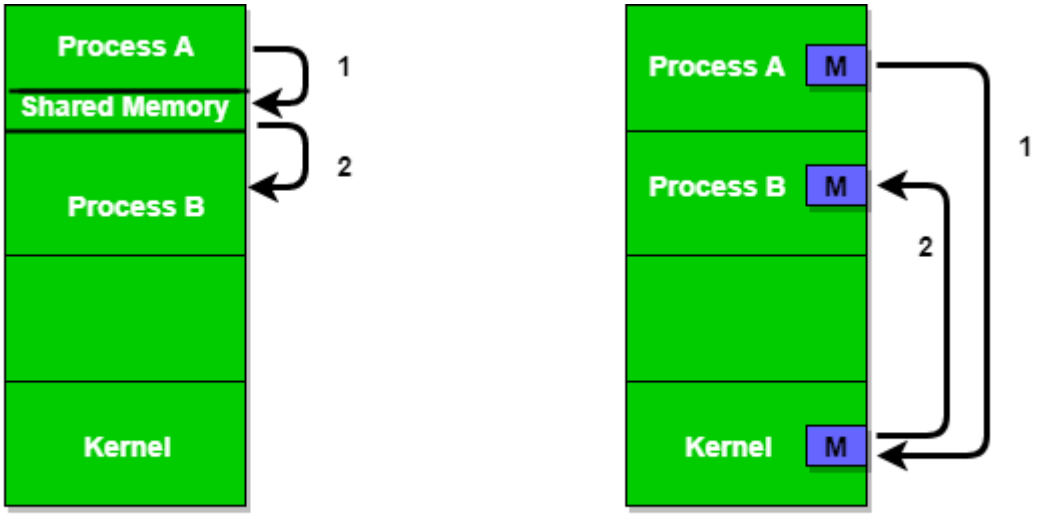


Figure 1 - Shared Memory and Message Passing

i) Shared Memory Method

Ex: Producer-Consumer problem

There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item. The two processes share a common space or memory location known as a buffer where the item produced by Producer is stored and from which the Consumer consumes the item, if needed. There are two versions of this problem: the first one is known as unbounded buffer problem in which Producer can keep on producing items and there is no limit on the size of the buffer, the second one is known as the bounded buffer problem in which Producer can produce up to a certain number of items before it starts waiting for Consumer to consume it. We will discuss the bounded buffer problem. First, the Producer and the Consumer will share some common memory, then producer will start producing items. If the total produced item is equal to the size of buffer, producer will wait to get it consumed by the Consumer. Similarly, the consumer will first check for the availability of the item. If no item is available, Consumer will wait for Producer to produce it. If there are items available, Consumer will consume it. The pseudo code to demonstrate is provided below:

Shared Data between the two Processes

```
#define buff_max 25
#define mod %
    struct item{
       // different member of the produced data
      // or consumed data
      ---------
   }
   // An array is needed for holding the items.
   // This is the shared place which will be
   // access by both process
   // item shared_buff [ buff_max ];
   // Two variables which will keep track of
   // the indexes of the items produced by producer
   // and consumer The free index points to
   // the next free index. The full index points to
   // the first full index.
   int free_index = 0;
   int full_index = 0;
```

Producer Process Code

```
item nextProduced;
   while(1){
```

```
    // check if there is no space
    // for production.
    // if so keep waiting.
    while((free_index+1) mod buff_max == full_index);
    shared_buff[free_index] = nextProduced;
    free_index = (free_index + 1) mod buff_max;
  }
```

Consumer Process Code

```
item nextConsumed;
  while(1){
    // check if there is an available
    // item  for consumption.
    // if not keep on waiting for
    // get them produced.
    while((free_index == full_index);

    nextConsumed = shared_buff[full_index];
    full_index = (full_index + 1) mod buff_max;
  }
```

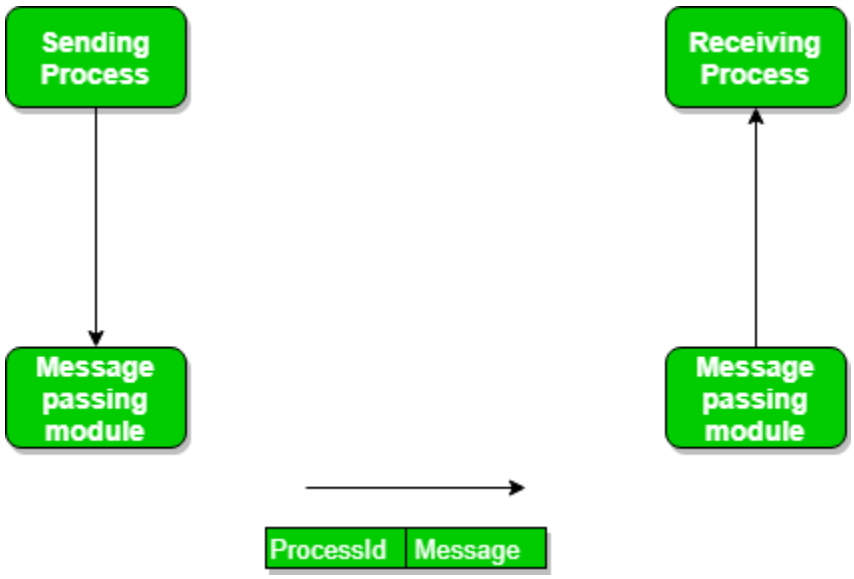
In the above code, the Producer will start producing again when the (free_index+1) mod buff max will be free because if it it not free, this implies that there are still items that can be consumed by the Consumer so there is no need to

produce more. Similarly, if free index and full index point to the same index, this implies that there are no items to consume.

ii) Messaging Passing Method

Now, We will start our discussion of the communication between processes via message passing. In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – **send**(message, destinaion) or **send**(message)
  – **receive**(message, host) or **receive**(message)



The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message can have two parts: **header and body.**
The **header part** is used for storing message type, destination id, source id,

message length, and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

Message Passing through Communication Link.

Direct and Indirect Communication link

Now, We will start our discussion about the methods of implementing communication link. While implementing the link, there are some questions which need to be kept in mind like :

1. How are links established?
2. Can a link be associated with more than two processes?
3. How many links can there be between every pair of communicating processes?
4. What is the capacity of a link? Is the size of a message that the link can accommodate fixed or variable?
5. Is a link unidirectional or bi-directional?

A link has some capacity that determines the number of messages that can reside in it temporarily for which every link has a queue associated with it which can be of zero capacity, bounded capacity, or unbounded capacity. In zero capacity, the sender waits until the receiver informs the sender that it has received the message. In non-zero capacity cases, a process does not know whether a message has been received or not after the send operation. For this, the sender must communicate with the receiver explicitly. Implementation of the link depends on the situation, it can be either a direct communication link or an in-directed communication link.

Direct Communication links are implemented when the processes uses a specific process identifier for the communication, but it is hard to identify the sender ahead of time.

For example: the print server.

In-direct Communication is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up.

Message Passing through Exchanging the Messages.

Synchronous and Asynchronous Message Passing:

A process that is blocked is one that is waiting for some event, such as a resource becoming available or the completion of an I/O operation. IPC is possible between the processes on same computer as well as on the processes running on different computer i.e. in networked/distributed system. In both cases, the process may or may not be blocked while sending a message or attempting to receive a message so message passing may be blocking or non-blocking. Blocking is considered **synchronous** and **blocking send** means the sender will be blocked until the message is received by receiver. Similarly, **blocking receive** has the receiver block until a message is available. Non-blocking is considered **asynchronous** and Non-blocking send has the sender sends the message and continue. Similarly, Non-blocking receive has the receiver receive a valid message or null. After a careful analysis, we can come to a conclusion that for a sender it is more natural to be non-blocking after message passing as there may be a need to send the message to different processes. However, the sender expects acknowledgement from the receiver in case the send fails. Similarly, it is more natural for a receiver to be blocking after issuing the receive as the information from the received message may be used for further execution. At the same time, if the message send keep on failing, the receiver will have to wait indefinitely. That is why we also consider the other possibility of message passing. There are basically three preferred combinations:

- Blocking send and blocking receive
- Non-blocking send and Non-blocking receive

- Non-blocking send and Blocking receive (Mostly used)

In Direct message passing, The process which want to communicate must explicitly name the recipient or sender of communication.
e.g. **send(p1, message)** means send the message to p1.
similarly, **receive(p2, message)** means receive the message from p2.
In this method of communication, the communication link gets established automatically, which can be either unidirectional or bidirectional, but one link can be used between one pair of the sender and receiver and one pair of sender and receiver should not possess more than one pair of links. Symmetry and asymmetry between sending and receiving can also be implemented i.e. either both process will name each other for sending and receiving the messages or only the sender will name receiver for sending the message and there is no need for receiver for naming the sender for receiving the message. The problem with this method of communication is that if the name of one process changes, this method will not work.

In Indirect message passing, processes use mailboxes (also referred to as ports) for sending and receiving messages. Each mailbox has a unique id and processes can communicate only if they share a mailbox. Link established only if processes share a common mailbox and a single link can be associated with many processes. Each pair of processes can share several communication links and these links may be unidirectional or bi-directional. Suppose two process want to communicate though Indirect message passing, the required operations are: create a mail box, use this mail box for sending and receiving messages, then destroy the mail box. The standard primitives used are: **send(A, message)** which means send the message to mailbox A. The primitive for the receiving the message also works in the same way e.g. **received (A, message)**. There is a problem in this mailbox implementation. Suppose there are more than two processes sharing the same mailbox and suppose the process p1 sends a message to the mailbox, which process will be the receiver? This can be solved by either enforcing that only two processes can share a single mailbox or enforcing that only one process is allowed to execute the receive at a given time or select any process randomly and notify

the sender about the receiver. A mailbox can be made private to a single sender/receiver pair and can also be shared between multiple sender/receiver pairs. Port is an implementation of such mailbox which can have multiple sender and single receiver. It is used in client/server applications (in this case the server is the receiver). The port is owned by the receiving process and created by OS on the request of the receiver process and can be destroyed either on request of the same receiver process or when the receiver terminates itself. Enforcing that only one process is allowed to execute the receive can be done using the concept of mutual exclusion. **Mutex mailbox** is create which is shared by n process. Sender is non-blocking and sends the message. The first process which executes the receive will enter in the critical section and all other processes will be blocking and will wait.

Now, lets discuss the Producer-Consumer problem using message passing concept. The producer places items (inside messages) in the mailbox and the consumer can consume an item when at least one message present in the mailbox. The code is given below:

Producer Code

```
void Producer(void){
    int item;
    Message m;
    while(1){
       receive(Consumer, &m);
       item = produce();
       build_message(&m , item ) ;
       send(Consumer, &m);
```

```
    }
  }
```

Consumer Code

```
filter_none
edit
play_arrow
brightness_4
void Consumer(void){
    int item;
    Message m;
    while(1){
        receive(Producer, &m);
        item = extracted_item();
        send(Producer, &m);
        consume_item(item);
    }
  }
```

Examples of IPC systems

1. Posix : uses shared memory method.
2. Mach : uses message passing
3. Windows XP : uses message passing using local procedural calls

Communication in client/server Architecture:

There are various mechanism:

- Pipe
- Socket
- Remote Procedural calls (RPCs)

The above three methods will be discussed in later articles as all of them are quite conceptual and deserve their own separate articles.

Operating Systems Client/Server Communication

Client/Server communication involves two components, namely a client and a server. They are usually multiple clients in communication with a single server. The clients send requests to the server and the server responds to the client requests.

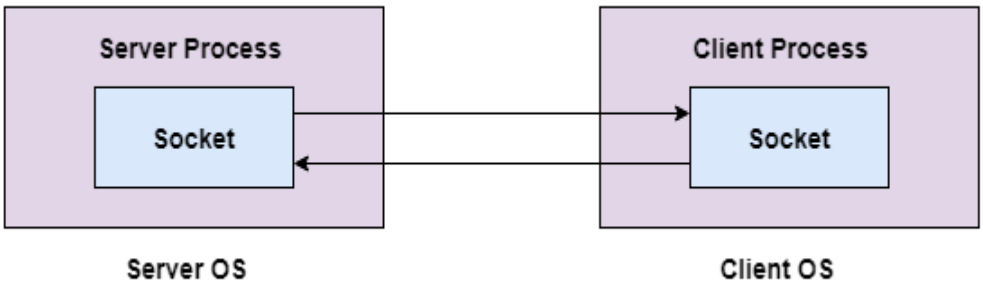
There are three main methods to client/server communication. These are given as follows −

Sockets

Sockets facilitate communication between two processes on the same machine or different machines. They are used in a client/server framework and consist of the IP address and port number. Many application protocols use sockets for data connection and data transfer between a client and a server.

Socket communication is quite low-level as sockets only transfer an unstructured byte stream across processes. The structure on the byte stream is imposed by the client and server applications.

A diagram that illustrates sockets is as follows −

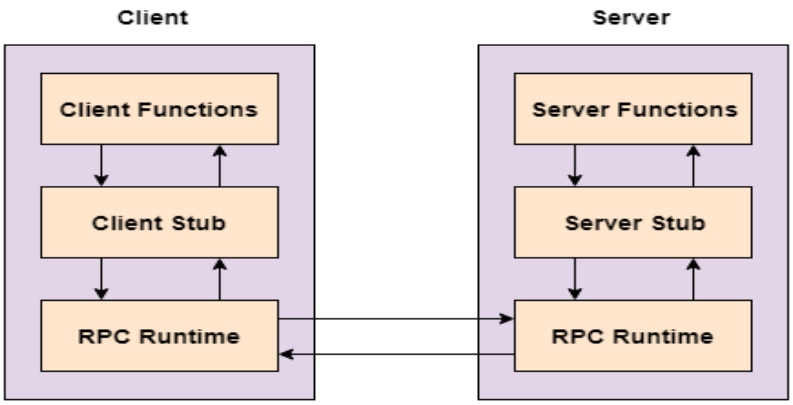


Remote Procedure Calls

These are interprocess communication techniques that are used for client-server based applications. A remote procedure call is also known as a subroutine call or a function call.

A client has a request that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client.

A diagram that illustrates remote procedure calls is given as follows −



Pipes

These are interprocess communication methods that contain two end points. Data is entered from one end of the pipe by a process and consumed from the other end by the other process.

The two different types of pipes are ordinary pipes and named pipes. Ordinary pipes only allow one way communication. For two way communication, two pipes are required. Ordinary pipes have a parent child relationship between the processes as the pipes can only be accessed by processes that created or inherited them.

Named pipes are more powerful than ordinary pipes and allow two way communication. These pipes exist even after the processes using them have terminated. They need to be explicitly deleted when not required anymore.

A diagram that demonstrates pipes are given as follows −



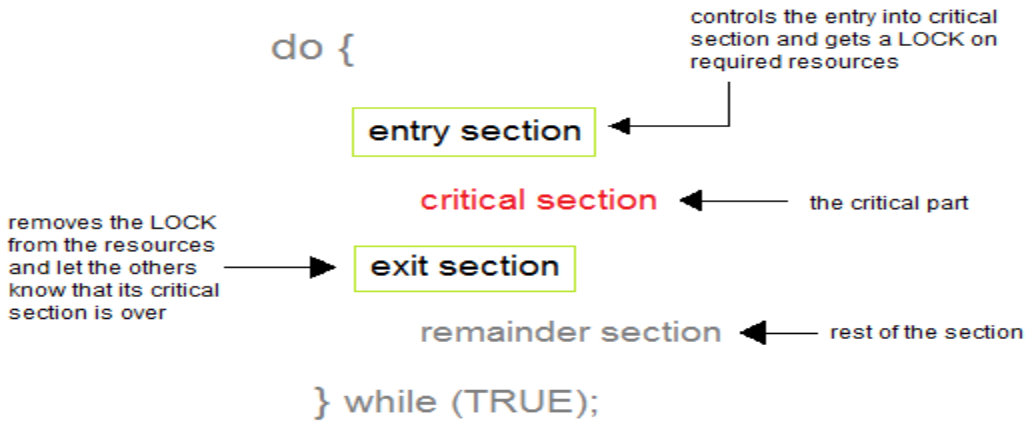
Process Synchronization

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

1. Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

2. Progress

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

3. Bounded Waiting

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

Synchronization Hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

Classical Problems of Synchronization

Semaphore can be used in other synchronization problems besides Mutual Exclusion.

Below are some of the classical problem depicting flaws of process synchronaization in systems where cooperating processes are present.

The following three problems:

1. Bounded Buffer (Producer-Consumer) Problem

2. Dining Philosophers Problem
3. The Readers Writers Problem

Bounded Buffer Problem

- This problem is generalised in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes.

Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer problem**.

- Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

Dining Philosophers Problem

- The dining philosopher's problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre, When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

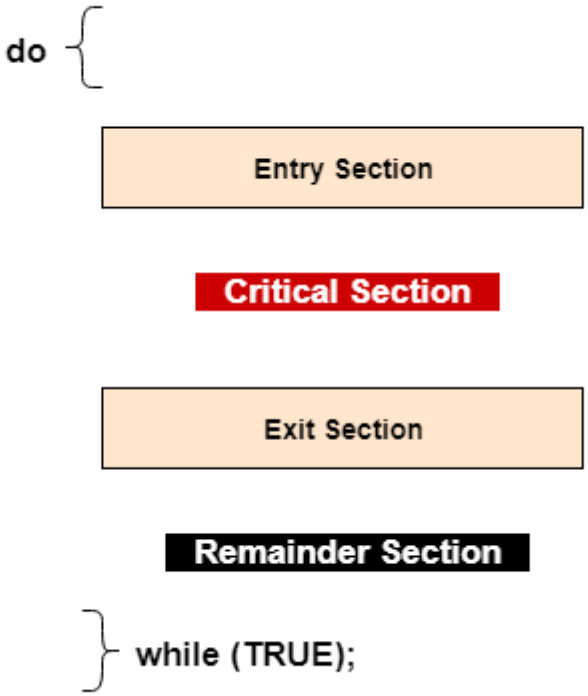
The Readers Writers Problem

- In this problem there are some processes(called **readers**) that only read the shared data, and never change it, and there are other processes(called **writers**) who may change the data in addition to reading, or instead of reading it.
- There are various type of readers-writers problem, most centred on relative priorities of readers and writers.

Critical Section Problem

The critical section is a code segment where the shared variables can be accessed. An atomic action is required in a critical section i.e. only one process can execute in its critical section at a time. All the other processes have to wait to execute in their critical sections.

A diagram that demonstrates the critical section is as follows −



In the above diagram, the entry section handles the entry into the critical section. It acquires the resources needed for execution by the process. The exit section handles the exit from the critical section. It releases the resources and also informs the other processes that the critical section is free.

Solution to the Critical Section Problem

The critical section problem needs a solution to synchronize the different processes. The solution to the critical section problem must satisfy the following conditions −

- **Mutual Exclusion**

Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

- **Progress**

Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

- **Bounded Waiting**

Bounded waiting means that each process must have a limited waiting time. Itt should not wait endlessly to access the critical section.

Peterson's solution

Peterson's solution provides a good algorithmic description of solving the critical-section problem and illustrates some of the complexities involved in designing software that addresses the requirements of mutual exclusion, progress, and bounded waiting.

```
do {
  flag[i] = true;
  turn = j;
  while (flag[j] && turn == j);
  /* critical section */
  flag[i] = false;
  /* remainder section */
}
```

```
while (true);
```

The structure of process Pi in Peterson's solution. This solution is restricted to two processes that alternate execution between their critical sections and remainder sections. The processes are numbered P0 and P1. We use Pj for convenience to denote the other process when Pi is present; that is, j equals 1 − I, Peterson's solution requires the two processes to share two data items −

int turn;

boolean flag[2];

The variable turn denotes whose turn it is to enter its critical section. I.e., if turn == i, then process Pi is allowed to execute in its critical section. If a process is ready to enter its critical section, the flag array is used to indicate that. For E.g., if flag[i] is true, this value indicates that Pi is ready to enter its critical section. With an explanation of these data structures complete, we are now ready to describe the algorithm shown in above. To enter the critical section, process Pi first sets flag[i] to be true and then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section, it can do so. Turn will be set to both i and j at roughly the same time, if both processes try to enter at the same time. Only one of these assignments will occur ultimately; the other will occur but will be overwritten immediately. The final value of turn determines which of the two processes is allowed to enter its critical section first. We now prove that this solution is correct. We need to show that −

- Mutual exclusion is preserved.
- The progress requirement is satisfied.
- The bounded-waiting requirement is met.

To prove 1, we note that each Pi enters its critical section only if either flag[j] == false or turn == i. Also note that, if both processes can be executing in their critical sections at the same time, then flag[0] == flag[1] == true. These two observations indicate that P0 and P1 could not have successfully executed their while

statements at about the same time, since the value of turn can be either 0 or 1 but cannot be both. Hence, one of the processes — say, Pj — must have successfully executed the while statement, whereas Pi had to execute at least one additional statement ("turn == j"). However, at that time, flag[j] == true and turn == j, and this condition will persist as long as Pj is in its critical section; as a result, mutual exclusion is preserved.

To prove properties 2 and 3, we note that if a process is stuck in the while loop with the condition flag[j] == true and turn == j, process Pi can be prevented from entering the critical section only; this loop is the only one possible. flag[j] will be == false, and Pi can enter its critical section if Pj is not ready to enter the critical section. If Pj has set, flag[j] = true and is also executing in its while statement, then either turn == i or turn == j. If turn == i, Pi will enter the critical section then. Pj will enter the critical section, If turn == j. Although once Pj exits its critical section, it will reset flag[j] to false, allowing Pi to enter its critical section. Pj must also set turn to i, if Pj resets flag[j] to true. Hence, since Pi does not change the value of the variable turn while executing the while statement, Pi will enter the critical section (progress) after at most one entry by Pj (bounded waiting).

Disadvantage

- Peterson's solution works for two processes, but this solution is best scheme in user mode for critical section.
- This solution is also a busy waiting solution so CPU time is wasted. So that **"SPIN LOCK"** problem can come. And this problem can come in any of the busy waiting solution.

Semaphores

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The definitions of wait and signal are as follows −

- **Wait**

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
wait(S)
{
  while (S<=0);
  S--;
}
```

- **Signal**

The signal operation increments the value of its argument S.

```
signal(S)
{
  S++;
}
```

Types of Semaphores

There are two main types of semaphores i.e. counting semaphores and binary semaphores. Details about these are given as follows −

- **Counting Semaphores**

These are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the resources are added, semaphore count automatically incremented and if the resources are removed, the count is decremented.

- **Binary Semaphores**

The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Advantages of Semaphores

Some of the advantages of semaphores are as follows −

- Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
- There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.
- Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

Disadvantages of Semaphores

Some of the disadvantages of semaphores are as follows −

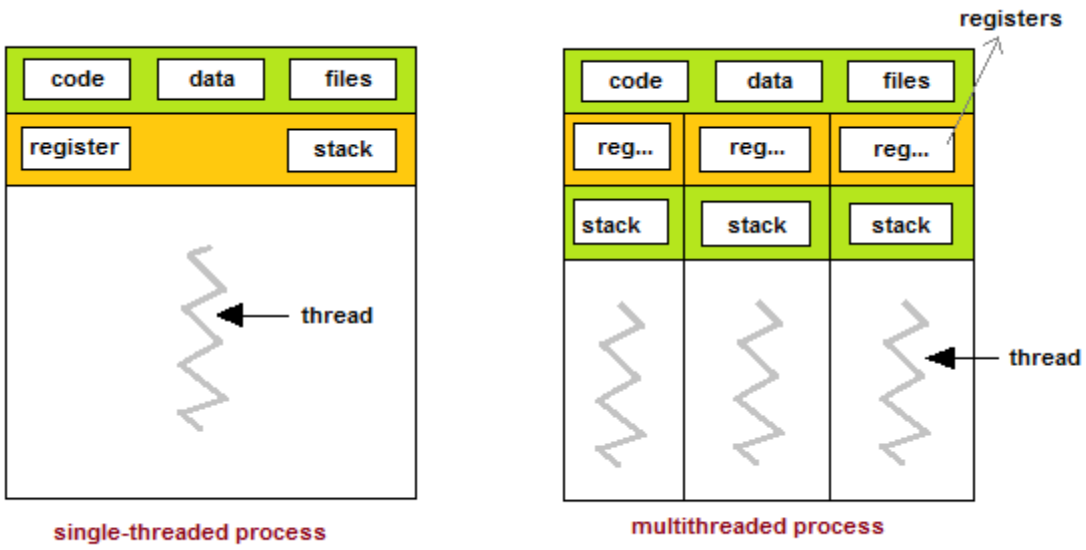
- Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
- Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
- Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

Threads

Thread is an execution unit which consists of its own program counter, a stack, and a set of registers. Threads are also known as Lightweight processes. Threads

are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel.

As each thread has its own independent resource for process execution, multpile processes can be executed parallely by increasing number of threads.



Types of Thread

There are two types of threads:

1. User Threads
2. Kernel Threads

User threads, are above the kernel and without kernel support. These are the threads that application programmers use in their programs.

Kernel threads are supported within the kernel of the OS itself. All modern OSs support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

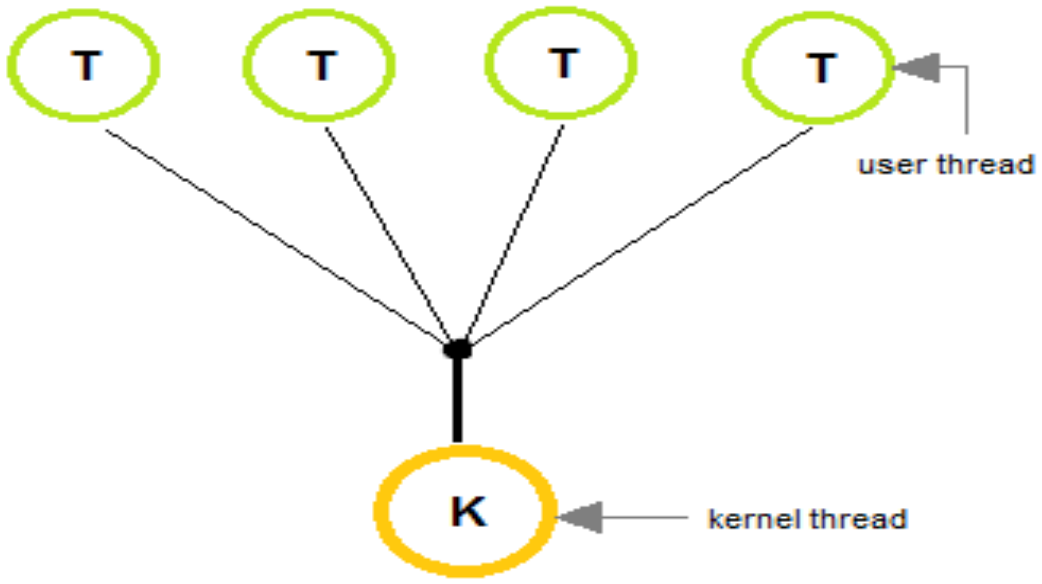
Multithreading Models

The user threads must be mapped to kernel threads, by one of the following strategies:

- Many to One Model
- One to One Model
- Many to Many Model

Many to One Model

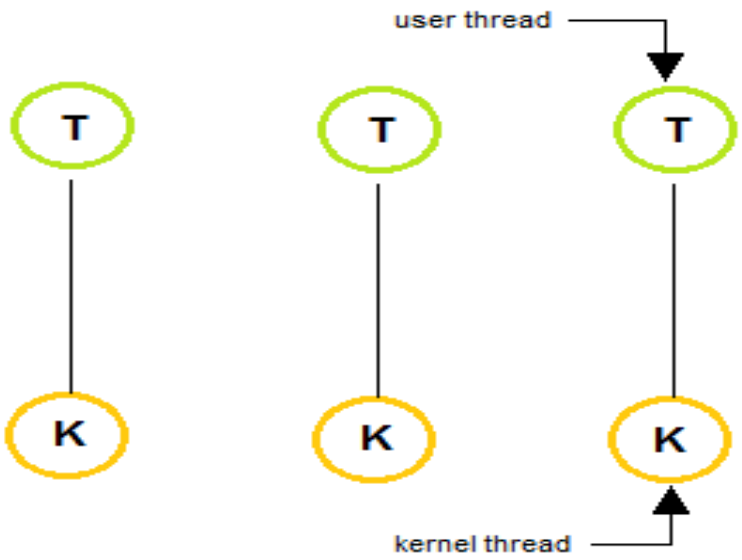
- In the **many to one** model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is efficient in nature.



One to One Model

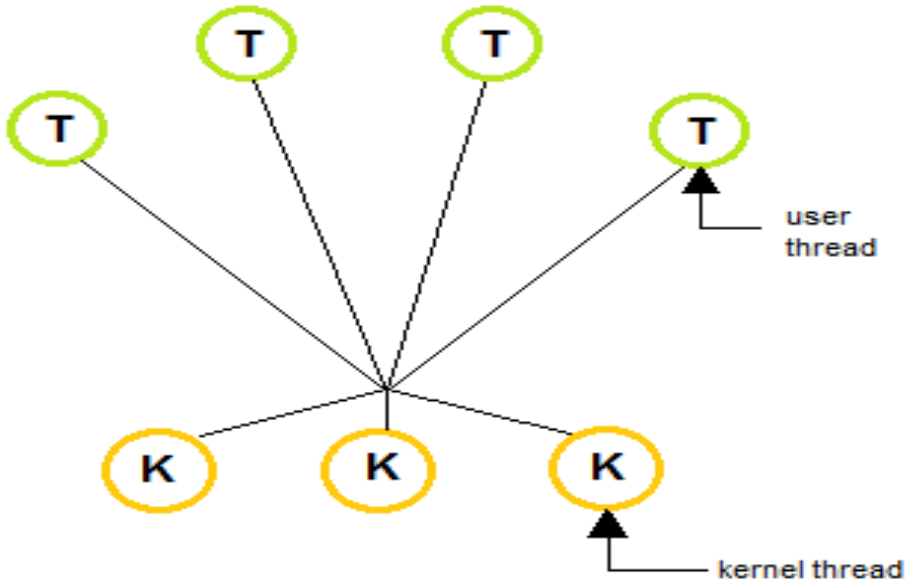
- The **one to one** model creates a separate kernel thread to handle each and every user thread.

- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.



Many to Many Model

- The **many to many** model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users can create any number of the threads.
- Blocking the kernel system calls does not block the entire process.
- Processes can be split across multiple processors.



What are Thread Libraries?

Thread libraries provide programmers with API for creation and management of threads.

Thread libraries may be implemented either in user space or in kernel space. The user space involves API functions implemented solely within the user space, with no kernel support. The kernel space involves system calls, and requires a kernel with thread library support.

Three types of Thread

1. **POSIX Pitheads**, may be provided as either a user or kernel library, as an extension to the POSIX standard.
2. **Win32 threads**, are provided as a kernel-level library on Windows systems.
3. **Java threads**: Since Java generally runs on a Java Virtual Machine, the implementation of threads is based upon whatever OS and hardware the JVM is running on, i.e. either Pitheads or Win32 threads depending on the system.

Benefits of Multithreading

1. Responsiveness
2. Resource sharing, hence allowing better utilization of resources.
3. Economy. Creating and managing threads becomes easier.
4. Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
5. Context Switching is smooth. Context switching refers to the procedure followed by CPU to change from one task to another.

Multithreading Issues

Below we have mentioned a few issues related to multithreading. Well, it's an old saying, All good things, come at a price.

Thread Cancellation

Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is **Asynchronous cancellation**, which terminates the target thread immediately. The other is **Deferred cancellation** allows the target thread to periodically check if it should be cancelled.

Signal Handling

Signals are used in UNIX systems to notify a process that a particular event has occurred. Now in when a Multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all, or a single thread.

fork() System Call

fork() is a system call executed in the kernel through which a process creates a copy of itself. Now the problem in Multithreaded process is, if one thread forks, will the entire process be copied or not?

Security Issues

Yes, there can be security issues because of extensive sharing of resources between multiple threads.

There are many other issues that you might face in a multithreaded process, but there are appropriate solutions available for them. Pointing out some issues here was just to study both sides of the coin.

Multicore programming

Multicore programming helps to create concurrent systems for deployment on multicore processor and multiprocessor systems. A multicore processor system is basically a single processor with multiple execution cores in one chip. It has multiple processors on the motherboard or chip. A Field-Programmable Gate Array (FPGA) is might be included in a multiprocessor system. A FPGA is an integrated circuit containing an array of programmable logic blocks and a hierarchy of reconfigurable interconnects. Input data is processed by to produce outputs. It can be a processor in a multicore or multiprocessor system, or a FPGA.

The multicore programming approach has following advantages −

- Multicore and FPGA processing helps to increase the performance of an embedded system.
- Also helps to achieve scalability, so the system can take advantage of increasing numbers of cores and FPGA processing power over time.

Concurrent systems that we create using multicore programming have multiple tasks executing in parallel. This is known as concurrent execution. When multiple parallel tasks are executed by a processor, it is known as multitasking. A CPU scheduler, handles the tasks that execute in parallel. The CPU implements tasks using operating system threads. So that tasks can execute independently but have some data transfer between them, such as data transfer between a data acquisition module and controller for the system. Data transfer occurs when there is a data dependency.

Implicit Threading and Language-based threads

Implicit Threading

One way to address the difficulties and better support the design of multithreaded applications is to transfer the creation and management of threading from application developers to compilers and run-time libraries. This, termed implicit threading, is a popular trend today.

Implicit threading is mainly the use of libraries or other language support to hide the management of threads. The most common implicit threading library is OpenMP, in context of C.

OpenMP is a set of compiler directives as well as an API for programs written in C, C++, or FORTRAN that provides support for parallel programming in shared-memory environments. OpenMP identifies parallel regions as blocks of code that may run in parallel. Application developers insert compiler directives into their code at parallel regions, and these directives instruct the OpenMP run-time library to execute the region in parallel. The following C program illustrates a compiler directive above the parallel region containing the printf() statement:

Example

```
#include <omp.h>

#include <stdio.h>

int main(int argc, char *argv[]){

  /* sequential code */

  #pragma omp parallel{

     printf("I am a parallel region.");

  }

  /* sequential code */

  return 0;

}
```

Output

```
I am a parallel region.
```

When OpenMP encounters the directive

```
#pragma omp parallel
```

It creates as many threads which are processing cores in the system. Thus, for a dual-core system, two threads are created, for a quad-core system, four are created; and so forth. Then all the threads simultaneously execute the parallel region. When each thread exits the parallel region, it is terminated. OpenMP provides several additional directives for running code regions in parallel, including parallelizing loops.

In addition to providing directives for parallelization, OpenMP allows developers to choose among several levels of parallelism. Eg, they can set the number of threads manually. It also allows developers to identify whether data are shared between threads or are private to a thread. OpenMP is available on several open-source and commercial compilers for Linux, Windows, and Mac OS X systems.

Grand Central Dispatch (GCD)

Grand Central Dispatch (GCD)—a technology for Apple's Mac OS X and iOS operating systems—is a combination of extensions to the C language, an API, and a run-time library that allows application developers to spot sections of code to run in parallel. Like OpenMP, GCD also manages most of the details of threading. It identifies extensions to the C and C++ languages known as blocks. A block is simply a self-contained unit of work. It is specified by a caret ˆ inserted in front of a pair of braces { }. A simple example of a block is shown below −

```
{
  ˆprintf("This is a block");
}
```

It schedules blocks for run-time execution by placing them on a dispatch queue. When GCD removes a block from a queue, it assigns the block to an available thread from the thread pool it manages. It identifies two types of dispatch queues: serial and concurrent. Blocks placed on a serial queue are removed in FIFO order. Once a block has been removed from the queue, it must complete execution before another block is removed. Each process has its own serial queue

(known as main queue). Developer can create additional serial queues that are local to particular processes. Serial queues are useful for ensuring the sequential execution of several tasks. Blocks placed on a concurrent queue are also removed in FIFO order, but several blocks may be removed at a time, thus allowing multiple blocks to execute in parallel. There are three system-wide concurrent dispatch queues, and they are distinguished according to priority: low, default, and high. Priorities represent an estimation of the relative importance of blocks. Quite simply, blocks with a higher priority should be placed on the high priority dispatch queue. The following code segment illustrates obtaining the default-priority concurrent queue and submitting a block to the queue using the dispatch async() function:

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch async(queue, ^{ printf("This is a block."); });
```

Internally, GCD's thread pool is composed of POSIX threads. GCD actively manages the pool, allowing the number of threads to grow and shrink according to application demand and system capacity.

Threads as Objects

In alternative languages, ancient object-oriented languages give explicit multithreading support with threads as objects. In these forms of languages, classes area written to either extend a thread class or implement a corresponding interface. This style resembles the Pthread approach, because the code is written with explicit thread management. However, the encapsulation of information inside the classes and extra synchronization options modify the task.

Java Threads

Java provides a Thread category and a Runnable interface that can be used. Each need to implement a public void run() technique that defines the entry purpose of the thread. Once an instance of the object is allotted, the thread is started by invoking the start() technique on that. Like with Pthreads, beginning the thread is asynchronous, that the temporal arrangement of the execution is non-deterministic.

Python Threads

Python additionally provides two mechanisms for multithreading. One approach is comparable to the Pthread style, wherever a function name is passed to a library method thread.start_new_thread(). This approach is very much and lacks the flexibility to join or terminate the thread once it starts. A additional flexible technique is to use the threading module to outline a class that extends threading. Thread. almost like the Java approach, the category should have a run() method that gives the thread's entry purpose. Once an object is instantiated from this category, it can be explicitly started and joined later.

Concurrency as Language Design

Newer programming languages have avoided race condition by building assumptions of concurrent execution directly into the language style itself. As an example, Go combines a trivial implicit threading technique (goroutines) with channels, a well-defined style of message-passing communication. Rust adopts a definite threading approach the same as Pthreads. However, Rust has terribly strong memory protections that need no extra work by the software engineer.

Goroutines

The Go language includes a trivial mechanism for implicit threading: place the keyword go before a call. The new thread is passed an association to a message-passing channel. Then, the most thread calls success := <-messages, that performs a interference scan on the channel. Once the user has entered the right guess of seven, the keyboard auditor thread writes to the channel, permitting the most thread to progress.

Channels and goroutines are core components of the Go language, that was designed beneath the belief that almost all programs would be multithreaded. This style alternative streamlines the event model, permitting the language itself up-to-date the responsibility for managing the threads and programing.

Rust Concurrency

Another language is Rust that has been created in recent years, with concurrency as a central design feature. The following example illustrates the use of thread::spawn() to create a new thread, which can later be joined by invoking join() on it. The argument to thread::spawn() beginning at the || is known as a

closure, which can be thought of as an anonymous function. That is, the child thread here will print the value of a.

Example

```
use std::thread;

fn main() {

  /* Initialize a mutable variable a to 7 */

  let mut a = 7;

  /* Spawn a new thread */

  let child_thread = thread::spawn(move || {

    /* Make the thread sleep for one second, then print a */

    a -= 1;

    println!("a = {}", a)

  });

  /* Change a in the main thread and print it */

  a += 1;

  println!("a = {}", a);

  /* Join the thread and print a again */

  child_thread.join();

}
```

However, there is a subtle point in this code that is central to Rust's design. Within the new thread (executing the code in the closure), the a variable is distinct from the a in other parts of this code. It enforces a very strict memory

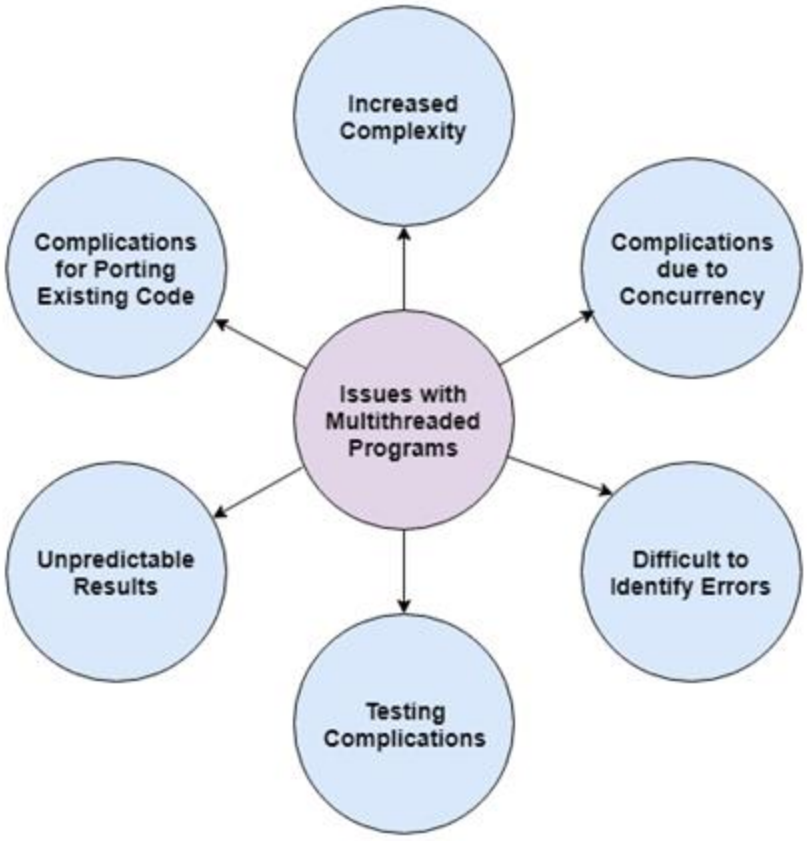
model (known as "ownership") which prevents multiple threads from accessing the same memory. In this example, the move keyword indicates that the spawned thread will receive a separate copy of a for its own use. Regardless of the scheduling of the two threads, the main and child threads cannot interfere with each other's modifications of a, because they are distinct copies. It is not possible for the two threads to share access to the same memory.

Threading Issues

Multithreaded programs allow the execution of multiple parts of a program at the same time. These parts are known as threads and are lightweight processes available within the process.

Threads improve the application performance using parallelism. They share information like data segment, code segment files etc. with their peer threads while they contain their own registers, stack, counter etc.

Some of the issues with multithreaded programs are as follows −



one by one −

- **Increased Complexity** − Multithreaded processes are quite complicated. Coding for these can only be handled by expert programmers.
- **Complications due to Concurrency** − It is difficult to handle concurrency in multithreaded processes. This may lead to complications and future problems.
- **Difficult to Identify Errors**− Identification and correction of errors is much more difficult in multithreaded processes as compared to single threaded processes.
- **Testing Complications**− Testing is a complicated process i multithreaded programs as compared to single threaded programs. This is because defects can be timing related and not easy to identify.
- **Unpredictable results**− Multithreaded programs can sometimes lead to unpredictable results as they are essentially multiple parts of a program that are running at the same time.
- **Complications for Porting Existing Code** − A lot of testing is required for porting existing code in multithreading. Static variables need to be removed and any code or function calls that are not thread safe need to be replaced.

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

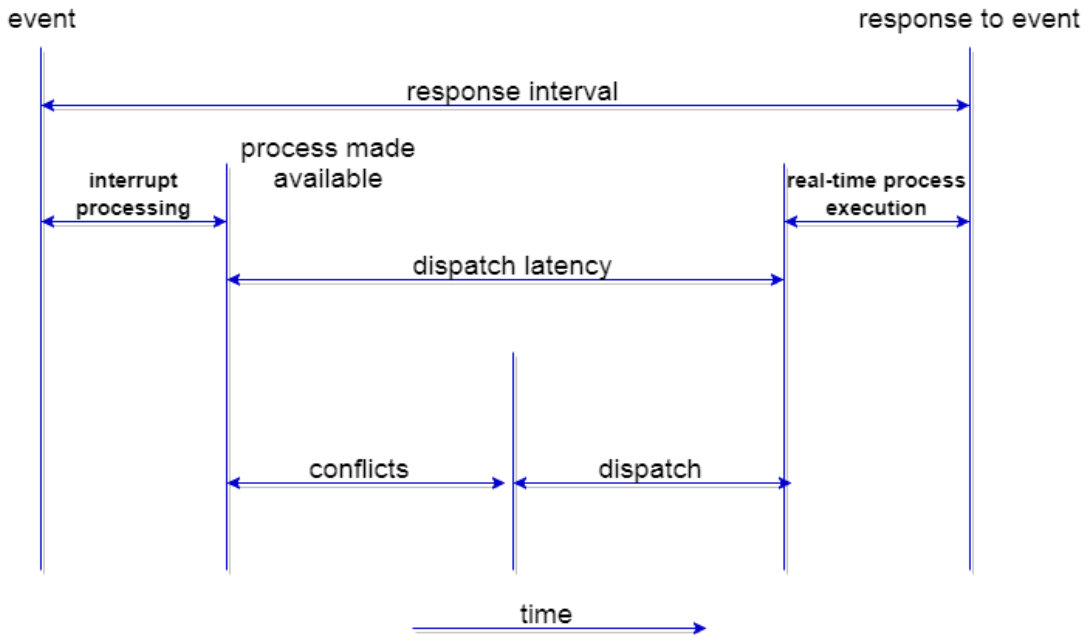
Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

CPU Scheduling: Dispatcher

Another component involved in the CPU scheduling function is the **Dispatcher**. The dispatcher is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program from where it left last time.

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time taken by the dispatcher to stop one process and start another process is known as the **Dispatch Latency**. Dispatch Latency can be explained using the below figure:



Types of CPU Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
4. When a process **terminates**.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

It is the only method that can be used on certain hardware platforms, because It does not require the special hardware(for example: a timer) needed for preemptive scheduling.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task

although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

CPU Scheduling: Scheduling Criteria

There are many different criterias to check when considering the **"best"** scheduling algorithm, they are:

CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilisation, computer scientists have defined some algorithms, they are:

1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

First Come First Serve Scheduling

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

- First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.
- It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.

- A perfect real life example of FCFS scheduling is **buying tickets at ticket counter**.

Calculating Average Waiting Time

For every scheduling algorithm, **Average waiting time** is a crucial parameter to judge it's performance.

AWT or Average waiting time is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution.

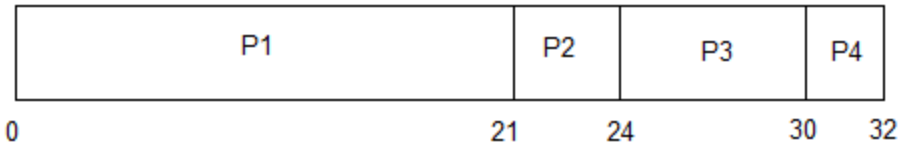
Lower the Average Waiting Time, better the scheduling algorithm.

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with **Arrival Time** 0, and given **Burst Time**, let's find the average waiting time using the FCFS scheduling algorithm.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2



The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

The average waiting time will be 18.75 ms

For the above given proccesses, first **P1** will be provided with the CPU resources,

- Hence, waiting time for **P1** will be 0
- **P1** requires 21 ms for completion, hence waiting time for **P2** will be 21 ms
- Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**, which will be (21 + 3) ms = 24 ms.
- For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

The **GANTT chart** above perfectly represents the waiting time for each process.

Problems with FCFS Scheduling

Below we have a few shortcomings or problems with the FCFS scheduling algorithm:

1. It is **Non Pre-emptive** algorithm, which means the **process priority** doesn't matter.

If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.

2. Not optimal Average Waiting Time.
3. Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource(CPU, I/O etc) utilization.

What is Convoy Effect?

Convoy Effect is a situation where many processes, who need to use a resource for short time are blocked by one process holding that resource for a long time.

This essentially leads to poort utilization of resources and hence poor performance.

Completion Time: Time taken for the execution to complete, starting from arrival time.

Turn Around Time: Time taken to complete after arrival. In simple words, it is the difference between the Completion time and the Arrival time.

Waiting Time: Total time the process has to wait before it's execution begins. It is the difference between the Turn Around time and the Burst time of the process.

Shortest Job First(SJF) Scheduling

Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

- This is the best approach to minimize waiting time.
- This is used in Batch Systems.
- It is of two types:
    1. Non Pre-emptive
    2. Pre-emptive
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

Non Pre-emptive Shortest Job First

Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2



In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

As we can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the average waiting time comes out 4.5 ms.

Problem with Non Pre-emptive SJF

If the **arrival time** for processes are different, which means all the processes are not available in the ready queue at time 0, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.

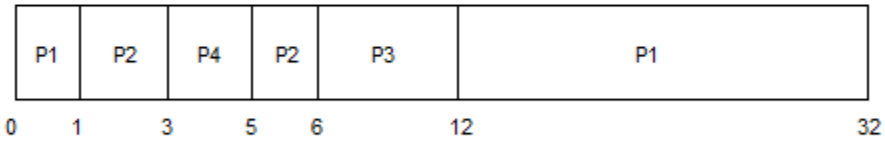
This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

As you can see in the **GANTT chart** above, as **P1** arrives first, hence it's execution starts immediately, but just after 1 ms, process **P2** arrives with a **burst time** of 3 ms which is less than the burst time of **P1**, hence the process **P1**(1 ms done, 20 ms left) is preemptied and process **P2** is executed.

As **P2** is getting executed, after 1 ms, **P3** arrives, but it has a burst time greater than that of **P2**, hence execution of **P2** continues. But after another millisecond, **P4** arrives with a burst time of 2 ms, as a result **P2**(2 ms done, 1 ms left) is preemptied and **P4** is executed.

After the completion of **P4**, process **P2** is picked up and finishes, then **P2** will get executed and at last **P1**.

The Pre-emptive SJF is also known as **Shortest Remaining Time First**, because at any given point of time, the job with the shortest remaining time is executed first.

Priority CPU Scheduling

In the Shortest Job First scheduling algorithm, the priority of a process is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process.

In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.

Processes with same priority are executed in FCFS manner.

The priority of process, when internally defined, can be decided based on **memory requirements**, **time limits** ,**number of open files**, **ratio of I/O burst to CPU burst** etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, makrte factor etc.

Types of Priority Scheduling Algorithm

Priority scheduling can be of two types:

1. **Preemptive Priority Scheduling**: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is

preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution.

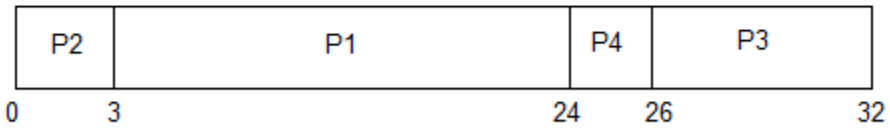
2. **Non-Preemptive Priority Scheduling**: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

Example of Priority Scheduling Algorithm

Consider the below table fo processes with their respective CPU burst times and the priorities.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,



The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

As you can see in the GANTT chart that the processes are given CPU time just on the basis of the priorities.

Problem with Priority Scheduling Algorithm

In priority scheduling algorithm, the chances of **indefinite blocking** or **starvation**.

A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently.

But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.

In 1973, when the IBM 7904 machine was shut down at MIT, a low-priority process was found which was submitted in 1967 and had not yet been run.

Using Aging Technique with Priority Scheduling

To prevent starvation of any process, we can use the concept of **aging** where we keep on increasing the priority of low-priority process based on the its waiting time.

For example, if we decide the aging factor to be **0.5** for each day of waiting, then if a process with priority **20**(which is comparitively low priority) comes in the ready queue. After one day of waiting, its priority is increased to **19.5** and so on.

Doing so, we can ensure that no process will have to wait for indefinite time for getting CPU time for processing.

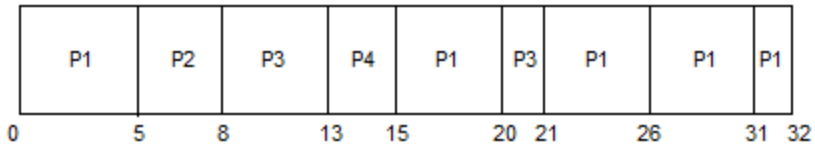
Round Robin Scheduling

- A fixed time is allotted to each process, called **quantum**, for execution.
- Once a process is executed for given time period that process is preemptied and other process executes for given time period.
- Context switching is used to save states of preemptied processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2



The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

Multilevel Queue Scheduling

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

For example: A common division is made between foreground(or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

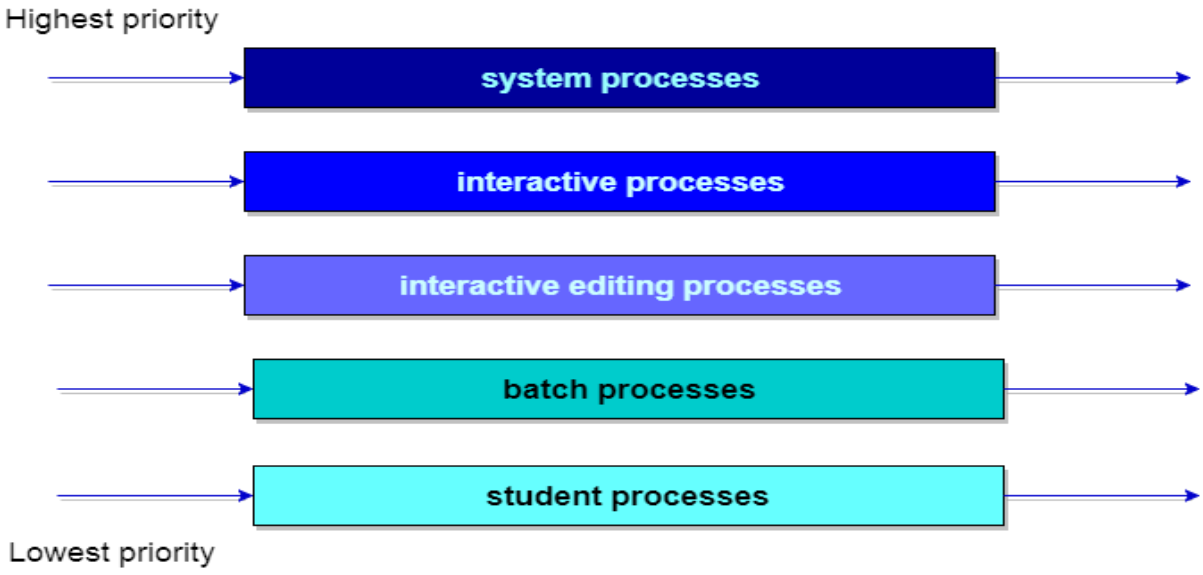
For example: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. **For example:** The foreground queue may have absolute priority over the background queue.

An example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

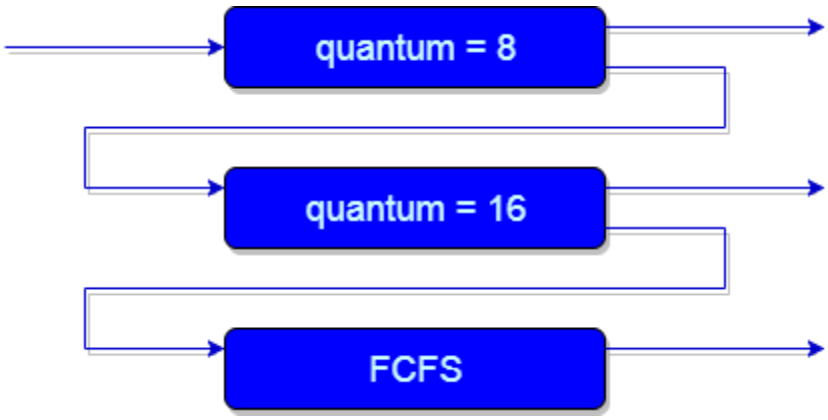
Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.



Multilevel Feedback Queue Scheduling

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.



An example of a multilevel feedback queue can be seen in the below figure.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.

- The method used to determine which queue a process will enter when that process needs service.

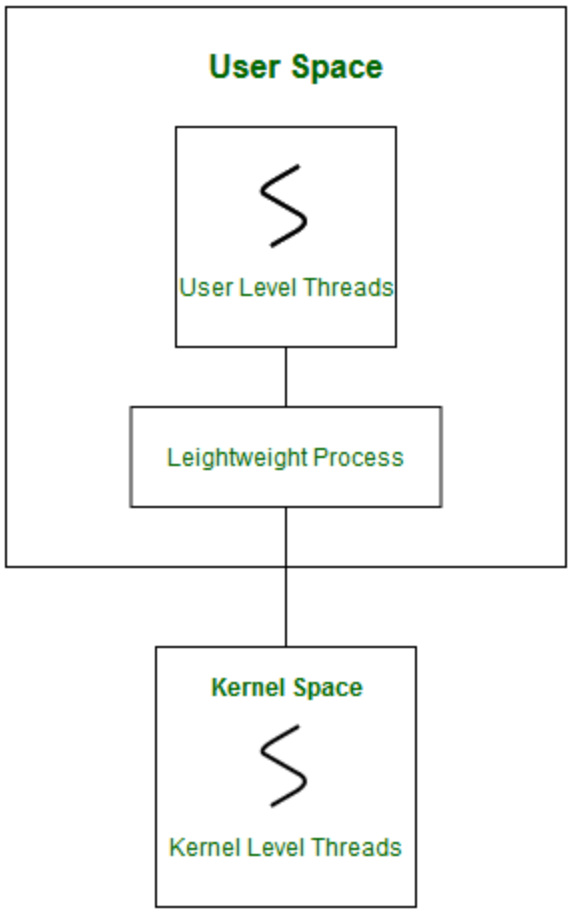
The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**.

Thread Scheduling

Scheduling of threads involves two boundary scheduling,

- Scheduling of user level threads (ULT) to kernel level threads (KLT) via leightweight process (LWP) by the application developer.
- Scheduling of kernel level threads by the system scheduler to perform different unique os functions.

Leightweight Process (LWP) :
Light-weight process are threads in the user space that acts as an interface for the ULT to access the physical CPU resources. Thread library schedules which thread of a process to run on which LWP and how long. The number of LWP created by the thread library depends on the type of application. In the case of an I/O bound application, the number of LWP depends on the number of user-level threads. This is because when an LWP is blocked on an I/O operation, then to invoke the other ULT the thread library needs to create and schedule another LWP. Thus, in an I/O bound application, the number of LWP is equal to the number of the ULT. In the case of a CPU bound application, it depends only on the application. Each LWP is attached to a separate kernel-level thread.



In real-time, the first boundary of thread scheduling is beyond specifying the scheduling policy and the priority. It requires two controls to be specified for the User level threads: Contention scope, and Allocation domain. These are explained as following below.

1. Contention Scope :

The word contention here refers to the competition or fight among the User level threads to access the kernel resources. Thus, this control defines the extent to which contention takes place. It is defined by the application developer using the thread library. Depending upon the extent of contention it is classified as **Process Contention Scope** and **System Contention Scope**.

1. **Process Contention Scope (PCS) –**
   The contention takes place among threads **within a same process**. The thread library schedules the high-prioritized PCS thread to access the

resources via available LWPs (priority as specified by the application developer during thread creation).

2. **System Contention Scope (SCS) –**
   The contention takes place among **all threads in the system**. In this case, every SCS thread is associated to each LWP by the thread library and are scheduled by the system scheduler to access the kernel resources.
   In LINUX and UNIX operating systems, the POSIX Pthread library provides a function Pthread_attr_setscope to define the type of contention scope for a thread during its creation.

```
int Pthread_attr_setscope(pthread_attr_t *attr, int scope)
```

   The first parameter denotes to which thread within the process the scope is defined.
   The second parameter defines the scope of contention for the thread pointed. It takes two values.

```
PTHREAD_SCOPE_SYSTEM

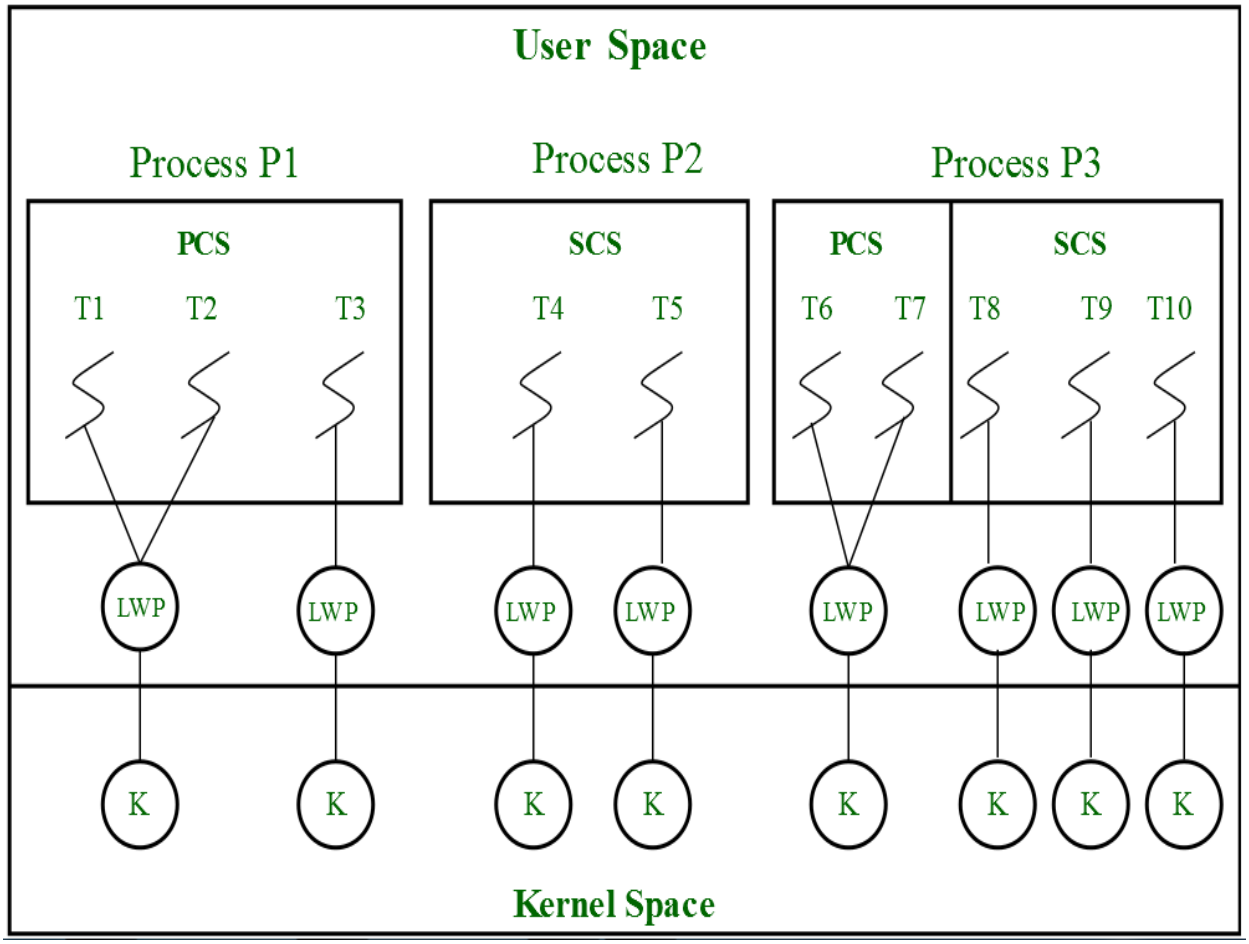
PTHREAD_SCOPE_PROCESS
```

   If the scope value specified is not supported by the system, then the function returns ENOTSUP.

2. Allocation Domain :

The allocation domain is **a set of one or more resources** for which a thread is competing. In a multicore system, there may be one or more allocation domains where each consists of one or more cores. One ULT can be a part of one or more allocation domain. Due to this high complexity in dealing with hardware and software architectural interfaces, this control is not specified. But by default, the multicore system will have an interface that affects the allocation domain of a thread.

Consider a scenario, an operating system with three process P1, P2, P3 and 10 user level threads (T1 to T10) with a single allocation domain. 100% of CPU resources will be distributed among all the three processes. The amount of CPU resources allocated to each process and to each thread depends on the contention scope, scheduling policy and priority of each thread defined by the application developer using thread library and also depends on the system scheduler. These User level threads are of a different contention scope.



In this case, the contention for allocation domain takes place as follows,

1. **Process P1:**
   All PCS threads T1, T2, T3 of Process P1 will compete among themselves. The PCS threads of the same process can share one or more LWP. T1 and T2 share an LWP and T3 are allocated to a separate LWP. Between T1 and T2 allocation of kernel resources via LWP is based on preemptive priority scheduling by the thread library. A Thread with a high priority will preempt low priority threads. Whereas, thread T1 of process p1 cannot preempt thread T3 of process p3 even if the priority of T1 is greater than the priority of T3. If the priority is equal, then the allocation of ULT to available LWPs is based on the scheduling policy of threads by the system scheduler(not by thread library, in this case).
2. **Process P2:**
   Both SCS threads T4 and T5 of process P2 will compete with processes P1 as a whole and with SCS threads T8, T9, T10 of process P3. The system scheduler will schedule the kernel resources among P1, T4, T5, T8, T9, T10,

and PCS threads (T6, T7) of process P3 considering each as a separate process. Here, the Thread library has no control of scheduling the ULT to the kernel resources.

3. **Process P3:**
   Combination of PCS and SCS threads. Consider if the system scheduler allocates 50% of CPU resources to process P3, then 25% of resources is for process scoped threads and the remaining 25% for system scoped threads. The PCS threads T6 and T7 will be allocated to access the 25% resources based on the priority by the thread library. The SCS threads T8, T9, T10 will divide the 25% resources among themselves and access the kernel resources via separate LWP and KLT. The SCS scheduling is by the system scheduler.

Note:

For every system call to access the kernel resources, a Kernel Level thread is created and associated to separate LWP by the system scheduler.

Number of Kernel Level Threads = Total Number of LWP

Total Number of LWP = Number of LWP for SCS + Number of LWP for PCS

Number of LWP for SCS = Number of SCS threads

Number of LWP for PCS = Depends on application developer

Here,

Number of SCS threads = 5

Number of LWP for PCS = 3

Number of SCS threads = 5

Number of LWP for SCS = 5

Total Number of LWP = 8 (=5+3)

Number of Kernel Level Threads = 8

Advantages of PCS over SCS :

- If all threads are PCS, then context switching, synchronization, scheduling everything takes place within the userspace. This reduces system calls and achieves better performance.
- PCS is cheaper than SCS.

- PCS threads share one or more available LWPs. For every SCS thread, a separate LWP is associated.For every system call, a separate KLT is created.
- The number of KLT and LWPs created highly depends on the number of SCS threads created. This increases the kernel complexity of handling scheduling and synchronization. Thereby, results in a limitation over SCS thread creation, stating that, the number of SCS threads to be smaller than the number of PCS threads.
- If the system has more than one allocation domain, then scheduling and synchronization of resources becomes more tedious. Issues arise when an SCS thread is a part of more than one allocation domain, the system has to handle n number of interfaces.

The second boundary of thread scheduling involves CPU scheduling by the system scheduler. The scheduler considers each kernel-level thread as a separate process and provides access to the kernel resources.

Multiple-Processor Scheduling

In multiple-processor scheduling **multiple CPU's** are available and hence **Load Sharing** becomes possible. However multiple processor scheduling is more **complex** as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

Approaches to Multiple-Processor Scheduling –

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the **Master Server** and the other processors executes only the **user code**. This is simple and reduces the need of data sharing. This entire scenario is called **Asymmetric Multiprocessing**.

A second approach uses **Symmetric Multiprocessing** where each processor is **self scheduling**. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

Processor Affinity –

Processor Affinity means a processes has an **affinity** for the processor on which it is currently running.
When a process runs on a specific processor there are certain effects on the cache memory. The data most recently accessed by the process populate the cache for the processor and as a result successive memory access by the process are often satisfied in the cache memory. Now if the process migrates to another processor, the contents of the cache memory must be invalidated for the first processor and the cache for the second processor must be repopulated. Because of the high cost of invalidating and repopulating caches, most of the SMP(symmetric multiprocessing) systems try to avoid migration of processes from one processor to another and try to keep a process running on the same processor. This is known as **PROCESSOR AFFINITY**.

There are two types of processor affinity:

1. **Soft Affinity –** When an operating system has a policy of attempting to keep a process running on the same processor but not guaranteeing it will do so, this situation is called soft affinity.
2. **Hard Affinity –** Hard Affinity allows a process to specify a subset of processors on which it may run. Some systems such as Linux implements soft affinity but also provide some system calls like sched_setaffinity() that supports hard affinity.

Load Balancing –

Load Balancing is the **phenomena** which keeps
the **workload** evenly **distributed** across all processors in an SMP system. Load balancing is necessary only on systems where each processor has its own private queue of process which are eligible to execute. Load balancing is unnecessary because once a processor becomes idle it immediately extracts a runnable process from the common run queue. On SMP(symmetric multiprocessing), it is important to keep the workload balanced among all processors to fully utilize the

benefits of having more than one processor else one or more processor will sit idle while other processors have high workloads along with lists of processors awaiting the CPU.

There are two general approaches to load balancing :

1. **Push Migration –** In push migration a task routinely checks the load on each processor and if it finds an imbalance then it evenly distributes load on each processors by moving the processes from overloaded to idle or less busy processors.
2. **Pull Migration –** Pull Migration occurs when an idle processor pulls a waiting task from a busy processor for its execution.

Multicore Processors –

In multicore processors **multiple processor** cores are places on the same physical chip. Each core has a register set to maintain its architectural state and thus appears to the operating system as a separate physical processor. **SMP systems** that use multicore processors are faster and consume **less power** than systems in which each processor has its own physical chip.

However multicore processors may **complicate** the scheduling problems. When processor accesses memory then it spends a significant amount of time waiting for the data to become available. This situation is called **MEMORY STALL**. It occurs for various reasons such as cache miss, which is accessing the data that is not in the cache memory. In such cases the processor can spend upto fifty percent of its time waiting for data to become available from the memory. To solve this problem recent hardware designs have implemented multithreaded processor cores in which two or more hardware threads are assigned to each core. Therefore if one thread stalls while waiting for the memory, core can switch to another thread.

There are two ways to multithread a processor :

1. **Coarse-Grained Multithreading –** In coarse grained multithreading a thread executes on a processor until a long latency event such as a memory stall occurs, because of the delay caused by the long latency event, the processor must switch to another thread to begin execution. The cost of switching between threads is high as the instruction pipeline must be terminated before the other thread can begin execution on the processor core. Once this new thread begins execution it begins filling the pipeline with its instructions.
2. **Fine-Grained Multithreading –** This multithreading switches between threads at a much finer level mainly at the boundary of an instruction cycle. The architectural design of fine grained systems include logic for thread switching and as a result the cost of switching between threads is small.

Virtualization and Threading –

In this type of **multiple-processor** scheduling even a single CPU system acts like a multiple-processor system. In a system with Virtualization, the virtualization presents one or more virtual CPU to each of virtual machines running on the system and then schedules the use of physical CPU among the virtual machines. Most virtualized environments have one host operating system and many guest operating systems. The host operating system creates and manages the virtual machines. Each virtual machine has a guest operating system installed and applications run within that guest.Each guest operating system may be assigned for specific use cases,applications or users including time sharing or even real-time operation. Any guest operating-system scheduling algorithm that assumes a certain amount of progress in a given amount of time will be negatively impacted by the virtualization. A time sharing operating system tries to allot 100 milliseconds to each time slice to give users a reasonable response time. A given 100 millisecond time slice may take much more than 100 milliseconds of virtual CPU time. Depending on how busy the system is, the time slice may take a second or more which results in a very poor response time for users logged into that virtual machine. The net effect of such scheduling layering is that individual virtualized operating systems receive only a portion of the available CPU cycles,

even though they believe they are receiving all cycles and that they are scheduling all of those cycles.Commonly, the time-of-day clocks in virtual machines are incorrect because timers take no longer to trigger than they would on dedicated CPU's.

Virtualizations can thus undo the good scheduling-algorithm efforts of the operating systems within virtual machines.

Scheduling in Real Time Systems

Real-time systems are systems that carry real-time tasks. These tasks need to be performed immediately with a certain degree of urgency. In particular, these tasks are related to control of certain events (or) reacting to them. Real-time tasks can be classified as hard real-time tasks and soft real-time tasks.

A hard real-time task must be performed at a specified time which could otherwise lead to huge losses. In soft real-time tasks, a specified deadline can be missed. This is because the task can be rescheduled (or) can be completed after the specified time,

In real-time systems, the scheduler is considered as the most important component which is typically a short-term task scheduler. The main focus of this scheduler is to reduce the response time associated with each of the associated processes instead of handling the deadline.

If a preemptive scheduler is used, the real-time task needs to wait until its corresponding tasks time slice completes. In the case of a non-preemptive scheduler, even if the highest priority is allocated to the task, it needs to wait until the completion of the current task. This task can be slow (or) of the lower priority and can lead to a longer wait.

A better approach is designed by combining both preemptive and non-preemptive scheduling. This can be done by introducing time-based interrupts in priority based systems which means the currently running process is interrupted on a time-based interval and if a higher priority process is present in a ready queue, it is executed by preempting the current process.

Based on schedulability, implementation (static or dynamic), and the result (self or dependent) of analysis, the scheduling algorithm are classified as follows.

1. **Static table-driven approaches:**
   These algorithms usually perform a static analysis associated with scheduling and capture the schedules that are advantageous. This helps in providing a schedule that can point out a task with which the execution must be started at run time.

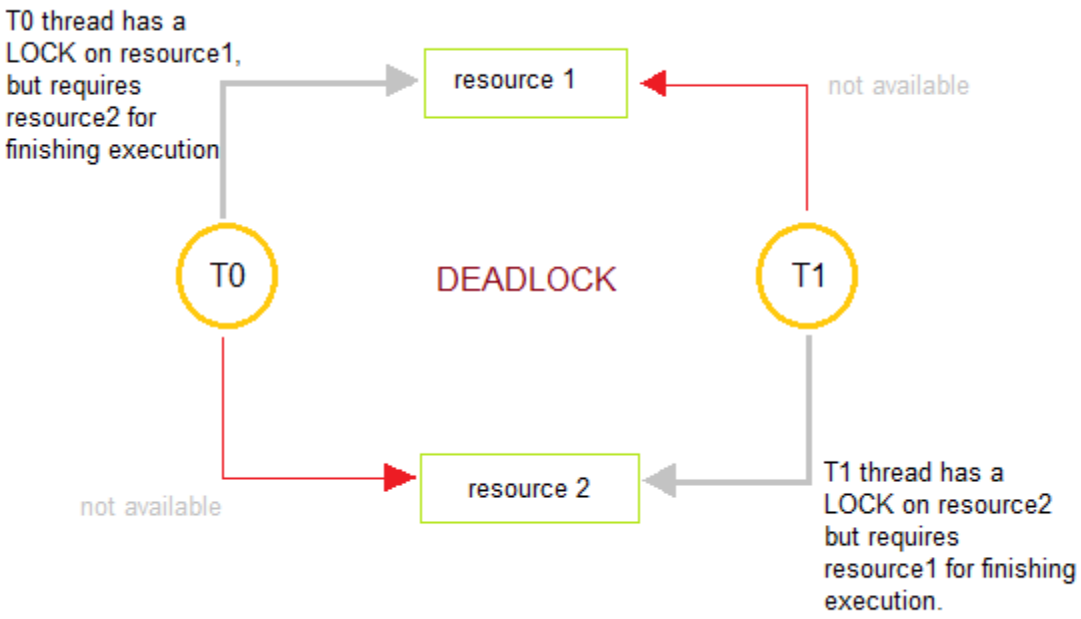
2. **Static priority-driven preemptive approaches:**
   Similar to the first approach, these type of algorithms also uses static analysis of scheduling. The difference is that instead of selecting a particular schedule, it provides a useful way of assigning priorities among various tasks in preemptive scheduling.

3. **Dynamic planning-based approaches:**
   Here, the feasible schedules are identified dynamically (at run time). It carries a certain fixed time interval and a process is executed if and only if satisfies the time constraint.

4. **Dynamic best effort approaches:**
   These types of approaches consider deadlines instead of feasible schedules. Therefore the task is aborted if its deadline is reached. This approach is used widely is most of the real-time systems.

Deadlock

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.



How to avoid Deadlocks

Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. **Mutual Exclusion**

Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

2. **Hold and Wait**

In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

3. **No Preemption**

Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

4. **Circular Wait**

Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing(or decreasing) order.

Handling Deadlock

The above points focus on preventing deadlocks. But what to do once a deadlock has occured. Following three strategies can be used to remove deadlock after its occurrence.

1. **Preemption**

We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.

2. **Rollback**

In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

3. **Kill one or more processes**

This is the simplest way, but it works.

What is a Livelock?

There is a variant of deadlock called livelock. This is a situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work. This is similar to deadlock in that no progress is made but differs in that neither process is blocked or waiting for anything.

A human example of livelock would be two people who meet face-to-face in a corridor and each moves aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time.

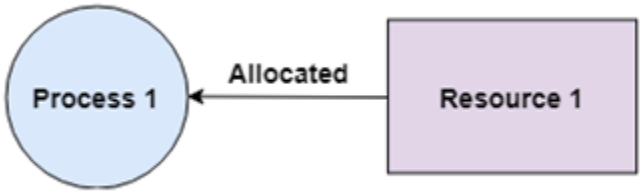
Deadlock Characterization

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive. They are given as follows −

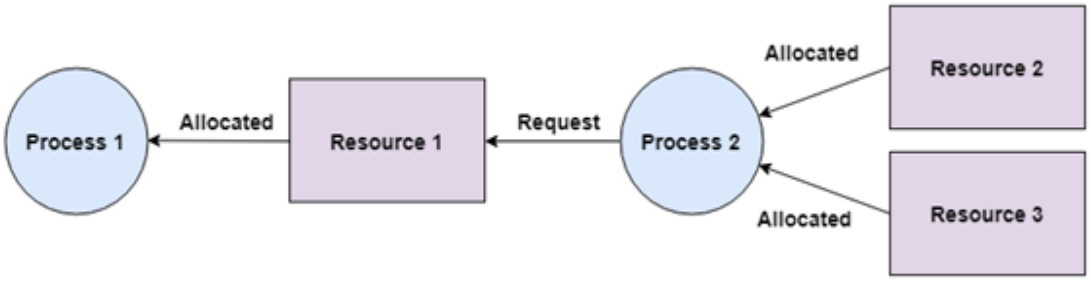
Mutual Exclusion

There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



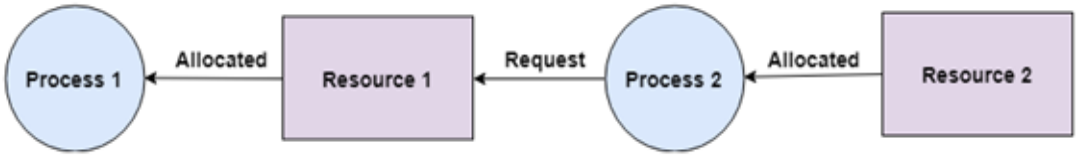
Hold and Wait

A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



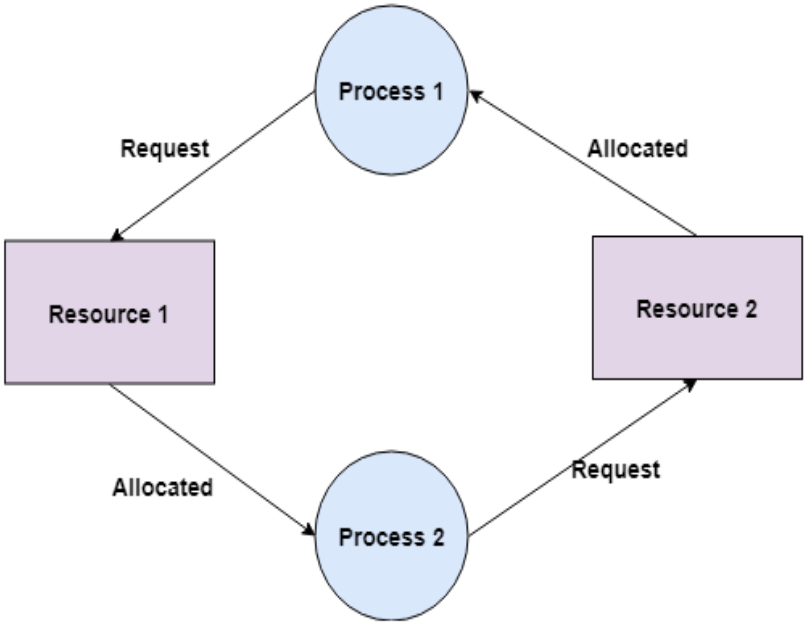
No Preemption

A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



Circular Wait

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Deadlock Prevention And Avoidance

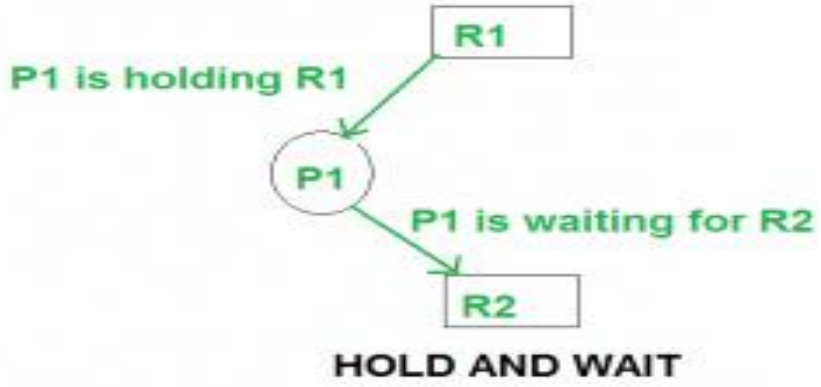
Deadlock Prevention

We can prevent Deadlock by eliminating any of the above four conditions.

Eliminate Mutual Exclusion
It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

Eliminate Hold and wait

1. Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.
2. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.



Eliminate No Preemption
Preempt resources from the process when resources required by other high priority processes.

Eliminate Circular Wait
Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.

For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

Deadlock Avoidance

Deadlock avoidance can be done with Banker's Algorithm.

Banker's Algorithm

Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

Inputs to Banker's Algorithm:

1. Max need of resources by each process.
2. Currently allocated resources by each process.
3. Max free available resources in the system.

The request will only be granted under the below condition:

1. If the request made by the process is less than equal to max need to that process.
2. If the request made by the process is less than equal to the freely available resource in the system.

Example:

Total resources in system:

A B C D

6 5 7 6

Available system resources are:

A B C D

3 1 1 2

Processes (currently allocated resources):

A B C D

P1 1 2 2 1

P2 1 0 3 3

P3 1 2 1 0

Processes (maximum resources):

A B C D

P1 3 3 2 2

P2 1 2 3 4

P3 1 3 5 0

Need = maximum resources - currently allocated resources.

Processes (need resources):

A B C D

P1 2 1 0 1

P2 0 2 0 1

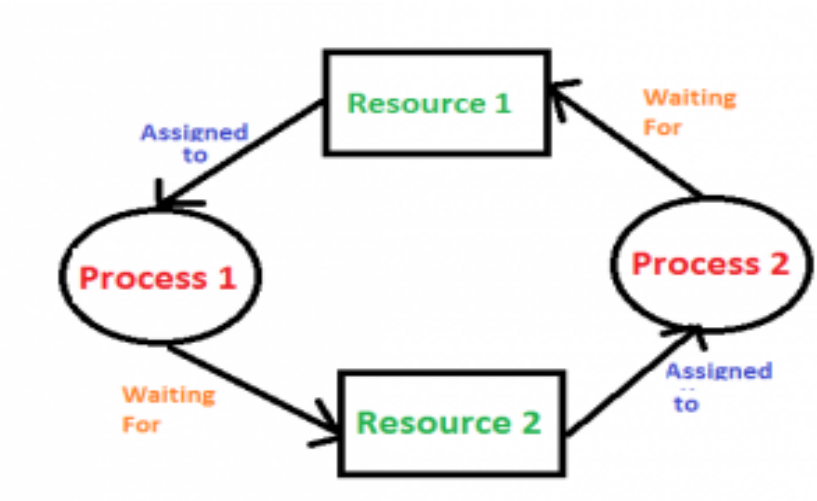
P3 0 1 4 0

Note: Deadlock prevention is more strict that Deadlock Avoidance.

Deadlock Detection And Recovery

Deadlock Detection

1. If resources have single instance:
   In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient condition for deadlock.



In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1 → P1 → R2 → P2. So, Deadlock is Confirmed.

2. If there are multiple instances of resources:
   Detection of the cycle is necessary but not sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

Deadlock Recovery

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is time and space consuming process. Real-time operating systems use Deadlock recovery.

Recovery method

1. **Killing the process:** killing all the process involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till system recover from deadlock.
2. **Resource Preemption:** Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

Memory Management

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Process Address Space

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated −

S.N.	Memory Addresses & Description
1	**Symbolic addresses** The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	**Relative addresses** At the time of compilation, a compiler converts symbolic addresses into relative addresses.

3	**Physical addresses** The loader generates these addresses at the time when a program is loaded into main memory.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

Static vs Dynamic Loading

The choice between Static or Dynamic Loading is to be made at the time of computer program being developed. If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency. The linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.

If you are writing a Dynamically loaded program, then your compiler will compile the program and for all the modules which you want to include dynamically, only references will be provided and rest of the work will be done at the time of execution.

At the time of loading, with **static loading**, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using **dynamic loading**, dynamic routines of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.

Static vs Dynamic Linking

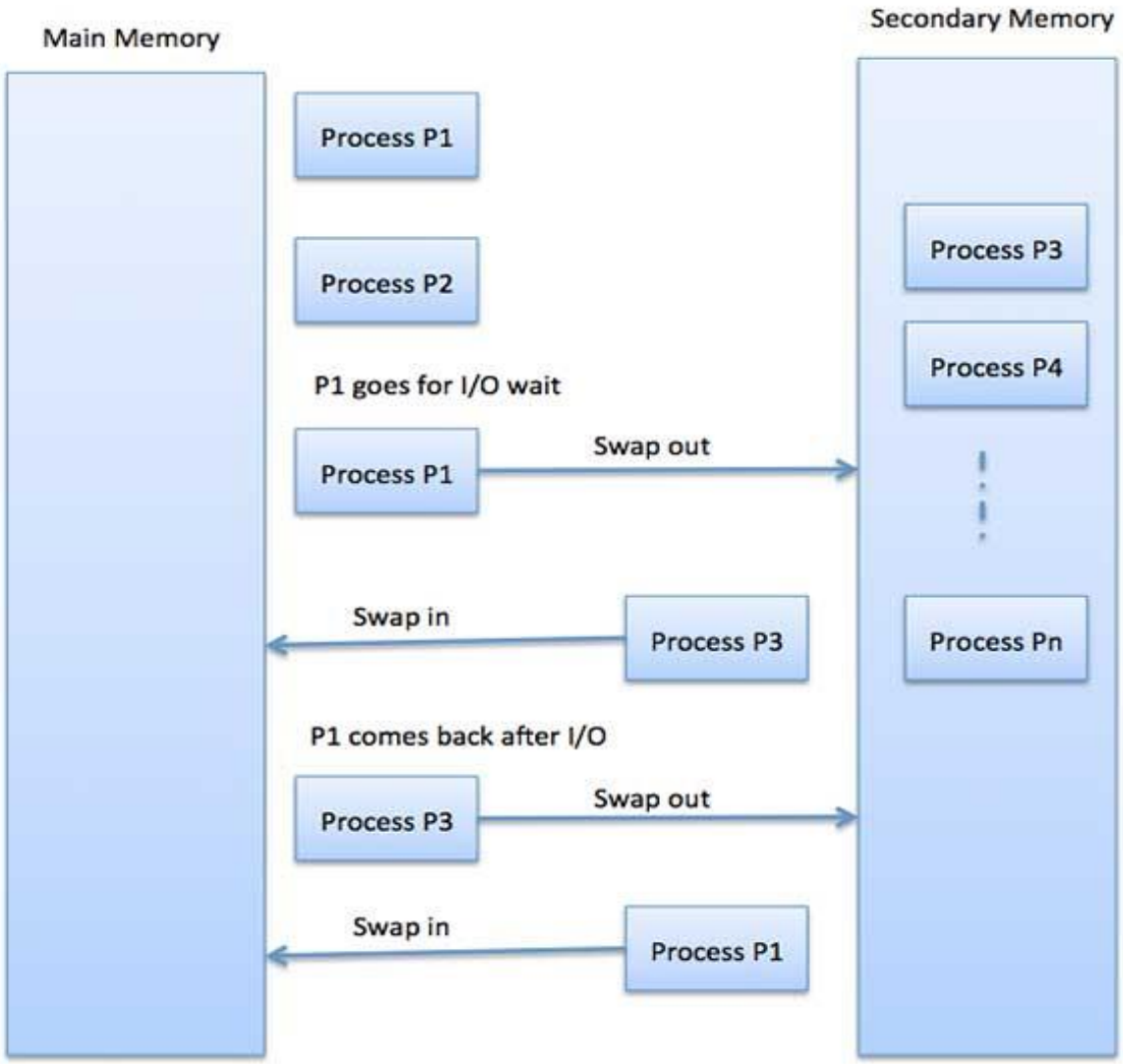
As explained above, when static linking is used, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.

When dynamic linking is used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module is provided at the time of compilation and linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in Unix are good examples of dynamic libraries.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

2048KB / 1024KB per second

= 2 seconds

= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

Memory Allocation

Main memory usually has two partitions −

- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation** In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

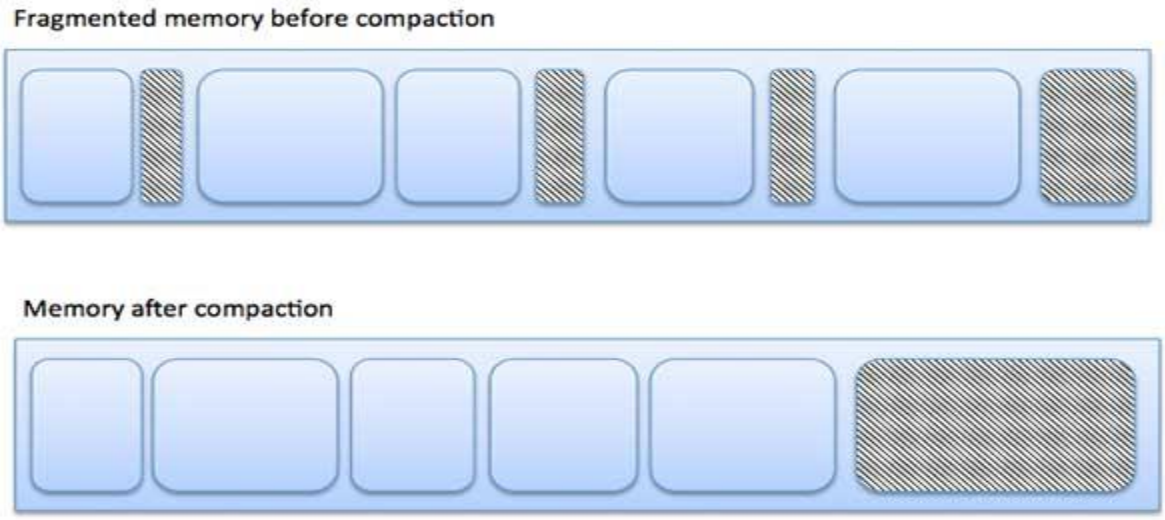
Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −



External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Paging

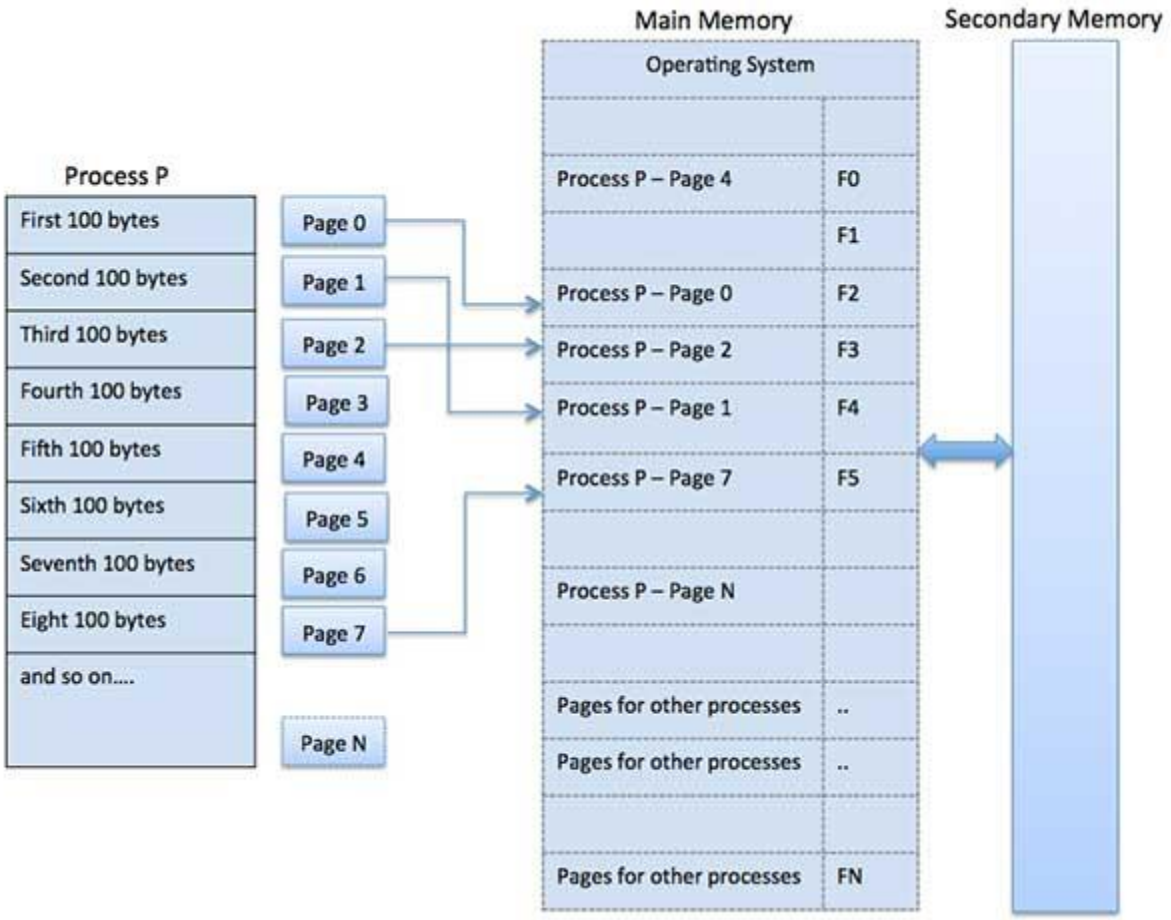
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

A non-contiguous policy with a fixed size partition is called paging. A computer can address more memory than the amount of physically installed on the system. This extra memory is actually called virtual memory. Paging technique is very important in implementing virtual memory. Secondary memory is divided into equal size partition (fixed) called pages. Every process will have a separate page table. The entries in the page table are the number of pages a process. At each entry either we have an invalid pointer which means the page is not in main memory or we will get the corresponding frame number. When the frame number is combined with instruction of set D than we will get the corresponding physical address. Size of a page table is generally very large so cannot be accommodated inside the PCB, therefore, PCB contains a register value PTBR(page table base register) which leads to the page table.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page

to have optimum utilization of the main memory and to avoid external fragmentation.



Address Translation

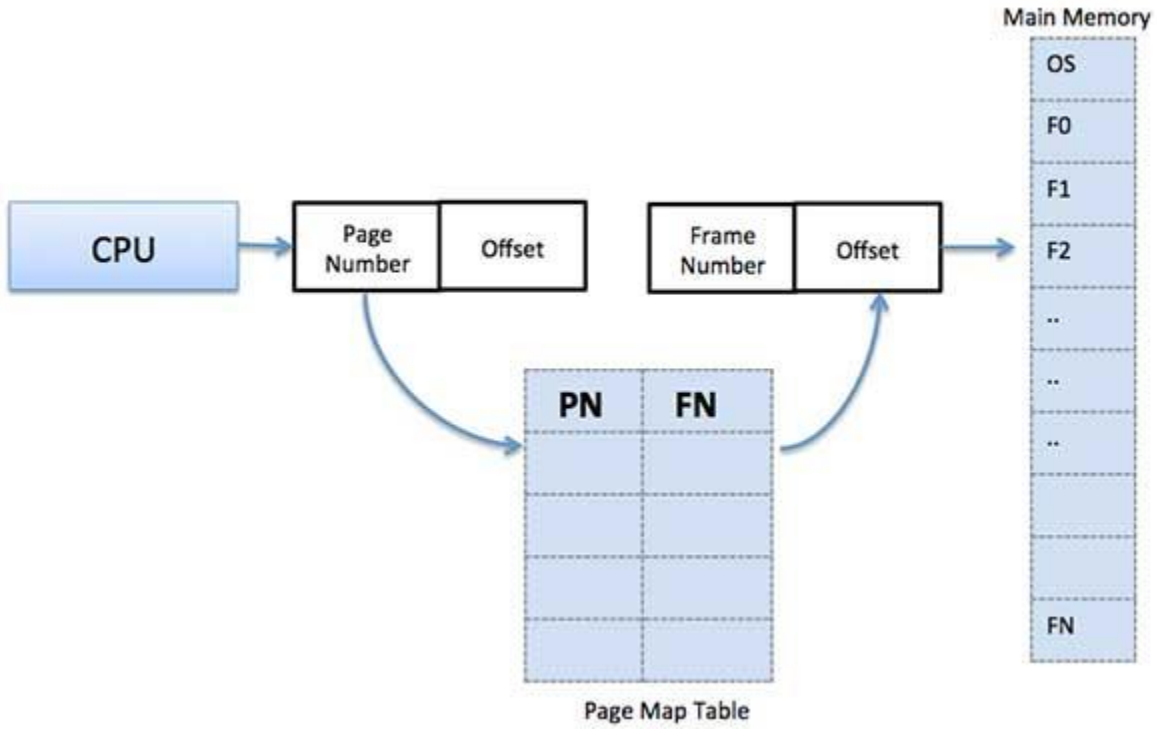
Page address is called **logical address** and represented by **page number** and the **offset**.

Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

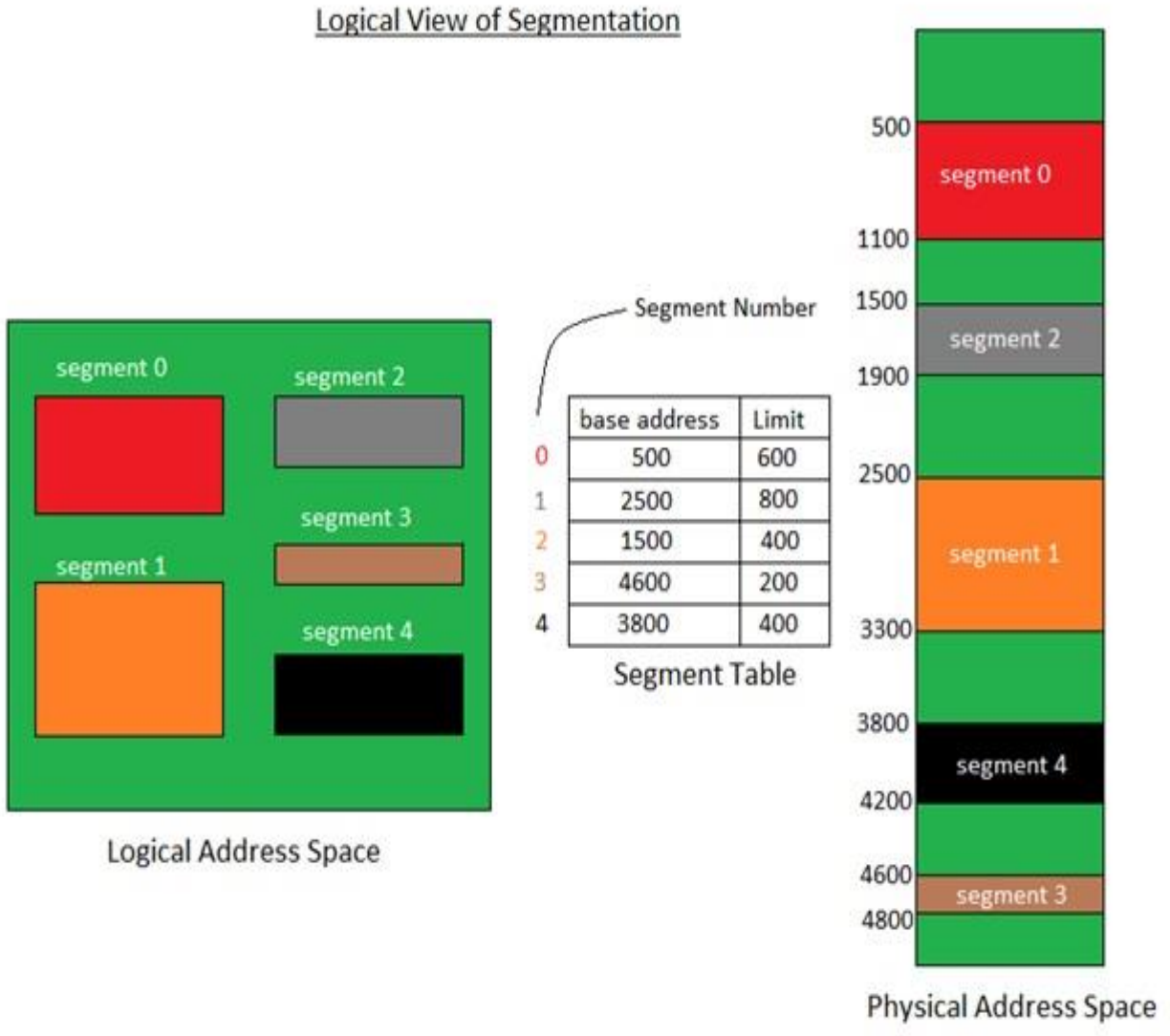
A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

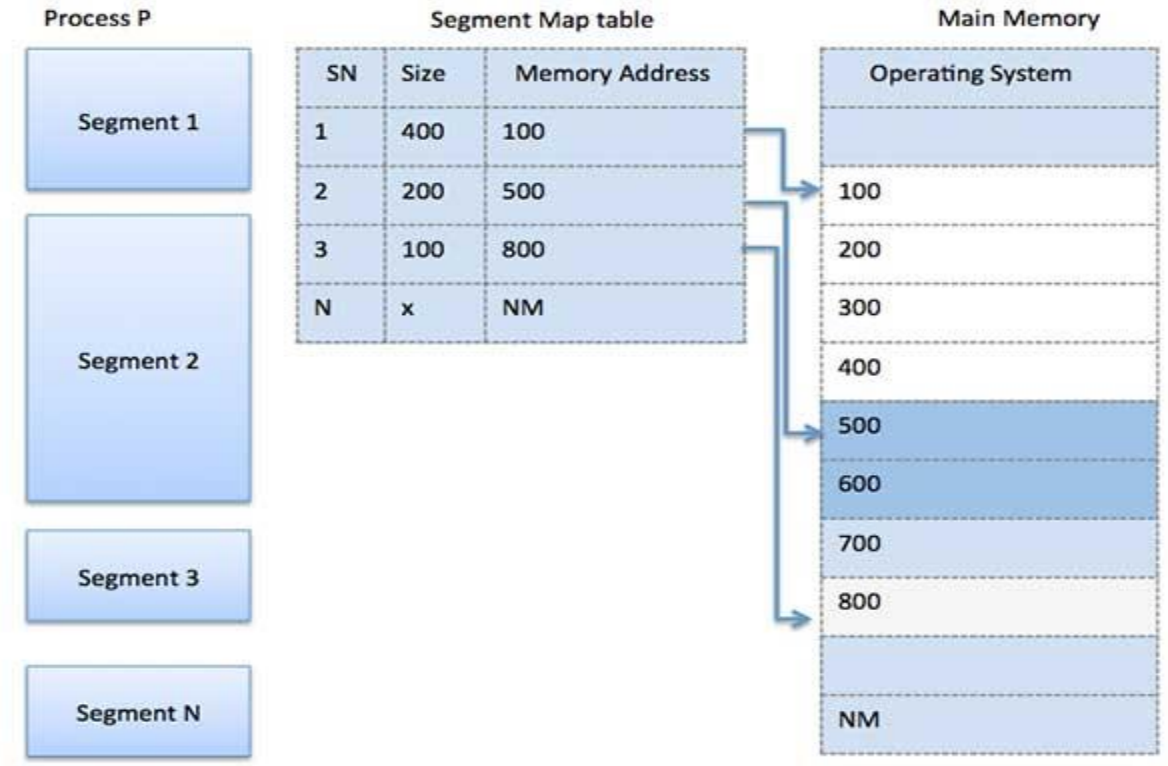
Segmentation is a programmer view of the memory where instead of dividing a process into equal size partition we divided according to program into partition called segments. The translation is the same as paging but paging segmentation is independent of internal fragmentation but suffers from external fragmentation.

Reason of external fragmentation is program can be divided into segments but segment must be contiguous in nature.

Segmentation with paging

In segmentation with paging, we take advantages of both segmentation as well as paging. It is a kind of multilevel paging but in multilevel paging, we divide a page table into equal size partition but here in segmentation with paging, we divide it according to segments. All the properties are the same as that of paging because segments are divided into pages.





Contiguous memory allocation

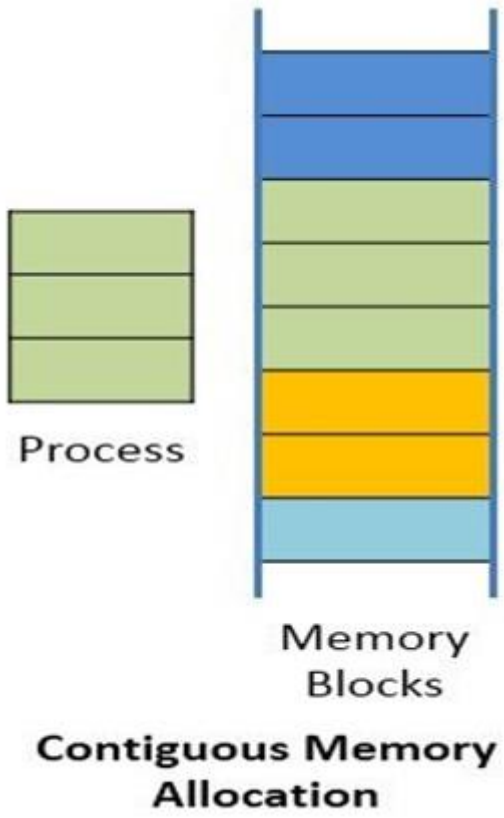
In **contiguous memory allocation**, all the available memory space remain together in one place. It means freely available memory partitions are not scattered here and there across the whole memory space.

In the **contiguous memory allocation**, both the operating system and the user must reside in the main memory. The main memory is divided into two portions one portion is for the operating and other is for the user program.

In the **contiguous memory allocation** when any user process request for the memory a single section of the contiguous memory block is given to that process according to its need. We can achieve contiguous memory allocation by dividing memory into the fixed-sized partition.

A single process is allocated in that fixed sized single partition. But this will increase the degree of multiprogramming means more than one process in the

main memory that bounds the number of fixed partition done in memory. Internal fragmentation increases because of the contiguous memory allocation.



Contiguous Memory Allocation

→ Fixed sized partition

In the fixed sized partition the system divides memory into fixed size partition (may or may not be of the same size) here entire partition is allowed to a process and if there is some wastage inside the partition is allocated to a process and if there is some wastage inside the partition then it is called internal fragmentation.

Advantage: Management or book keeping is easy.

Disadvantage: Internal fragmentation

→ Variable size partition

In the variable size partition, the memory is treated as one unit and space allocated to a process is exactly the same as required and the leftover space can be reused again.

Advantage: There is no internal fragmentation.

Disadvantage: Management is very difficult as memory is becoming purely fragmented after some time.

Demand Paging

According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required. In other words, it says that do not load any page in the main memory until it is required.

Whenever any page is referred for the first time in the main memory, then that page will be found in the secondary memory.

After that, it may or may not be present in the main memory depending upon the page replacement algorithm which will be covered later in this tutorial.

What is a Page Fault?

If the referred page is not present in the main memory then there will be a miss and the concept is called Page miss or page fault.

The CPU has to access the missed page from the secondary memory. If the number of page fault is very high then the effective access time of the system will become very high.

What is Thrashing?

If the number of page faults is equal to the number of referred pages or the number of page faults are so high so that the CPU remains busy in just reading the pages from the secondary memory then the effective access time will be the time taken by the CPU to read one word from the secondary memory and it will be so high. The concept is called thrashing.

If the page fault rate is PF %, the time taken in getting a page from the secondary memory and again restarting is S (service time) and the memory access time is ma then the effective access time can be given as;

1. EAT = PF X S + (1 - PF) X (ma)

Page Replacement Algorithms

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms :

- **First In First Out (FIFO) –**
  This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

- **Example-1**Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames.Find number of page faults.

  Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
  when 3 comes, it is already in memory so —> **0 Page Faults.**
  Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
  6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**
  Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

Belady's anomaly – Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

- **Optimal Page replacement –**
  In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will takes place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

- **Least Recently Used –**
  In this algorithm page will be replaced which is least recently used.

Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.
4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Storage Management

The term storage management encompasses the technologies and processes organizations use to maximize or improve the performance of their data storage resources. It is a broad category that includes virtualization, replication, mirroring, security, compression, traffic analysis, process automation,
storage provisioning and related techniques.

By some estimates, the amount of digital information stored in the world's computer systems is doubling every year. As a result, organizations feel constant pressure to expand their storage capacity. However, doubling a company's

storage capacity every year is an expensive proposition. In order to reduce some of those costs and improve the capabilities and security of their storage solutions, organizations turn to a variety of storage management solutions.

Storage Management Benefits

Many storage management technologies, like storage virtualization, deduplication and compression, allow companies to better utilize their existing storage. The benefits of these approaches include lower costs -- both the one-time capital expenses associated with storage devices and the ongoing operational costs for maintaining those devices.

Most storage management techniques also simplify the management of storage networks and devices. That can allow companies to save time and even reduce the number of IT workers needed to maintain their storage systems, which in turn, also reduces overall storage operating costs.

Storage management can also help improve a data center's performance. For example, compression and technology can enable faster I/Os, and automatic storage provisioning can speed the process of assigning storage resources to various applications.

In addition, virtualization and automation technologies can help an organization improve its agility. These storage management techniques make it possible to reassign storage capacity quickly as business needs change, reducing wasted space and improving a company's ability to respond to evolving market conditions.

Finally, many storage management technologies, such as replication, mirroring and security, can help a data center improve its reliability and availability. These techniques are often particularly important for backup and archive storage, although they also apply to primary storage. IT departments often turn to these technologies for help in meeting SLAs or achieving compliance goals.

Storage Management: Related Terms

Storage management is very closely related to Storage Resource Management (SRM). SRM often refers particularly to software used to manage storage networks and devices. By contrast, the term "storage management" can refer to devices and processes, as well as actual software. In addition, SRM usually refers specifically to software for allocating storage capacity based on company policies and ongoing events. It may include asset management, charge back, capacity management, configuration management, data and media migration, event management, performance and availability management, policy management, quota management, and media management capabilities. In short, SRM is a subset of storage management; however, the two terms are sometimes used interchangeably.

Storage management is also closely associated with networked storage solutions, such as storage area networks (SANs) and network-attached storage (NAS) devices. Because using SAN and NAS devices is more complicated than using direct-attached storage (DAS), many organizations deploy SRM software when they deploy their storage networking environments. However, storage management techniques like replication, mirroring, security, compression and others can be utilized with DAS devices as well as with SANs and NAS arrays.

Storage management is often used in virtualized or cloud computing environments.

Storage Management Implementation

Because storage management is such a broad category, it's difficult to provide detailed instructions on how to install or how to use storage management technologies. In general, storage management technology can be deployed as software or it can be included in a hardware device. Storage management techniques can be applied to primary, backup or archived storage. Deployment and implementation procedures will vary widely depending on the type of storage management selected and the vendor. In addition, the skills and training of storage administrators and other personnel add another level to an organization's storage management capabilities.

Storage Management Technology

The primary organization involved in establishing storage management standards is the Storage Networking Industry Association (SNIA). It has put forth several important storage specifications, including the Storage Management Initiative Specification (SMI-S) and the Cloud Data Management Interface (CDMI). SMI-S defines the attributes of storage hardware, such as Fibre Channel switches, Fibre Channel and iSCI arrays, NAS devices, tape libraries and host profiles. It also addresses storage management software issues, such as configuration discovery, provisioning and trending, security, asset management, compliance and cost management, event management and data protection. The CDMI specification provides standards for cloud storage services, enabling interoperability among various storage management solutions.

Mass storage

Mass storage refers to various techniques and devices for storing large amounts of data. The earliest storage devices were punched paper cards, which were used as early as 1804 to control silk-weaving looms. Modern mass storage devices include all types of disk drives and tape drives.

Mass storage is distinct from memory, which refers to temporary storage areas within the computer. Unlike main memory, mass storage devices retain data even when the computer is turned off.

Examples of Mass Storage Devices (MSD)

Common types of mass storage include the following:

- solid-state drives (SSD)
- hard drives
- external hard drives
- optical drives
- tape drives

- RAID storage
- USB storage
- flash memory cards

Today, mass storage is measured in gigabytes (1,024 megabytes) and terabytes(1,024 gigabytes). Older mass storage technology, such as floppy drives, are measured in kilobytes (1,024 bytes), megabytes (1,024 kilobytes),

Mass storage is sometimes called auxiliary storage.

RAID

RAID is short for **r**edundant **a**rray of **i**ndependent **d**isks.

Originally, the term RAID was defined as **r**edundant **a**rray of **i**nexpensive **d**isks, but now it usually refers to a **r**edundant **a**rray of **i**ndependent **d**isks. RAID storage uses multiple disks in order to provide fault tolerance, to improve overall performance, and to increase storage capacity in a system. This is in contrast with older storage devices that used only a single disk drive to store data.

RAID allows you to store the same data redundantly (in multiple paces) in a balanced way to improve overall performance. RAID disk drives are used frequently on servers but aren't generally necessary for personal computers.

How RAID Works

With RAID technology, data can be mirrored on one or more disks in the same array, so that if one disk fails, the data is preserved. Thanks to a technique known as striping (a technique for spreading data over multiple disk drives), RAID also offers the option of reading or writing to more than one disk at the same time in order to improve performance.

In this arrangement, sequential data is broken into segments which are sent to the various disks in the array, speeding up throughput. A typical RAID array uses multiple disks that appear to be a single device so it can provide more storage capacity than a single disk.

Standard RAID Levels

RAID devices use many different architectures, called levels, depending on the desired balance between performance and fault tolerance. RAID levels describe how data is distributed across the drives. Standard RAID levels include the following:

Level 0: Striped disk array without fault tolerance

Provides data striping (spreading out blocks of each file across multiple disk drives) but no redundancy. This improves performance but does not deliver fault tolerance. If one drive fails then all data in the array is lost.

Level 1: Mirroring and duplexing

Provides disk mirroring. Level 1 provides twice the read transaction rate of single disks and the same write transaction rate as single disks.

Level 2: Error-correcting coding

Not a typical implementation and rarely used, Level 2 stripes data at the bit level rather than the block level.

Level 3: Bit-interleaved parity

Provides byte-level striping with a dedicated parity disk. Level 3, which cannot service simultaneous multiple requests, also is rarely used.

Level 4: Dedicated parity drive

A commonly used implementation of RAID, Level 4 provides block-level striping (like Level 0) with a parity disk. If a data disk fails, the parity data is used to create a replacement disk. A disadvantage to Level 4 is that the parity disk can create write bottlenecks.

Level 5: Block interleaved distributed parity

Provides data striping at the byte level and also stripe error correction information. This results in excellent performance and good fault tolerance. Level 5 is one of the most popular implementations of RAID.

Level 6: Independent data disks with double parity

Provides block-level striping with parity data distributed across all disks.

Level 10: A stripe of mirrors

Not one of the original RAID levels, multiple RAID 1 mirrors are created, and a RAID 0 stripe is created over these.

Non-Standard RAID Levels

Some devices use more than one level in a hybrid or nested arrangement, and some vendors also offer non-standard proprietary RAID levels. Examples of non-standard RAID levels include the following:

Level 0+1: A Mirror of Stripes

Not one of the original RAID levels, two RAID 0 stripes are created, and a RAID 1 mirror is created over them. Used for both replicating and sharing data among disks.

Level 7

A trademark of Storage Computer Corporation that adds caching to Levels 3 or 4.

RAID 1E

A RAID 1 implementation with more than two disks. Data striping is combined with mirroring each written stripe to one of the remaining disks in the array.

RAID S

Also called Parity RAID, this is EMC Corporation's proprietary striped parity RAID system used in its Symmetrix storage systems.

RAID History and Alternative Storage Options

Before RAID devices became popular, most systems used a single drive to store data. This arrangement is sometimes referred to as a SLED (single large expensive disk). However, SLEDs have some drawbacks. First, they can create I/O bottlenecks because the data cannot be read from the disk quickly enough to keep up with the other components in a system, particularly the processor. Second, if a SLED fails, all the data is lost unless it has been recently backed up onto another disk or tape.

In 1987, three University of California, Berkeley, researchers -- David Patterson, Garth A. Gibson, and Randy Katz -- first defined the term RAID in a paper titled A Case for Redundant Arrays of Inexpensive Disks (RAID). They theorized that spreading data across multiple drives could improve system performance, lower costs and reduce power consumption while avoiding the potential reliability problems inherent in using inexpensive, and less reliable, disks. The paper also described the five original RAID levels.

Today, RAID technology is nearly ubiquitous among enterprise storage devices and is also found in many high-capacity consumer storage devices. However, some non-RAID storage options do exist. One alternative is JBOD (Just a Bunch of Drives). JBOD architecture utilizes multiple disks, but each disk in the device is addressed separately. JBOD provides increased storage capacity versus a single disk, but doesn't offer the same fault tolerance and performance benefits as RAID devices.

Another RAID alternative is concatenation or spanning. This is the practice of combining multiple disk drives so that they appear to be a single drive. Spanning increases the storage capacity of a drive; however, as with JBOD, spanning does not provide reliability or speed benefits.

RAID Is Not Data Backup

RAID should not be confused with data backup. Although some RAID levels do provide redundancy, experts advise utilizing a separate storage system for backup and disaster recovery purposes.

Setting Up a RAID Array

In order to set up a RAID array, you'll need a group of disk drives and either a software or a hardware controller. Software RAID runs directly on a server, utilizing server resources. As a result, it may cause some applications to run more slowly. Most server operating systems include some built-in RAID management capabilities.

You can also set up your own RAID array by adding a RAID controller to a server or a desktop PC. The RAID controller runs essentially the same software, but it uses its own processor instead of the system's CPU. Some less expensive "fake RAID" controllers provide RAID management software but don't have a separate processor.

Alternatively, you can purchase a pre-built RAID array from a storage vendor. These appliances generally include two RAID controllers and a group of disks in their own housing.

Using a RAID array is usually no different than using any other kind of primary storage. The RAID management will be handled by the hardware or software controller and is generally invisible to the end user.

RAID Technology Standards

The Storage Networking Industry Association has established the Common RAID Disk Data Format (DDF) specification. In an effort to promote interoperability among different RAID vendors, it defines how data should be distributed across the disks in a RAID device.

Another industry group called the RAID Advisory Board worked during the 1990s to promote RAID technology, but the group is no longer active.

Disk Structure

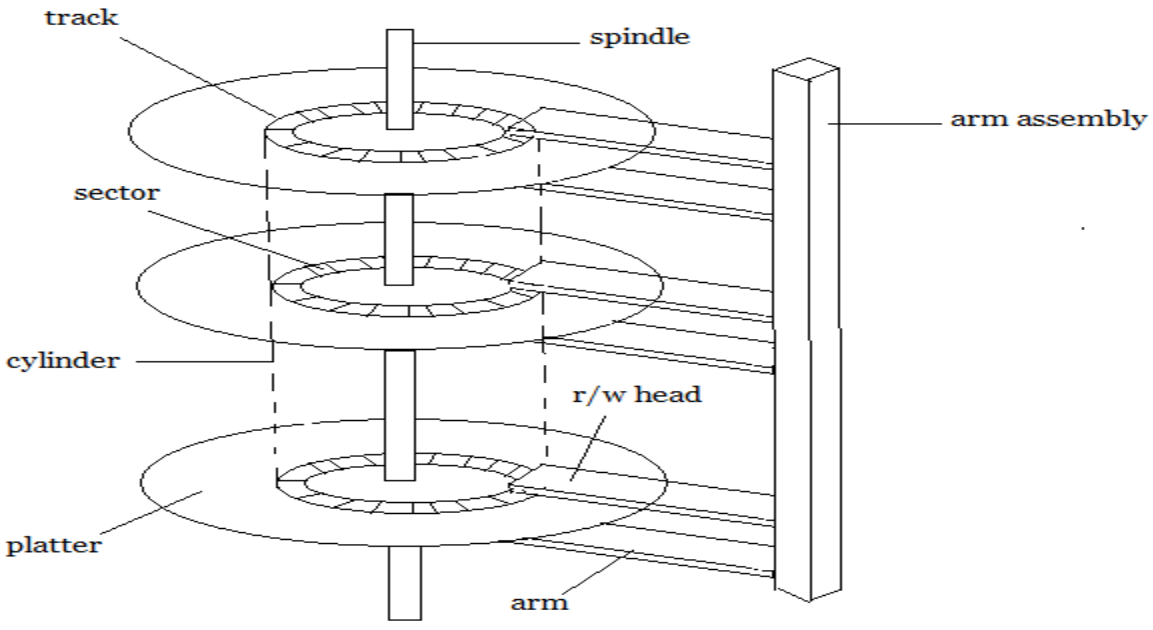
Secondary Storage

Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

- Secondary storage is also called auxiliary storage.
- Secondary storage is less expensive when compared to primary memory like RAMs.
- The speed of the secondary storage is also lesser than that of primary storage.
- Hence, the data which is less frequently accessed is kept in the secondary storage.
- A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.

Magnetic Disk Structure

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.



Structure of a magnetic disk

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

Operating System Input Output I/O

The three main jobs of a computer are Input, Output, and Processing. In most of the cases, the most important job is Input / Output, and the processing is simply incidental. For an example, when we browse a web page or edit any file, our immediate attention is to read or enter some information, not for computing an answer. The fundamental role of the operating system in computer Input / Output is to manage and organize I/O operations and all I/O devices.

The various devices that are connected to the computer need to be controlled and it is a key concern of operating-system designers. This is as I/O devices vary so

widely in their functionality and speed (for example a mouse, a hard disk and a CD-ROM), varied methods are required for controlling them. These methods form the I/O sub-system of the kernel of OS that separates the rest of the kernel from the complications of managing I/O devices.

File Access Methods

When a file is used, information is read and accessed into computer memory and there are several ways to access this information of the file. Some systems provide only one access method for files. Other systems, such as those of IBM, support many access methods, and choosing the right one for a particular application is a major design problem.

There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.

1. **Sequential Access –**
   It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.

Read and write make up the bulk of the operation on a file. A read operation -read next- read the next position of the file and automatically advance a file pointer, which keeps track I/O location. Similarly, for the writewrite next append to the end of the file and advance to the newly written material.

Key points:

1.

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one
- When we use write command, it will allocate memory and move the pointer to the end of the file

- Such a method is reasonable for tape.

2. **Direct Access –**
   Another method is direct access method also known as relative access method. A filed-length logical record that allows the program to read and write record rapidly. in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59 and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.

A block number provided by the user to the operating system is normally a relative block number, the first relative block of the file is 0 and then 1 and so on.

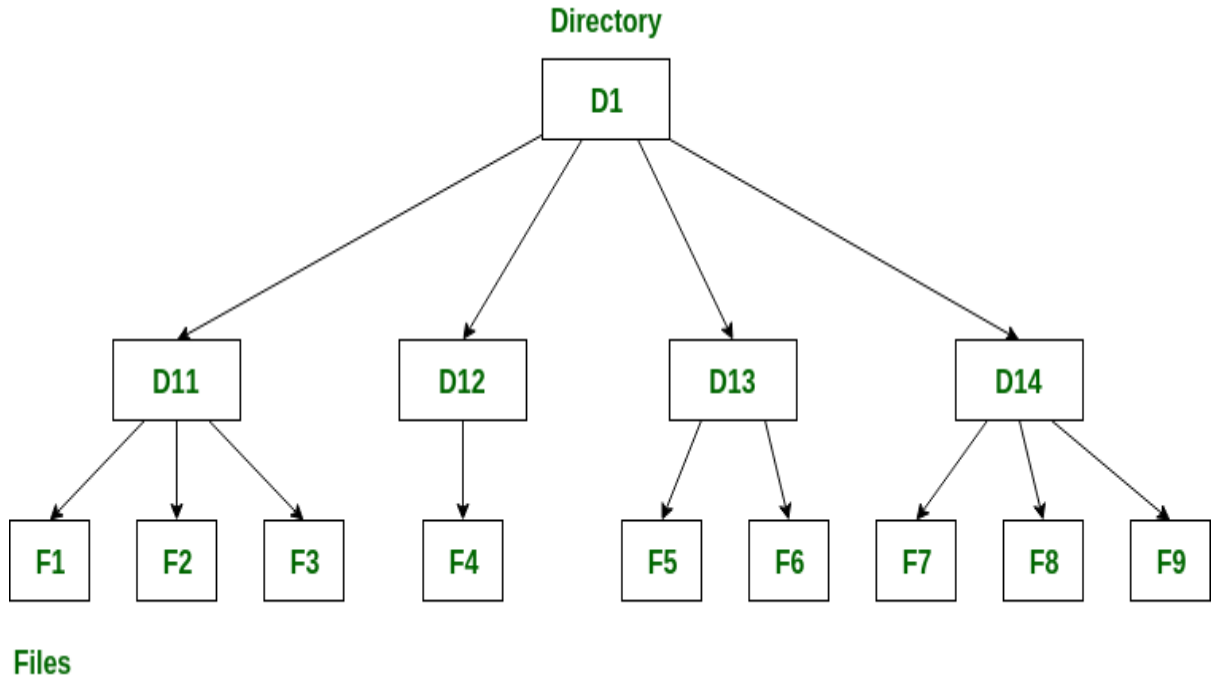
3. **Index sequential method –**
   It is the other method of accessing a file which is built on the top of the direct access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index and then by the help of pointer we access the file directly.

Key points:

- It is built on top of Sequential access.
- It control the pointer by using index.

Structures of Directory

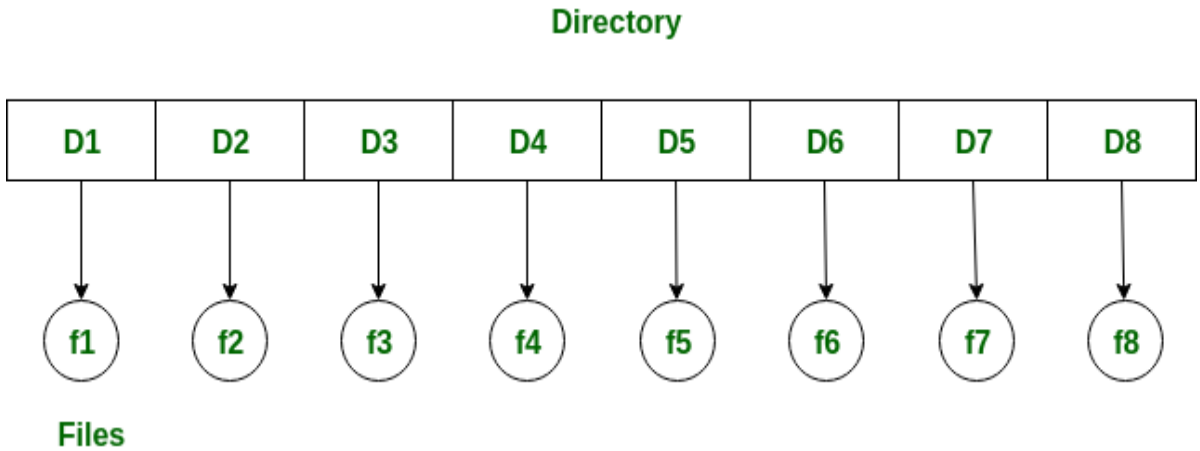
A **directory** is a container that is used to contain folders and file. It organizes files and folders into a hierarchical manner.



There are several logical structures of a directory, these are given below.

1. **Single-level directory –**
   Single level directory is simplest directory structure.In it all files are contained in same directory which make it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have the unique name . if two users call their dataset test, then the unique name rule violated.



Advantages:

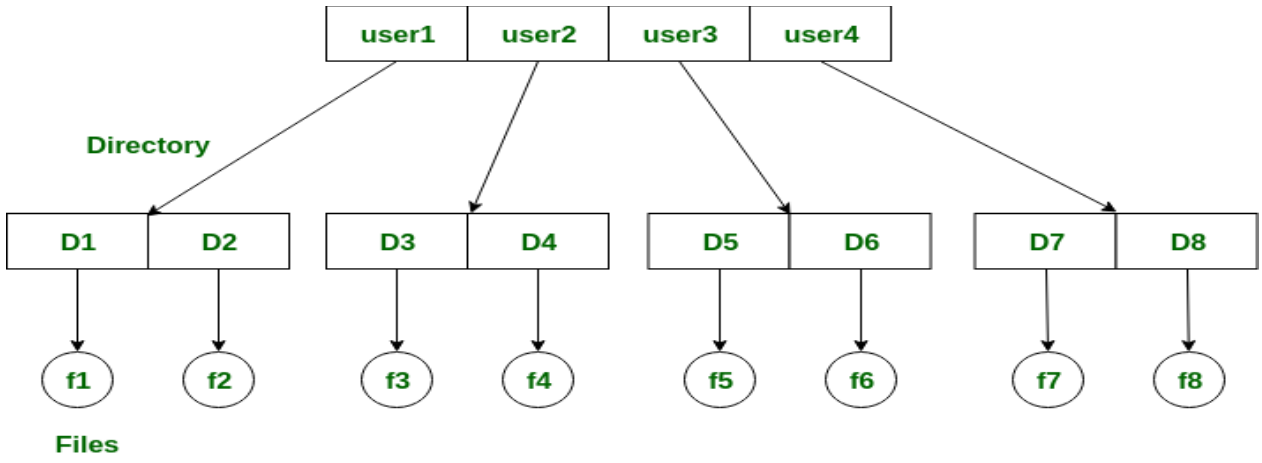
- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

- There may chance of name collision because two files can not have the same name.
- Searching will become time taking if the directory is large.
- In this can not group the same type of files together.

2. **Two-level directory –**
   As we have seen, a single level directory often leads to confusion of files names among different users. the solution to this problem is to create a separate directory for each user.

In the two-level directory structure, each user has there own user files directory (UFD). The UFDs has similar structures, but each lists only the files of a single user. system's master file directory (MFD) is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.



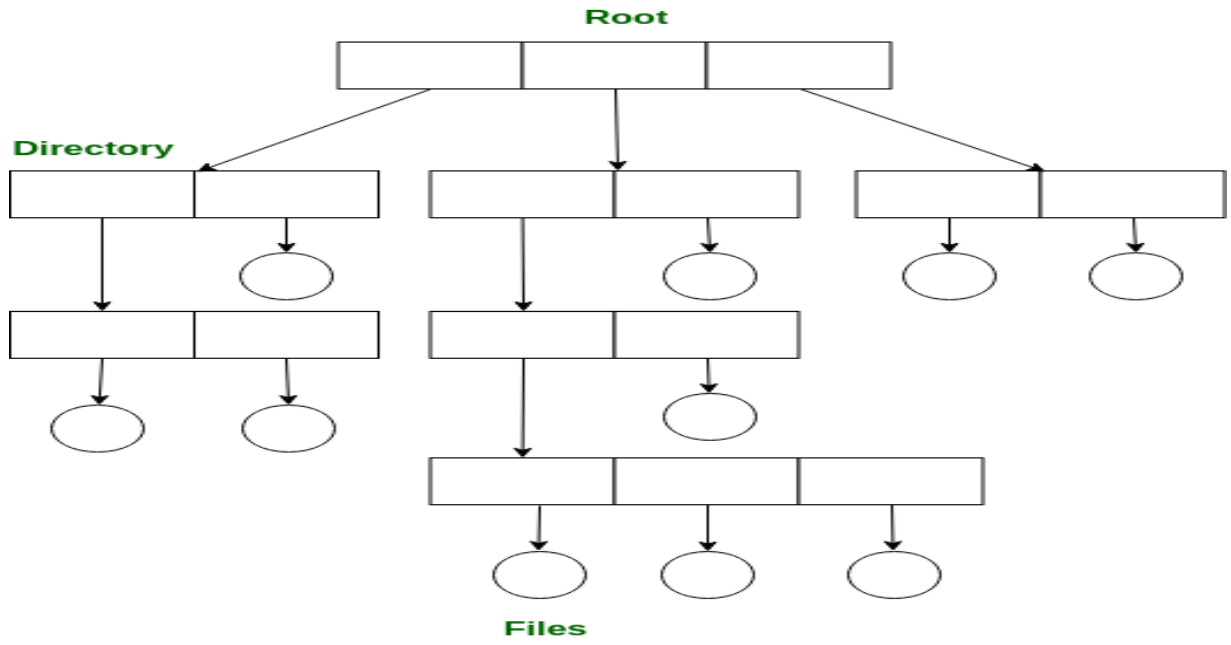
Advantages:

- We can give full path like /User-name/directory-name/.
- Different users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

Disadvantages:

1.
   - A user is not allowed to share files with other users.
   - Still it not very scalable, two files of the same type cannot be grouped together in the same user.

2. **Tree-structured directory –**
   Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.
   This generalization allows the user to create there own subdirectories and to organize on their files accordingly.



A tree structure is the most common directory structure. The tree has a root directory, and every file in the system have a unique path.

Advantages:

- Very generalize, since full path name can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute path as well as relative.

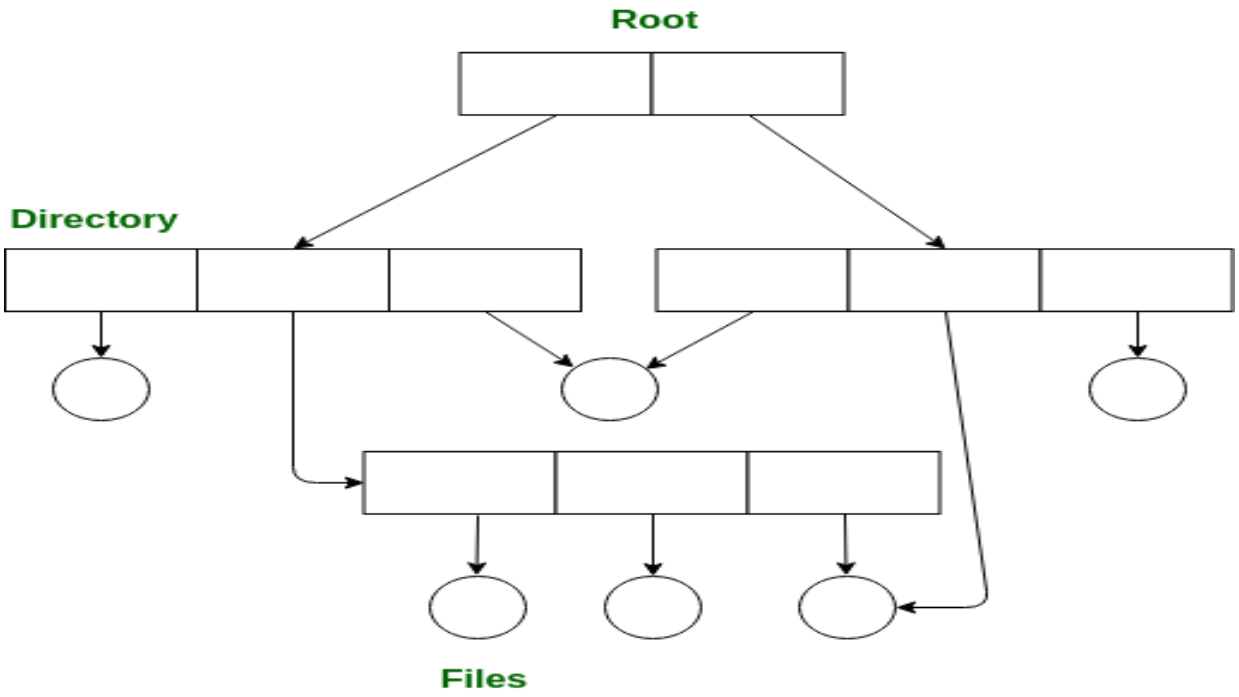
Disadvantages:

- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can not share files.
- It is inefficient, because accessing a file may go under multiple directories.

3. **Acyclic graph directory –**
   An acyclic graph is a graph with no cycle and allows to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory.

It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separating them from other projects and files of other programmers, since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use Acyclic directories.

It is the point to note that shared file is not the same as copy file . If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



Advantages:

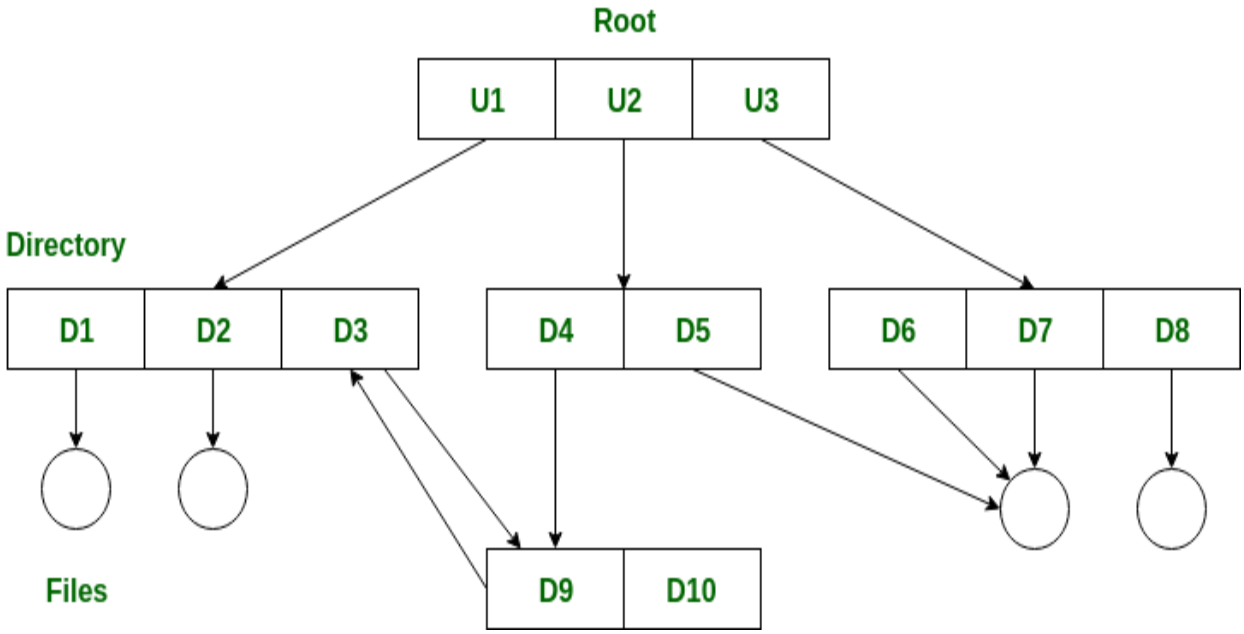
- We can share files.
- Searching is easy due to different-different paths.

Disadvantages:

- We share the files via linking, in case of deleting it may create the problem,
- If the link is softlink then after deleting the file we left with a dangling pointer.
- In case of hardlink, to delete a file we have to delete all the reference associated with it.

4. **General graph directory structure –**
   In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.

The main problem with this kind of directory structure is to calculate total size or space that has been taken by the files and directories.



Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

Disadvantages:

- It is more costly than others.
- It needs garbage collection.

Disk Data Structures

There are various on disk data structures that are used to implement a file system. This structure may vary depending upon the operating system.

1. **Boot Control Block**

Boot Control Block contains all the information which is needed to boot an operating system from that volume. It is called boot block in UNIX file system. In NTFS, it is called the partition boot sector.

2. **Volume Control Block**

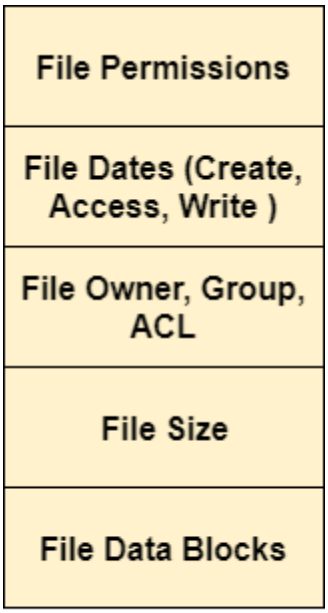
Volume control block all the information regarding that volume such as number of blocks, size of each block, partition table, pointers to free blocks and free FCB blocks. In UNIX file system, it is known as super block. In NTFS, this information is stored inside master file table.

3. **Directory Structure (per file system)**

A directory structure (per file system) contains file names and pointers to corresponding FCBs. In UNIX, it includes inode numbers associated to file names.

4. **File Control Block**

File Control block contains all the details about the file such as ownership details, permission details, file size,etc. In UFS, this detail is stored in inode. In NTFS, this information is stored inside master file table as a relational database structure. A typical file control block is shown in the image below.



File Control Block

File system Mounting

Each filesystem has its own root directory . The filesystem whose root directory is the root of the system's directory tree is called root filesystem . Other filesystems can be mounted on the system's directory tree; the directories on which they are inserted are called mount points . A mounted filesystem is the child of the mounted filesystem to which the mount point directory belongs. For instance, the /proc virtual filesystem is a child of the root filesystem (and the root filesystem is the parent of /proc).

In most traditional Unix-like kernels, each filesystem can be mounted only once. Suppose that an Ext2 filesystem stored in the /dev/fd0 floppy disk is mounted on /flp by issuing the command:

```
mount -t ext2 /dev/fd0 /flp
```

Until the filesystem is unmounted by issuing a umount command, any other mount command acting on /dev/fd0 fails.

However, Linux 2.4 is different: it is possible to mount the same filesystem several times. For instance, issuing the following command right after the previous one will likely succeed in Linux:

```
mount -t ext2 -o ro /dev/fd0 /flp-ro
```

As a result, the Ext2 filesystem stored in the floppy disk is mounted both on /flp and on /flp-ro; therefore, its files can be accessed through both /flp and /flp-ro (in this example, accesses through /flp-ro are read-only).

Of course, if a filesystem is mounted n times, its root directory can be accessed through n mount points, one per mount operation. Although the same filesystem can be accessed ...

File Sharing

File sharing is the practice of sharing or offering access to digital information or resources, including documents, multimedia (audio/video), graphics, computer programs, images and e-books. It is the private or public distribution of data or resources in a network with different levels of sharing privileges.

File sharing can be done using several methods. The most common techniques for file storage, distribution and transmission include the following:

- Removable storage devices
- Centralized file hosting server installations on networks
- World Wide Web-oriented hyperlinked documents
- Distributed peer-to-peer networks

Explained File Sharing

File sharing is a multipurpose computer service feature that evolved from removable media via network protocols, such as File Transfer Protocol (FTP). Beginning in the 1990s, many remote file-sharing mechanisms were introduced, including FTP, hotline and Internet relay chat (IRC).

Operating systems also provide file-sharing methods, such as network file sharing (NFS). Most file-sharing tasks use two basic sets of network criteria, as follows:

- Peer-to-Peer (P2P) File Sharing: This is the most popular, but controversial, method of file sharing because of the use of peer-to-peer software. Network computer users locate shared data with third-party software. P2P file sharing allows users to directly access, download and edit files. Some third-party software facilitates P2P sharing by collecting and segmenting large files into smaller pieces.
- File Hosting Services: This P2P file-sharing alternative provides a broad selection of popular online material. These services are quite often used with Internet collaboration methods, including email, blogs, forums, or other mediums, where direct download links from the file hosting services can be included. These service websites usually host files to enable users to download them.

Once users download or make use of a file using a file-sharing network, their computer also becomes a part of that network, allowing other users to download

files from the user's computer. File sharing is generally illegal, with the exception of sharing material that is not copyrighted or proprietary. Another issue with file-sharing applications is the problem of spyware or adware, as some file-sharing websites have placed spyware programs in their websites. These spyware programs are often installed on users' computers without their consent and awareness.

File Systems

File system is the part of the operating system which is responsible for file management. It provides a mechanism to store the data and access to the file contents including data and programs. Some Operating systems treats everything as a file for example Ubuntu.

The File system takes care of the following issues

- **File Structure**

We have seen various data structures in which the file can be stored. The task of the file system is to maintain an optimal file structure.

- **Recovering Free space**

Whenever a file gets deleted from the hard disk, there is a free space created in the disk. There can be many such spaces which need to be recovered in order to reallocate them to other files.

- **disk space assignment to the files**

The major concern about the file is deciding where to store the files on the hard disk. There are various disks scheduling algorithm which will be covered later in this tutorial.

- **tracking data location**

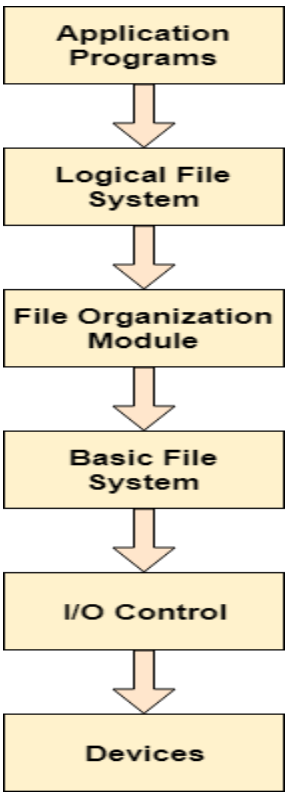
A File may or may not be stored within only one block. It can be stored in the non contiguous blocks on the disk. We need to keep track of all the blocks on which the part of the files reside.

File System Structure

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



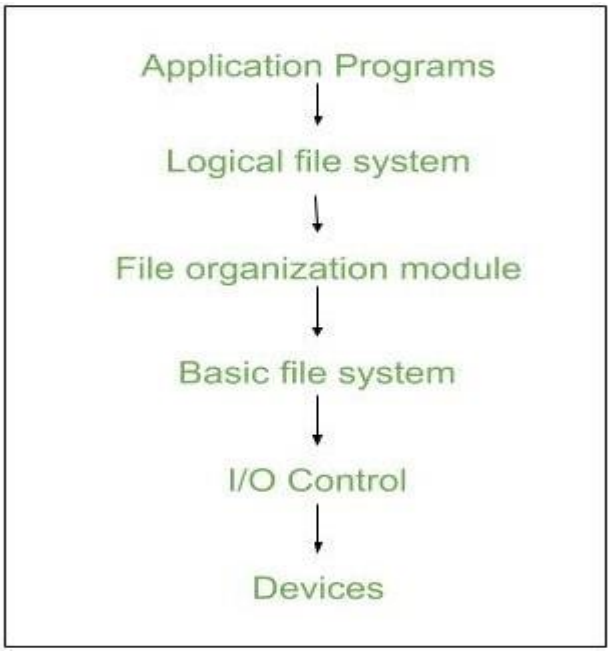
- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.

- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

File System Implementation

A file is a collection of related information. The file system resides on secondary storage and provides efficient and convenient access to the disk by allowing data to be stored, located, and retrieved.

File system organized in many layers :



- **I/O Control level –**
  Device drivers acts as interface between devices and Os, they help to transfer data between disk and main memory. It takes block number a input and as output it gives low level hardware specific instruction.
  /li>

- **Basic file system –**
  It Issues general commands to device driver to read and write physical blocks on disk.It manages the memory buffers and caches. A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.

- **File organization Module –**
  It has information about files, location of files and their logical and physical blocks.Physical blocks do not match with logical numbers of logical block numbered from 0 to N. It also has a free space which tracks unallocated blocks.

- **Logical file system –**
  It manages metadata information about a file i.e includes all details about a file except the actual contents of file. It also maintains via file control blocks. File control block (FCB) has information about a file – owner, size, permissions, location of file contents.

Advantages :

1. Duplication of code is minimized.
2. Each file system can have its own logical file system.

Disadvantages :
If we access many files at same time then it results in low performance.

We can **implement** file system by using two types data structures :

1. On-disk Structures –

Generally they contain information about total number of disk blocks, free disk blocks, location of them and etc. Below given are different on-disk structures :

1. **Boot Control Block –**
   It is usually the first block of volume and it contains information needed to boot an operating system.In UNIX it is called boot block and in NTFS it is called as partition boot sector.

2. **Volume Control Block –**
   It has information about a particular partition ex:- free block count, block size and block pointers etc.In UNIX it is called super block and in NTFS it is stored in master file table.

3. **Directory Structure –**
   They store file names and associated inode numbers.In UNIX, includes file names and associated file names and in NTFS, it is stored in master file table.

4. **Per-File FCB –**
   It contains details about files and it has a unique identifier number to allow association with directory entry. In NTFS it is stored in master file table.

File Control Block (FCB)
File Permissions
File dates (create, access, write)
File owner, group ,ACL
File size
File data blocks or pointers to file data blocks

2. In-Memory Structure :

They are maintained in main-memory and these are helpful for file system management for caching. Several in-memory structures given below :

5. **Mount Table –**
   It contains information about each mounted volume.
6. **Directory-Structure cache –**
   This cache holds the directory information of recently accessed directories.
7. **System wide open-file table –**
   It contains the copy of FCB of each open file.
8. **Per-process open-file table –**
   It contains information opened by that particular process and it maps with appropriate system wide open-file.

Directory Implementation :

9. **Linear List –**
   It maintains a linear list of filenames with pointers to the data blocks.It is time-consuming also.To create a new file, we must first search the directory to be sure that no existing file has the same name then we add a file at end of the directory.To delete a file, we search the directory for the named file and release the space.To reuse the directory entry either we can mark the entry as unused or we can attach it to a list of free directories.
10. **Hash Table –**
    The hash table takes a value computed from the file name and returns a pointer to the file. It decreases the directory search time. The insertion and deletion process of files is easy. The major difficulty is hash tables are its generally fixed size and hash tables are dependent on hash function on that size.

Directory Implementation

There is the number of algorithms by using which, the directories can be implemented. However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system.

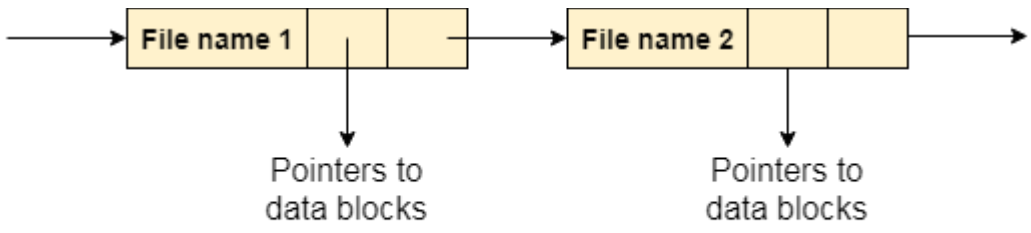
The directory implementation algorithms are classified according to the data structure they are using. There are mainly two algorithms which are used in these days.

1. Linear List

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

Characteristics

1. When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.



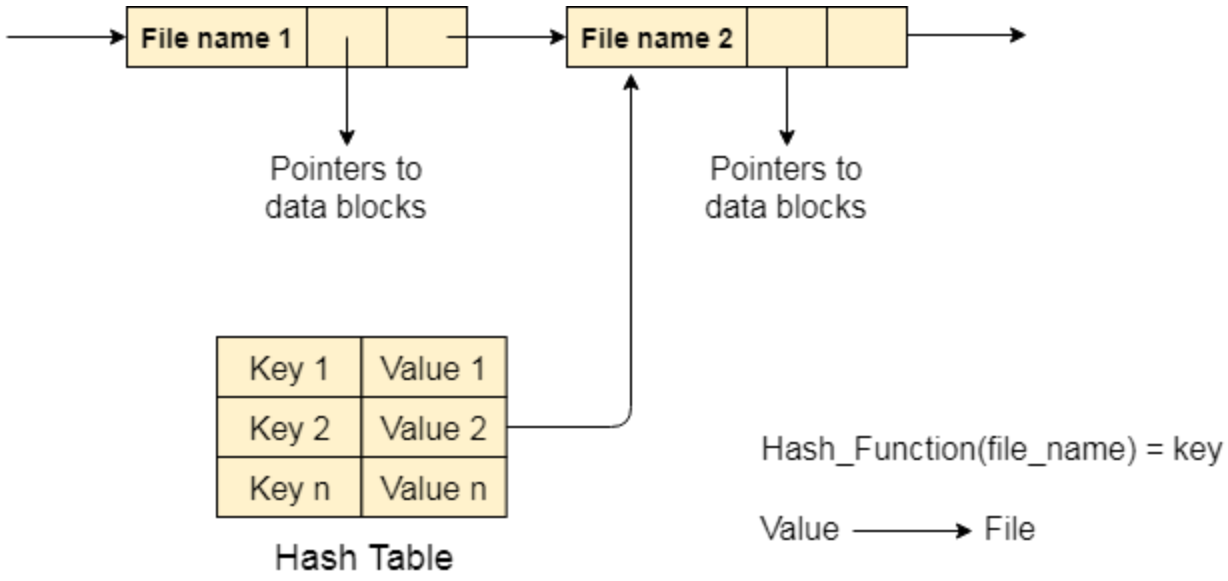
Linear List

2. Hash Table

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.



Allocation Methods

There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

There are following methods which can be used for allocation.

1. Contiguous Allocation.

2. Extents
3. Linked Allocation
4. Clustering
5. FAT
6. Indexed Allocation
7. Linked Indexed Allocation
8. Multilevel Indexed Allocation
9. Inode

Free Space Management

A file system is responsible to allocate the free blocks to the file therefore it has to keep track of all the free blocks present in the disk. There are mainly two approaches by using which, the free blocks in the disk are managed.

1. Bit Vector

In this approach, the free space list is implemented as a bit map vector. It contains the number of bits where each bit represents each block.

If the block is empty then the bit is 1 otherwise it is 0. Initially all the blocks are empty therefore each bit in the bit map vector contains 1.

LAs the space allocation proceeds, the file system starts allocating blocks to the files and setting the respective bit to 0.

2. Linked List

It is another approach for free space management. This approach suggests linking together all the free blocks and keeping a pointer in the cache which points to the first free block.

Therefore, all the free blocks on the disks will be linked together with a pointer. Whenever a block gets allocated, its previous free block will be linked to its next free block.

File System-Efficiency and Performance

Efficiency

The efficient use of disk space depends heavily on the disk allocation and directory algorithms in use. For instance, UNIX inodes are preallocated on a volume. Even an "empty" disk has a percentage of its space lost to inodes. However, by preallocating the inodes and. spreading them across the volume, we improve the file system's performance. This improved performance results from the UNIX allocation and free-space algorithms, which try to keep a file's data blocks near that file's inode block to reduce seek time. As another example, let's reconsider the clustering scheme discussed in Section 11.4, which aids in file-seek and file-transfer performance at the cost of internal fragmentation.

To reduce this fragmentation, BSD UNIX varies the cluster size as a file grows. Large clusters are used where they can be filled, and small clusters are used for small files and the last cluster of a file. This system is described in Appendix A. The types of data normally kept in a file's directory (or inode) entry also require consideration. Commonly, a 'last write date" is recorded to supply information to the user and, to determine whether the file needs to be backed up. Some systems also keep a "last access date," so that a user can determine when the file was last read.

The result of keeping this information is that, whenever the file is read, a field in the directory structure must be written to. That means the block must be read into memory, a section changed, and the block written back out to disk, because operations on disks occur only in block (or cluster) chunks. So any time a file is opened for reading, its directory entry must be read and written as well. This requirement can be inefficient for frequently accessed files, so we must weigh its benefit against its performance cost when designing a file system. Generally,

every data item associated with a file needs to be considered for its effect on efficiency and performance.

As an example, consider how efficiency is affected by the size of the pointers used to access data. Most systems use either 16- or 32-bit pointers throughout the operating system. These pointer sizes limit the length of a file to either 2 16 (64 KB) or 232 bytes (4 GB). Some systems implement 64-bit pointers to increase this limit to 264 bytes, which is a very large number indeed. However, 64-bit pointers take more space to store and in turn make the allocation and free-space-management methods (linked lists, indexes, and so on) use more disk space. One of the difficulties in choosing a pointer size, or indeed any fixed allocation size within an operating system, is planning for the effects of changing technology. Consider that the IBM PC XT had a 10-MB hard drive and an MS-DOS file system that could support only 32 MB. (Each FAT entry was 12 bits, pointing to an 8-KB cluster.)

As disk capacities increased, larger disks had to be split into 32-MB partitions, because the file system could not track blocks beyond 32 MB. As hard disks with capacities of over 100 MB became common, most disk controllers include local memory to form an on-board cache that is large enough to store entire tracks at a time. Once a seek is performed, the track is read into the disk cache starting at the sector under the disk head (reducing latency time).

The disk controller then transfers any sector requests to the operating system. Once blocks make it from the disk controller into main memory, the operating system may cache the blocks there. Some systems maintain a separate section of main memory for a buffer cache, where blocks are kept under the assumption that they will be used again shortly. Other systems cache file data using a page cache.

The page cache uses virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks. Caching file data using virtual addresses is far more efficient than caching through physical disk blocks, as accesses interface with virtual memory rather than the file system. Several systems—including

Solaris, Linux, and Windows NT, 2000, and XP—use page caching to cache both process pages and file data. This is known as unified virtual memory. Some versions of UNIX and Linux provide a unified buffer cache. To illustrate the benefits of the unified buffer cache, consider the two alternatives for opening and accessing a file. One approach is to use memory mapping (Section 9.7); the second is to use the standard system calls readO and write 0 . Without a unified buffer cache, we have a situation similar to Figure 11.11.

Here, the read() and write () system calls go through the buffer cache. The memory-mapping call, however, requires using two caches—the page cache and the buffer cache. A memory mapping proceeds by reading in disk blocks from the file system and storing them in the buffer cache. Because the virtual memory system does not interface with the buffer cache, the contents of the file in the buffer cache must be copied into the page cache. This situation is known as double caching and requires caching file-system data twice. Not only does it waste memory but it also wastes significant CPU and I/O cycles due to the extra data movement within, system memory.

In add ition, inconsistencies between the two caches can result in corrupt files. In contrast, when a unifiedthe disk data structures and algorithms in MS-DOS had to be modified to allow larger file systems. (Each FAT entry was expanded to 16 bits and later to 32 bits.) The initial file-system decisions were made for efficiency reasons; however, with the advent of MS-DOS version 4, millions of computer users were inconvenienced when they had to switch to the new, larger file system. Sun's ZFS file system uses 128-bit pointers, which theoretically should never need to be extended. (The minimum mass of a device capable of storing 2'2S bytes using atomic-level storage would be about 272 trillion kilograms.) As another example, consider the evolution of Sun's Solaris operating system.

Originally, many data structures were of fixed length, allocated at system startup. These structures included the process table and the open-file table. When the process table became full, no more processes could be created. When the file table became full, no more files could be opened. The system would fail to provide services to users. Table sizes could be increased only by recompiling the

kernel and rebooting the system. Since the release of Solaris 2, almost all kernel structures have been allocated dynamically, eliminating these artificial limits on system performance. Of course, the algorithms that manipulate these tables are more complicated, and the operating system is a little slower because it must dynamically allocate and deallocate table entries; but that price is the usual one for more general, functionality.

Performance

Even after the basic file-system algorithms have been selected, we can still improve performance in several ways. As will be discussed in Chapter 13, most disk controllers include local memory to form an on-board cache that is large enough to store entire tracks at a time. Once a seek is performed, the track is read into the disk cache starting at the sector under the disk head (reducing latency time). The disk controller then transfers any sector requests to the operating system. Once blocks make it from the disk controller into main memory, the operating system may cache the blocks there. Some systems maintain a separate section of main memory for a buffer cache, where blocks are kept under the assumption that they will be used again shortly. Other systems cache file data using a page cache.

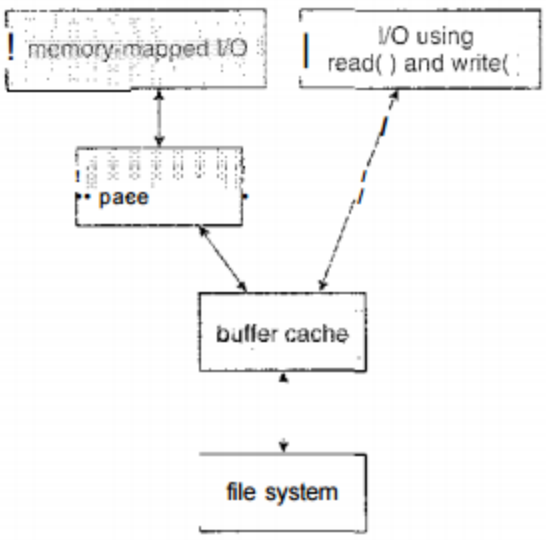


Figure 11.11 I/O without a unified buffer cache.

The page cache uses virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks. Caching file data using virtual addresses is far more efficient than caching through physical disk blocks, as accesses interface with virtual memory rather than the file system. Several systems—including Solaris, Linux, and Windows NT, 2000, and XP—use page caching to cache both process pages and file data. This is known as unified virtual memory. Some versions of UNIX and Linux provide a unified buffer cache.

To illustrate the benefits of the unified buffer cache, consider the two alternatives for opening and accessing a file. One approach is to use memory mapping (Section 9.7); the second is to use the standard system calls readO and write 0 .

Without a unified buffer cache, we have a situation similar to Figure 11.11. Here, the read() and write () system calls go through the buffer cache. The memory-mapping call, however, requires using two caches—the page cache and the buffer cache. A memory mapping proceeds by reading in disk blocks from the file system and storing them in the buffer cache. Because the virtual memory system does not interface with the buffer cache, the contents of the file in the buffer cache must be copied into the page cache.

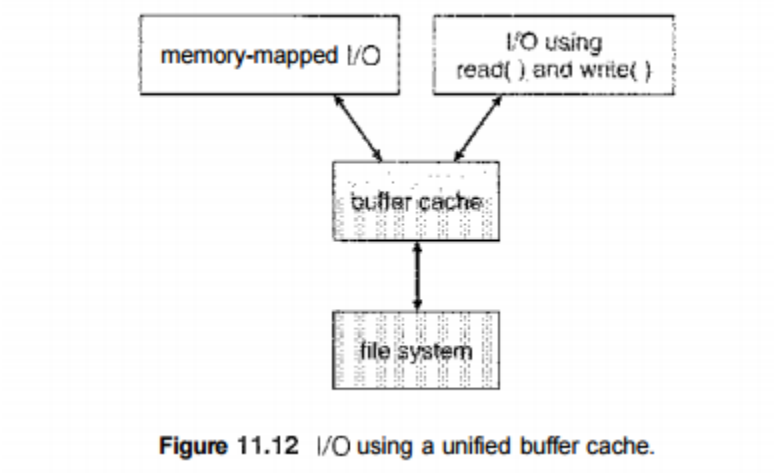


Figure 11.12 I/O using a unified buffer cache.

This situation is known as double caching and requires caching file-system data twice. Not only does it waste memory but it also wastes significant CPU and I/O cycles due to the extra data movement within, system memory. In add ition, inconsistencies between the two caches can result in corrupt files. In contrast,

when a unified buffer cache is provided, both memory mapping and the read () and write () system calls use the same page cache. This has the benefit of avoiding double caching, and it allows the virtual memory system to manage file-system data. The unified buffer cache is shown in Figure 11.12. Regardless of whether we are caching disk blocks or pages (or both), LEU (Section 9.4.4) seems a reasonable general-purpose algorithm for block or page replacement. However, the evolution of the Solaris page-caching algorithms reveals the difficulty in choosing an algorithm. Solaris allows processes and the page cache to share unused inemory.

Versions earlier than Solaris 2.5.1 made no distinction between allocating pages to a process and allocating them to the page cache. As a result, a system performing many I/O operations used most of the available memory for caching pages. Because of the high rates of I/O, the page scanner (Section 9.10.2) reclaimed pages from processes— rather than from the page cache—when free memory ran low. Solaris 2.6 and Solaris 7 optionally implemented priority paging, in which the page scanner gives priority to process pages over the page cache. Solaris 8 applied a fixed limit to process pages and the file-system page cache, preventing either from forcing the other out of memory. Solaris 9 and 10 again changed the algorithms to maximize memory use and minimize thrashing. This real-world example shows the complexities of performance optimizing and caching.

There are other issvies that can affect the performance of I/O such as whether writes to the file system occur synchronously or asynchronously. Synchronous writes occur in the order in which the disk subsystem receives them, and the writes are not buffered. Thus, the calling routine must wait for the data to reach the disk drive before it can proceed. Asynchronous writes are done the majority of the time. In an asynchronous write, the data are stored in the cache, and control returns to the caller. Metadata writes, among others, can be synchronous.

Operating systems frequently include a flag in the open system call to allow a process to request that writes be performed synchronously. For example, databases use this feature for atomic transactions, to assure that data reach

stable storage in the required order. Some systems optimize their page cache by using different replacement algorithms, depending on the access type of the file.

A file being read or written sequentially should not have its pages replaced in LRU order, because the most 11.7 Recovery 435 recently used page will be used last, or perhaps never again. Instead, sequential access can be optimized by techniques known as free-behind and read-ahead. Free-behind removes a page from the buffer as soon as the next page is requested. The previous pages are not likely to be used again and waste buffer space. With read-ahead, a requested page and several subsequent pages are read and cached. These pages are likely to be requested after the current page is processed.

Retrieving these data from the disk in one transfer and caching them saves a considerable amount of time. One might think a track cache on the controller eliminates the need for read-ahead on a multiprogrammed system. However, because of the high latency and overhead involved in making many small transfers from the track cache to main memory, performing a read-ahead remains beneficial. The page cache, the file system, and the disk drivers have some interesting interactions. When data are written to a disk file, the pages are buffered in the cache, and the disk driver sorts its output queue according to disk address. These two actions allow the disk driver to minimize disk-head seeks and to write data at times optimized for disk rotation.

Unless synchronous writes are required, a process writing to disk simply writes into the cache, and the system asynchronously writes the data to disk when convenient. The user process sees very fast writes. When data are read from a disk file, the block I/O system does some read-ahead; however, writes are much more nearly asynchronous than are reads. Thus, output to the disk through the file system is often faster than is input for large transfers, counter to intuition.

File System-Recovery

Recovery

Files and directories are kept both in main memory and on disk, and care must taken to ensure that system failure does not result in loss of data or in data inconsistency. We deal with these issues in the following sections.

Consistency Checking

As discussed in Section 11.3, some directory information is kept in main memory (or cache) to speed up access. The directory information in main memory is generally more up to date than is the corresponding information on the disk, because cached directory information is not necessarily written to disk as soon as the update takes place.

Magnetic disks sometimes fail, and care must be taken to ensure that the data lost in such a failure are not lost forever. To this end, system programs can be used to back up data from disk to another storage device, such as a floppy disk, magnetic tape, optical disk, or other hard disk.

Recovery from the loss of an individual file, or of an entire disk, may then be a matter of restoring the data from backup. To minimize the copying needed, we can use information from each file's directory entry. For instance, if the backup program knows when the last backup of a file was done, and the file's last write date in the directory indicates that the file has not changed since that date, then the file does not need to be copied again. A typical backup schedule may then be as follows:

• Day 1. Copy to a backup medium all files from the disk. This is called a full backup.

• Day 2. Copy to another medium all files changed since day 1. This is an incremental backup.

Day 3. Copy to another medium all files changed since day 2.

• Day N. Copy to another medium all files changed since day N— 1. Then go back to Day 1. The new cycle can have its backup written over the previous set or onto a new set of backup media.

In this manner, we can restore an entire disk by starting restores with the full backup and continuing through each of the incremental backups. Of course, the larger the value of N, the greater the number of tapes or disks that must be read for a complete restore. An added advantage of this backup cycle is that we can restore any file accidentally deleted during the cycle by retrieving the deleted file from the backup of the previous day.

The length of the cycle is a compromise between the amount of backup medium needed and the number of days back from which a restore can be done. To decrease the number of tapes that must be read, to do a restore, an option is to perform a full backup and then each day back up all files that have changed since the full backup. In this way, a restore can be done via the most recent incremental backup and. the full backup, with no other incremental backups needed. The trade-off is that more files will be modified each day, so each successive incremental backup involves more files and more backup media.

A user may notice that a particular file is missing or corrupted long after the damage was done. For this reason, we usually plan to take a full backup from time to time that will be saved "forever." It is a good idea to store these permanent backups far away from the regular backups to protect against hazard, such as a fire that destroys the computer and all the backups too. And if the backup cycle reuses media, we must take care not to reuse the media too many times—if the media wear out, it might not be possible to restore any data from the backups.

I/O Hardware

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories −

- **Block devices** − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices** − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

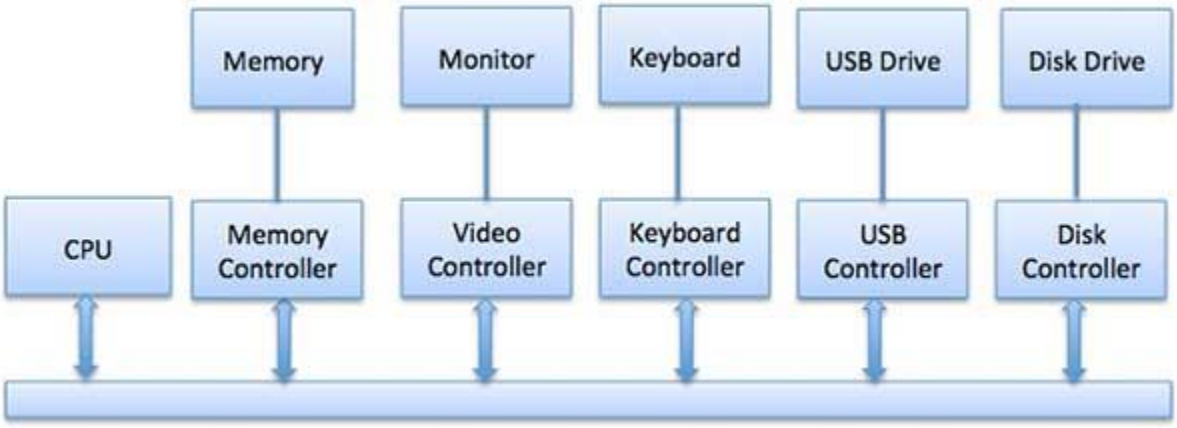
Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Synchronous vs asynchronous I/O

- **Synchronous I/O** – In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O** – I/O proceeds concurrently with CPU execution

Communication to I/O Devices

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

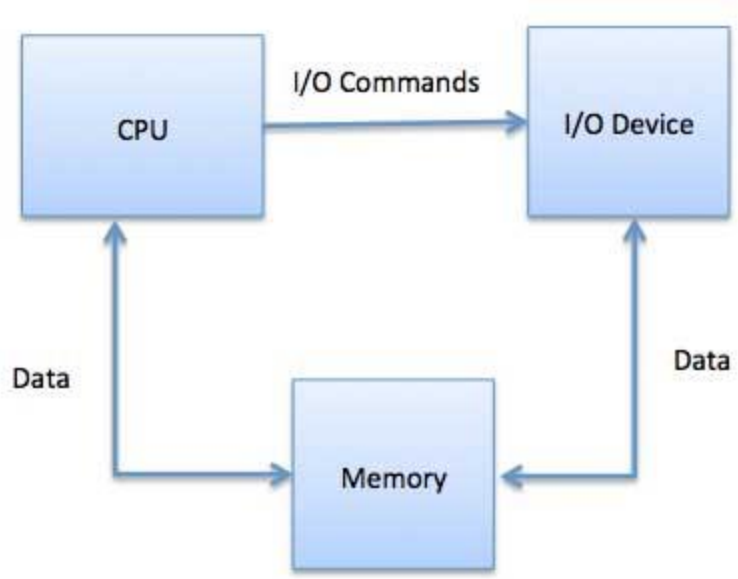
Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory

locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

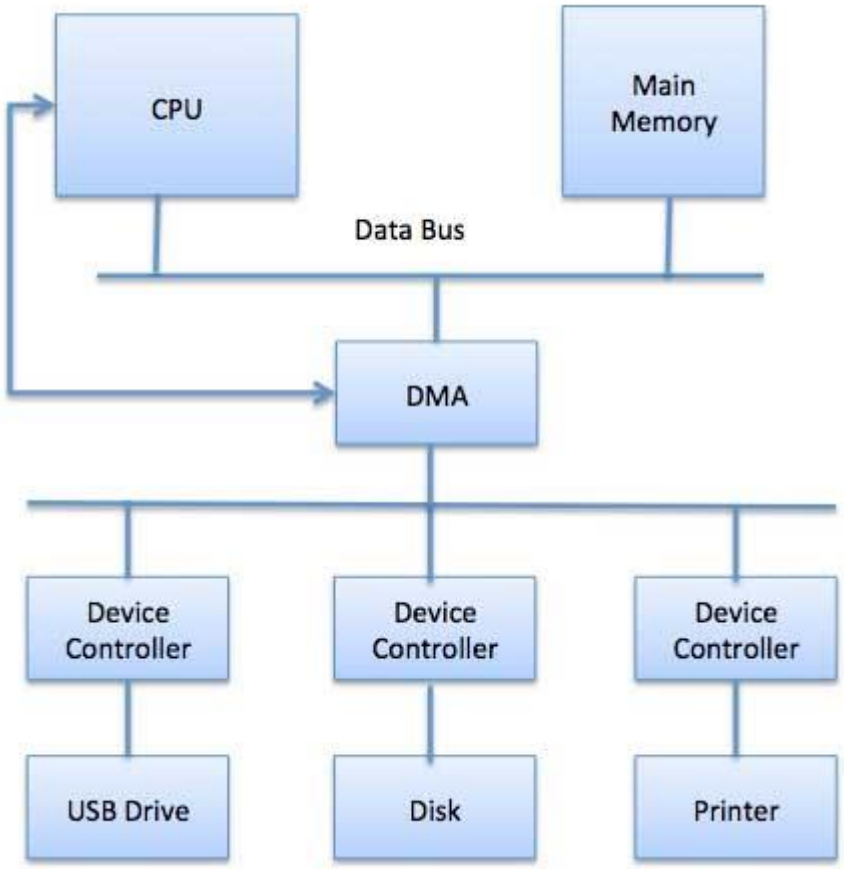
Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved

at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



The operating system uses the DMA hardware as follows −

Step	Description

1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.
6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

Polling vs Interrupts I/O

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as **polling** and **interrupts**. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

Polling I/O

Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.

Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

Application I/O interface

In this section, we discuss structuring techniques and interfaces for the operating system that enable I/O devices to be treated in a standard, uniform way. We explain, for instance, how an application can open a file on a disk without knowing what kind of disk it is and how new disks and other devices can be added to a computer without disruption of the operating system. Like other complex software-engineering problems, the approach here involves abstraction, encapsulation, and software layering. Specifically we can abstract away the detailed differences in I/O devices by identifying a fewgeneral kinds. Each general kind is accessed through a standardized set of functions—an interface. The differences are encapsulated in kernel modules called device drivers that internally are custom-tailored to each device but that export one of the standard interfaces.

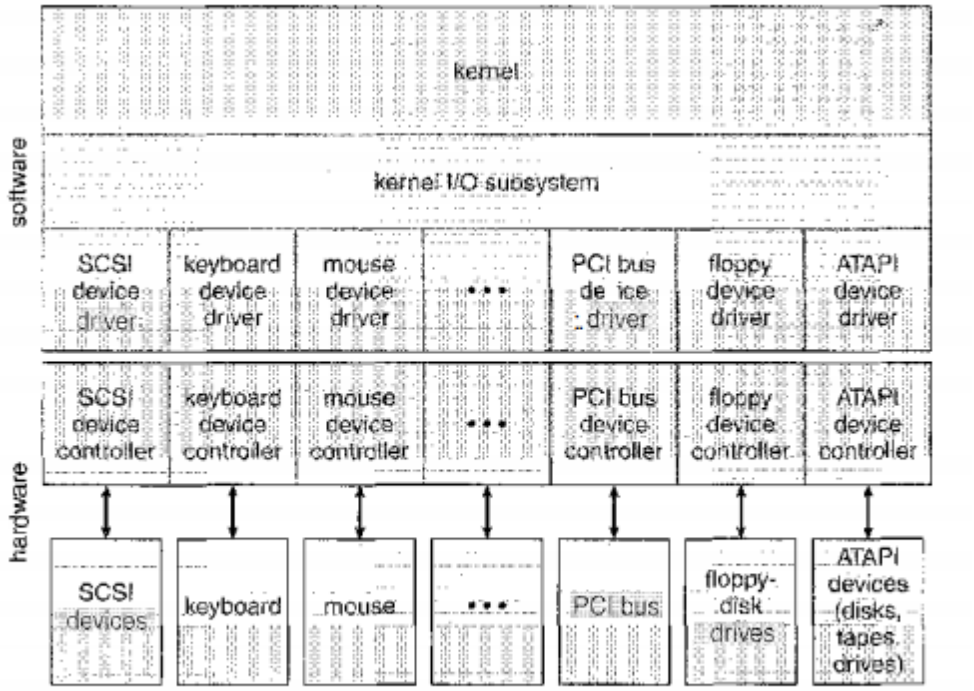


Figure 13.6 A kernel I/O structure.

Figure 13.6 illustrates how the I/O-related portions of the kernel are structured in software layers. The purpose of the device-driver layer is to hide the differences among device controllers from the I/O subsystem of the kernel, much as the I/O system calls encapsulate the behavior of devices in a few generic classes that hide hardware differences from applications. Making the I/O subsystem independent of the hardware simplifies the job of the operating-system developer. It also benefits the hardware manufacturers. They either design new devices to be compatible with an existing host controller interface (such as SCSI-2), or they write device drivers to interface the new hardware to popular operating systems. Thus, we can attach new peripherals to a computer without waiting for the operating-system vendor to develop support code. Unfortunately for device-hardware manufacturers, each type of operating system has its own standards for the device-driver interface. A given device may ship with multiple device drivers—for instance, drivers for MS-DOS, Windows 95/98, Windows NT/2000, and Solaris. Devices vary on many dimensions, as illustrated in Figure 13.7.

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Figure 13.7 Characteristics of I/O devices.

• Character-stream or block. A character-stream, device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.

• Sequential or random-access. A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.

• Synchronous or asynchronous. A synchronous device performs data transfers with predictable response times. An asynchronous device exhibits irregular or unpredictable response times.

Sharable or dedicated. A sharable device can be used concurrently by several processes or threads; a dedicated device cannot. 8 Speed of operation. Device speeds range from a few bytes per second to a few gigabytes per second.

• Read-write, read only, or write only. Some devices perform both input and output, but others support only one data direction. For the purpose of application access, many of these differences are hidden by the operating system, and the devices are grouped into a few conventional types. The resulting styles of device access have been found to be useful and broadly applicable. Although the exact

system calls may differ across operating systems, the device categories are fairly standard. The major access conventions include block I/O, character-stream I/O, memory-mapped file access, and network sockets. Operating systems also provide special system calls to access a few additional devices, such as a time-of-day clock and a timer. Some operating systems provide a set of system calls for graphical display, video, and audio devices. Most operating systems also have an escape (or back door) that transparently passes arbitrary commands from an application to a device driver. In UNIX, this system call is ioctl O (for "I/O" control). The ioctl O system call enables an application to access any functionality that can be implemented by any device driver, without the need to invent a new system call. The ioctl O system call has three arguments.

The first is a file descriptor that connects the application to the driver by referring to a hardware device managed by that driver. The second is an integer that selects one of the commands implemented in the driver. The third is a pointer to an arbitrary data structure in memory that enables the application and driver to communicate any necessary control information or data.

Block and Character Devices

The block-device interface captures all the aspects necessary for accessing disk drives and other block-oriented devices. The device is expected to understand commands such as read () and wr it e (); if it is a random-access device, it is also expected to have a seek() command to specify which block to transfer next. Applications normally access such a device through a file-system interface. We can see that readO, write (), and seekO capture the essential behaviors of block-storage devices, so that applications are insulated from the low-level differences among those devices.

The operating system itself, as well as special applications such as databasemanagement systems, may prefer to access a block device as a simple linear array of blocks. This mode of access is sometimes called raw I/O. If the application performs its own buffering, then using a file system would cause extra, unneeded buffering. Likewise, if an application provides its own locking of

file blocks or regions, then any operating-system locking services would be redundant at the least and contradictory at the worst.

To avoid these conflicts, raw-device access passes control of the device directly to the application, letting the operating system step out of the way. Unfortunately, no operating-system services are then performed on this device. A compromise that is becoming common is for the operating system to allow a mode of operation on a file that disables buffering and locking.

In the UNIX world, this is called direct I/O. Memory-mapped file access can be layered on top of block-device drivers. Rather than offering read and write operations, a memory-mapped interface provides access to disk storage via an array of bytes in main memory. The system call that maps a file into memory returns the virtual memory address that contains a copy of the file. The actual data transfers are performed only when needed to satisfy access to the memory image. Because the transfers are handled by the same mechanism as that used for demand-paged virtual memory access, memory-mapped I/O is efficient. Memory mapping is also convenient for programmers—access to a memory-mapped file is as simple as reading from and writing to memory. Operating systems that offer virtual memory commonly use the mapping interface for kernel services. For instance, to execute a program, the operating system maps the executable into memory and then transfers control to the entry address of the executable.

The mapping interface is also commonly used for kernel access to swap space on disk. A keyboard is an example of a device that is accessed through a characterstream interface. The basic system calls in this interface enable an application to get() or putO one character. On top of this interface, libraries can be built that offer line-at-a-time access, with buffering and editing services (for example, when a user types a backspace, the preceding character is removed from the input stream). This style of access is convenient for input devices such as keyboards, mice, and modems that produce data for input "spontaneously" — that is, at times that cannot necessarily be predicted by the application. This

access style is also good for output devices such as printers and audio boards, which naturally fit the concept of a linear stream of bytes.

Network Devices

Because the performance and addressing characteristics of network I/O differ significantly from, those of disk I/O, most operating systems provide a network I/O interface that is different from the read 0 -writ e () -seekO interface used for disks. One interface available in many operating systems, including UNIX and Windows NT, is the network socket interface. Think of a wall socket for electricity: Any electrical appliance can be plugged in. By analogy, the system calls in the socket interface enable an application to create a socket, to connect a local socket to a remote address (which plugs this application into a socket created by another application), to 13.3 Application I/O Interface 509 listen for any remote application to plug into the local socket, and to send and receive packets over the connection.

To support the implementation of servers, the socket interface also provides a function called selec t () that manages a set of sockets. A call to selec t () returns information about which sockets have a packet waiting to be received and which sockets have room to accept a packet to be sent. The use of selec t () eliminates the polling and busy waiting that would otherwise be necessary for network I/O. These functions encapsulate the essential behaviors of networks, greatly facilitating the creation of distributed applications that can use any underlying network hardware and protocol stack.

Many other approaches to interprocess communication and network communication have been implemented. For instance, Windows NT provides one interface to the network interface card and a second interface to the network protocols (Section C.6). In UNIX, which has a long history as a proving ground for network technology, we find half-duplex pipes, full-duplex FIFOs, full-duplex STREAMS, message queues, and sockets. Information on UNIX networking is given in Appendix A (Section A.9). Clocks and Timers

Most computers have hardware clocks and timers that provide three basic functions:

- Give the current time.
- Give the elapsed time.

Set a timer to trigger operation X at time T. These functions are used heavily by the operating system, as well as by timesensitive applications. Unfortunately, the system calls that implement these functions are not standardized across operating systems. The hardware to measure elapsed time and to trigger operations is called a programmable interval timer. It can be set to wait a certain amount of time and then generate an interrupt, and it can be set to do this once or to repeat the process to generate periodic interrupts. The scheduler uses this mechanism to generate an interrupt that will preempt a process at the end of its time slice.

The disk I/O subsystem uses it to invoke the flushing of dirty cache buffers to disk periodically, and the network subsystem uses it to cancel operations that are proceeding too slowly because of network congestion or failures. Hie operating system may also provide an interface for user processes to use timers. The operating system can support more timer requests than the number of timer hardware channels by simulating virtual clocks. To do so, the kernel (or the timer device driver) maintains a list of interrupts wanted by its own routines and by user requests, sorted in earliest-time-first order. It sets the timer for the earliest time. When the timer interrupts, the kernel signals the requester and reloads the timer with the next earliest time.

On many computers, the interrupt rate generated by the hardware clock is between 18 and 60 ticks per second. This resolution is coarse, since a modern computer can execute hundreds of millions of instructions per second. The precision of triggers is limited by the coarse resolution of the timer, together with the overhead of maintaining virtual clocks. Furthermore, if the timer 510 Chapter 13 I/O Systems ticks are used to maintain the system tiine-of-day clock, the system? clock can drift. In most computers, the hardware clock is constructed

from a highfrequency counter. In some computers, the value of this counter can he read from a device register, in which case the counter can be considered a highresolution clock. Although this clock does not generate interrupts, it offers accurate measurements of time intervals.

Blocking and Nonblocking IO

Another aspect of the system-call interface relates to the choice between blocking I/O and nonblocking I/O. When an application issues a blocking system call, the execution of the application is suspended. The application is moved from the operating system's run queue to a wait queue. After the system call completes, the application is moved back to the run queue, where it is eligible to resume execution, at which time it will receive the values returned by the system call. The physical actions performed by I/O devices are generally asynchronous—they take a varying or unpredictable amount of time. Nevertheless, most operating systems use blocking system calls for the application interface, because blocking application code is easier to understand than nonblocking application code. Some user-level processes need nonblocking I/O.

One example is a user interface that receives keyboard and mouse input while processing and displaying data on the screen. Another example is a video application that reads frames from a file on disk while simultaneously decompressing and displaying the output on the display. One way an application writer can overlap execution with I/O is to write a multithreaded application. Some threads can perform blocking system calls, while others continue executing. The Solaris developers used this technique to implement a user-level library for asynchronous I/O, freeing the application writer from that task. Some operating systems provide nonblocking I/O system calls. A nonblocking call does not halt the execution of the application for an extended time. Instead, it returns quickly, with a return value that indicates how many bytes were transferred. An alternative to a nonblocking system call is an asynchronous system call. An asynchronous call returns immediately, without waiting for the I/O to complete. The application continues to execute its code.

The completion of the I/O at some future time is communicated to the application, either through the setting of some variable in the address space of the application or through the triggering of a signal or software interrupt or a call-back routine that is executed outside the linear control flow of the application. The difference between nonblocking and asynchronous system calls is that a nonblocking readQ returns immediately with whatever data are available — the full number of bytes requested, fewer, or none at all. An asynchronous read() call requests a transfer that will be performed in its entirety but that will complete at some future time. These two I/O methods are shown in Figure 13.8.

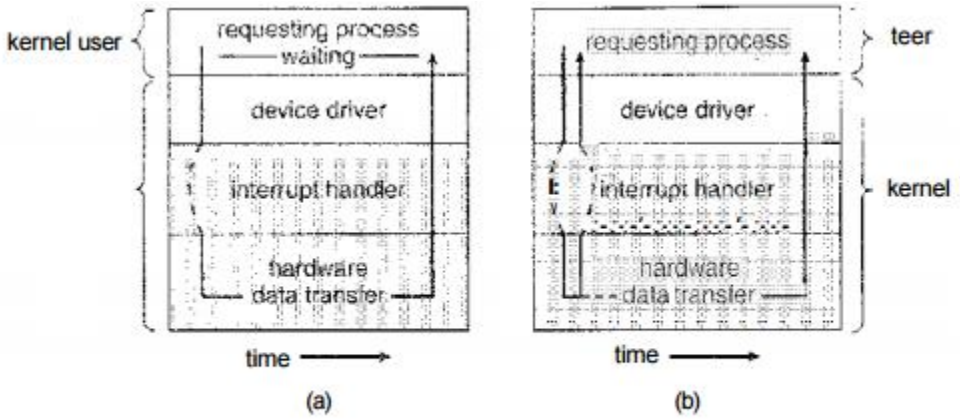


Figure 13.8 Two I/O methods: (a) synchronous and (b) asynchronous.

A good example of nonblocking behavior is the selec t () system call for network sockets. This system call takes an argument that specifies a maximum waiting time. By setting it to 0, an application can poll for network activity without blocking. But using select() introduces extra overhead, because the selec t () call only checks whether I/O is possible. For a data transfer, selectO must be followed by some kind of readO or writeO command. A variation on this approach, found in Mach, is a blocking multiple-read call. It specifies desired reads for several devices in one system call and returns as soon as any one of them completes.

Kernel I/O Subsystem

The kernel provides many services related to I/O. Several services such as scheduling, caching, spooling, device reservation, and error handling – are provided by the kernel, s I/O subsystem built on the hardware and device-driver

infrastructure. The I/O subsystem is also responsible for protecting itself from the errant processes and malicious users.

1. **I/O Scheduling –**
   To schedule a set of I/O request means to determine a good order in which to execute them. The order in which application issues the system call are the best choice. Scheduling can improve the overall performance of the system, can share device access permission fairly to all the processes, reduce the average waiting time, response time, turnaround time for I/O to complete.

OS developers implement scheduling by maintaining a wait queue of the request for each device. When an application issue a blocking I/O system call, The request is placed in the queue for that device. The I/O scheduler rearrange the order to improve the efficiency of the system.

2. **Buffering –**
   A buffer is a memory area that stores data being transferred between two devices or between a device and an application. Buffering is done for three reasons.

   1. First is to cope with a speed mismatch between producer and consumer of a data stream.
   2. The second use of buffering is to provide adaptation for data that have different data-transfer sizes.
   3. Third use of buffering is to support copy semantics for the application I/O, “copy semantic ” means, suppose that an application wants to write data on a disk that is stored in its buffer. it calls the **write()** system’s call, providing a pointer to the buffer and the integer specifying the number of bytes to write.

Q. After the system call returns, what happens if the application of the buffer changes the content of the buffer?

Ans. With copy semantic, the version of the data written to the disk is guaranteed to be the version at the time of the application system call.

3. **Caching –**
   A cache is a region of fast memory that holds a copy of data. Access to the cached copy is much easier than the original file. For instance, the instruction of the currently running process is stored on the disk, cached in physical memory, and copied again in the CPU's secondary and primary cache.

   The main difference between a buffer and a cache is that a buffer may hold only the existing copy of a data item, while cache, by definition, holds a copy on faster storage of an item that resides elsewhere.

4. **Spooling and Device Reservation –**
   A spool is a buffer that holds the output of a device, such as a printer that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixes together.

   The OS solves this problem by preventing all output continuing to the printer. The output of all applications is spooled in a separate disk file. When an application finishes printing then the spooling system queues the corresponding spool file for output to the printer.

5. **Error Handling –**
   An Os that uses protected memory can guard against many kinds of hardware and application errors, so that a complete system failure is not the usual result of each minor mechanical glitch, Devices, and I/O transfers can fail in many ways, either for transient reasons, as when a network becomes overloaded or for permanent reasons, as when a disk controller becomes defective.

6. **I/O Protection –**
   Errors and the issue of protection are closely related. A user process may

attempt to issue illegal I/O instructions to disrupt the normal function of a system. We can use the various mechanisms to ensure that such disruption cannot take place in the system.

To prevent illegal I/O access, we define all I/O instructions to be privileged instructions. The user cannot issue I/O instruction directly.

Transforming of I/O Requests to Hardware Operations

We know that there is handshaking between device driver and device controller but here question is that how operating system connects application request or we can say I/O request to set of network wires or to specific disk sector or we can say to hardware -operations.

To understand concept example which is as follows.

Example –
We are reading file from disk. The application we request for will refers to data by file name. Within disk, file system maps from file name through file-system directories to obtain space allocation for file. In MS-DOS, name of file maps to number that indicates as entry in file-access table, and that entry to table tells us that which disk blocks are allocated to file. In UNIX, name maps to inode number, and inode number contains information about space-allocation. But here question arises that how connection is made from file name to disk controller?

The method that is used by MS-DOS, is relatively simple operating system. The first part of MS-DOS file name, is preceding with colon, is string that identifies that there is specific hardware device.

The UNIX uses different method from MS-DOS. It represents device names in regular file-system name space. Unlike MS-DOS file name, which has colon separator, but UNIX path name has no clear separation of device portion. In fact, no part of path name is name of device. Unix has mount table that associates with prefixes of path names with specific hardware device names.

Modern operating systems gain significant flexibility from multiple stages of lookup tables in path between request and physical device stages controller. There is general mechanisms is which is used to pass request between application and drivers. Thus, without recompiling kernel, we can introduce new devices and drivers into computer. In fact, some operating system have the ability to load device drivers on demand. At the time of booting, system firstly probes hardware buses to determine what devices are present. It is then loaded to necessary drivers, accordingly I/O request.

The typical life cycle of blocking read request, is shown in the following figure. From figure, we can suggest that I/O operation requires many steps that together consume large number of CPU cycles.

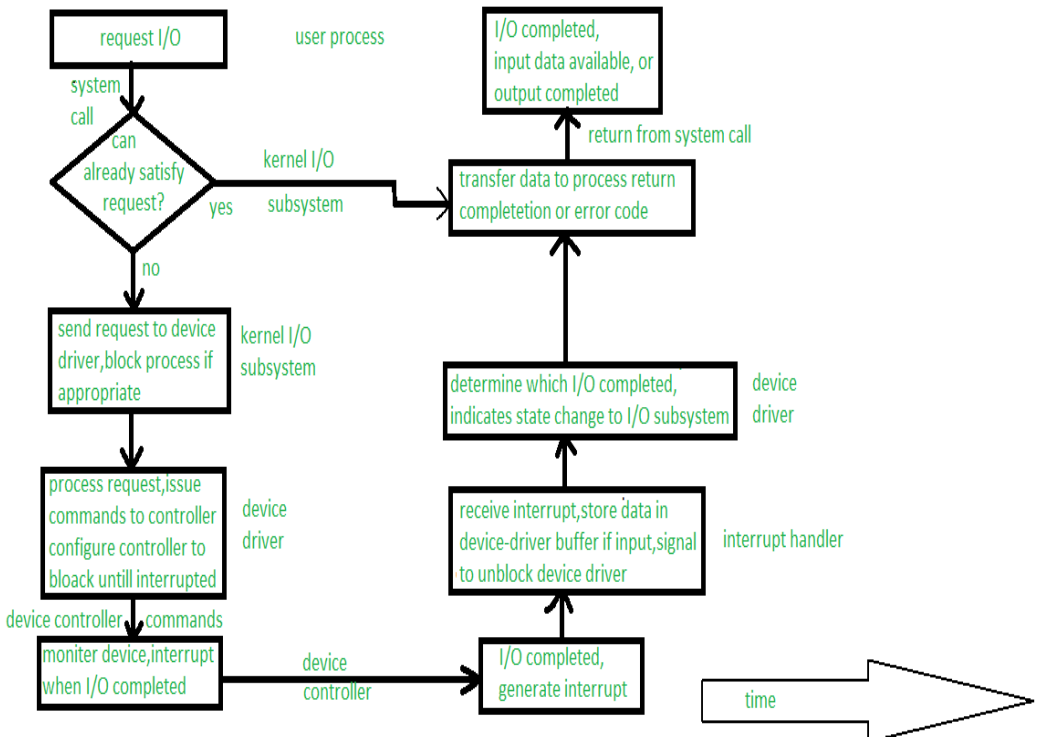


Figure – The life cycle of I/O request

1. **System call –**
   Whenever, any I/O request comes, process issues blocking read() system call to previously opened file descriptor of file. Basically, role of system-call code is to check parameters for correctness in kernel. If data we put in form

of input is already available in buffer cache, data is going to returned to process, and in that case I/O request is completed.

2. **Alternative approach if input is not available –**
   If the data is not available in buffer cache then physical I/O must be performed. The process is removes from run queue and is placed on wait queue for device, and I/O request is scheduled. After scheduling, I/O subsystem sends request to device driver via subroutine call or in-kernel message but it depends upon operating system by which mode request will send.

3. **Role of Device driver –**
   After receiving the request, device driver have to receive data and it will receive data by allocating kernel buffer space and after receiving data it will schedules I/O. After all this, command will be given to device controller by writing into device-control registers.

4. **Role of Device Controller –**
   Now, device controller operates device hardware. Actually, data transfer is done by device hardware.

5. **Role of DMA controller –**
   After data transfer, driver may poll for status and data, or it may have set up DMA transfer into kernel memory. The transfer is managed by DMA controller. At last when transfers complete, it will generates interrupt.

6. **Role of interrupt handler –**
   The interrupt is send to correct interrupt handler through interrupt-vector table. It store any necessary data, signals device driver, and returns from interrupt.

7. **Completion of I/O request –**
   When, device driver receives signal. This signal determines that I/O request has completed and also determines request's status, signals kernel I/O subsystem that request has been completed. After transferring data or

return codes to address space kernel moves process from wait queue back to ready queue.

8. **Completion of System call –**
   When process moves to ready queue it means process is unblocked. When the process is assigned to CPU, it means process resumes execution at completion of system call.

Operating System Security (OS Security)

Operating system security (OS security) is the process of ensuring OS integrity, confidentiality and availability.

OS security refers to specified steps or measures used to protect the OS from threats, viruses, worms, malware or remote hacker intrusions. OS security encompasses all preventive-control techniques, which safeguard any computer assets capable of being stolen, edited or deleted if OS security is compromised.

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

Operating System Security (OS Security)

OS security encompasses many different techniques and methods which ensure safety from threats and attacks. OS security allows different applications and programs to perform required tasks and stop unauthorized interference.

OS security may be approached in many ways, including adherence to the following:

- Performing regular OS patch updates

- Installing updated antivirus engines and software
- Scrutinizing all incoming and outgoing network traffic through a firewall
- Creating secure accounts with required privileges only (i.e., user management)

Protection

Protection refers to a mechanism which controls the access of programs, processes, or users to the resources defined by a computer system. We can take protection as a helper to multi programming operating system, so that many users might safely share a common logical name space such as directory or files.

Need of Protection:

- To prevent the access of unauthorized users and
- To ensure that each active programs or processes in the system uses resources only as the stated policy,
- To improve reliability by detecting latent errors.

Role of Protection:

The role of protection is to provide a mechanism that implement policies which defines the uses of resources in the computer system.Some policies are defined at the time of design of the system, some are designed by management of the system and some are defined by the users of the system to protect their own files and programs.

Every application has different policies for use of the resources and they may change over time so protection of the system is not only concern of the designer of the operating system. Application programmer should also design the protection mechanism to protect their system against misuse.

Policy is different from mechanism. Mechanisms determine how something will be done and policies determine what will be done.Policies are changed over time

and place to place. Separation of mechanism and policy is important for the flexibility of the system.

Access Matrix

Access Matrix is a security model of protection state in computer system. It is represented as a matrix. Access matrix is used to define the rights of each process executing in the domain with respect to each object. The rows of matrix represent domains and columns represent objects. Each cell of matrix represents set of access rights which are given to the processes of domain means each entry(i, j) defines the set of operations that a process executing in domain Di can invoke on object Oj.

	F1	F2	F3	PRINTER
D1	read		read	
D2				print
D3		read	execute	
D4	read write		read write	

According to the above matrix: there are four domains and four objects- three files(F1, F2, F3) and one printer. A process executing in D1 can read files F1 and F3. A process executing in domain D4 has same rights as D1 but it can also write on files. Printer can be accessed by only one process executing in domain D2. The mechanism of access matrix consists of many policies and semantic properties. Specifically, We must ensure that a process executing in domain Di can access only those objects that are specified in row i.

Policies of access matrix concerning protection involve which rights should be included in the (i, j)th entry. We must also decide the domain in which each process executes. This policy is usually decided by the operating system. The Users decide the contents of the access-matrix entries.

Association between the domain and processes can be either static or dynamic. Access matrix provides an mechanism for defining the control for this association between domain and processes. When we switch a process from one domain to another, we execute a switch operation on an object(the domain). We can control domain switching by including domains among the objects of the access matrix. Processes should be able to switch from one domain (Di) to another domain (Dj) if and only is a switch right is given to access(i, j).

	F1	F2	F3	PRINTER	D1	D2	D3	D4
D1	read		read			switch		
D2				print			switch	switch
D3		read	execute					
D4	read write		read write		switch			

According to the matrix: a process executing in domain D2 can switch to domain D3 and D4. A process executing in domain D4 can switch to domain D1 and process executing in domain D1 can switch to domain D2.

Access Control

Access control is a method of guaranteeing that users are who they say they are and that they have the appropriate access to company data.

At a high level, access control is a selective restriction of access to data. It consists of two main components: authentication and authorization, says Daniel Crowley, head of research for IBM's X-Force Red, which focuses on data security.

Authentication is a technique used to verify that someone is who they claim to be. Authentication isn't sufficient by itself to protect data, Crowley notes. What's needed is an additional layer, authorization, which determines whether a user should be allowed to access the data or make the transaction they're attempting.

Without authentication and authorization, there is no data security, Crowley says. "In every data breach, access controls are among the first policies investigated," notes Ted Wagner, CISO at SAP National Security Services, Inc. "Whether it be the inadvertent exposure of sensitive data improperly secured by an end user or the Equifax breach, where sensitive data was exposed through a public-facing web server operating with a software vulnerability, access controls are a key component. When not properly implemented or maintained, the result can be catastrophic."

Any organization whose employees connect to the internet—in other words, every organization today—needs some level of access control in place. "That's especially true of businesses with employees who work out of the office and require access to the company data resources and services," says Avi Chesla, CEO of cybersecurity firm empow.

Put another way: If your data could be of any value to someone without proper authorization to access it, then your organization needs strong access control, Crowley says.

Another reason for strong access control: Access mining

The collection and selling of access descriptors on the dark web is a growing problem. For example, a new report from Carbon Black describes how one cryptomining botnet, Smominru, mined not only cryptcurrency, but also sensitive information including internal IP addresses, domain information, usernames and passwords. The Carbon Black researchers believe it is "highly plausible" that this

threat actor sold this information on an "access marketplace" to others who could then launch their own attacks by remote access.

These access marketplaces "provide a quick and easy way for cybercriminals to purchase access to systems and organizations.... These systems can be used as zombies in large-scale attacks or as an entry point to a targeted attack," said the report's authors. One access marketplace, Ultimate Anonymity Services (UAS) offers 35,000 credentials with an average selling price of $6.75 per credential.

The Carbon Black researchers believe cybercriminals will increase their use of access marketplaces and access mining because they can be "highly lucrative" for them. The risk to an organization goes up if its compromised user credentials have higher privileges than needed.

Access control policy: Key considerations

Most security professionals understand how critical access control is to their organization. But not everyone agrees on how access control should be enforced, says Chesla. “Access control requires the enforcement of persistent policies in a dynamic world without traditional borders,” Chesla explains. Most of us work in hybrid environments where data moves from on-premises servers or the cloud to offices, homes, hotels, cars and coffee shops with open wi-fi hot spots, which can make enforcing access control difficult.

“Adding to the risk is that access is available to an increasingly large range of devices,” Chesla says, including PCs, laptops, smart phones, tablets, smart speakers and other internet of things (IoT) devices. “That diversity makes it a real challenge to create and secure persistency in access policies.”

In the past, access control methodologies were often static. “Today, network access must be dynamic and fluid, supporting identity and application-based use cases,” Chesla says.

A sophisticated access control policy can be adapted dynamically to respond to evolving risk factors, enabling a company that’s been breached to “isolate the relevant employees and data resources to minimize the damage,” he says.

Enterprises must assure that their access control technologies "are supported consistently through their cloud assets and applications, and that they can be smoothly migrated into virtual environments such as private clouds," Chesla advises. "Access control rules must change based on risk factor, which means that organizations must deploy security analytics layers using AI and machine learning that sit on top of the existing network and security configuration. They also need to identify threats in real-time and automate the access control rules accordingly."

4 Types of access control

Organizations must determine the appropriate **access control model** to adopt based on the type and sensitivity of data they're processing, says Wagner. Older access models include discretionary access control (DAC) and mandatory access control (MAC), role based access control (RBAC) is the most common model today, and the most recent model is known as attribute based access control (ABAC).

Discretionary access control (DAC)

With DAC models, the data owner decides on access. DAC is a means of assigning access rights based on rules that users specify.

Mandatory access control (MAC)

MAC was developed using a nondiscretionary model, in which people are granted access based on an information clearance. MAC is a policy in which access rights are assigned based on regulations from a central authority.

Role Based Access Control (RBAC)

RBAC grants access based on a user's role and implements key security principles, such as "least privilege" and "separation of privilege." Thus, someone attempting to access information can only access data that's deemed necessary for their role.

Attribute Based Access Control (ABAC)

In ABAC, each resource and user are assigned a series of attributes, Wagner explains. “In this dynamic method, a comparative assessment of the user’s attributes, including time of day, position and location, are used to make a decision on access to a resource.”

It’s imperative for organizations to decide which model is most appropriate for them based on data sensitivity and operational requirements for data access. In particular, organizations that process personally identifiable information (PII) or other sensitive information types, including Health Insurance Portability and Accountability Act (HIPAA) or Controlled Unclassified Information (CUI) data, must make access control a core capability in their security architecture, Wagner advises.

Access control solutions

A number of technologies can support the various access control models. In some cases, multiple technologies may need to work in concert to achieve the desired level of access control, Wagner says.

“The reality of data spread across cloud service providers and SaaS applications and connected to the traditional network perimeter dictate the need to orchestrate a secure solution,” he notes. “There are multiple vendors providing privilege access and identity management solutions that can be integrated into a traditional Active Directory construct from Microsoft. Multifactor authentication can be a component to further enhance security.”

Why authorization remains a challenge

Today, most organizations have become adept at authentication, says Crowley, especially with the growing use of multifactor authentication and biometric-based authentication (such as facial or iris recognition). In recent years, as high-profile data breaches have resulted in the selling of stolen password credentials on the dark web, security professionals have taken the need for multi-factor authentication more seriously, he adds.

Authorization is still an area in which security professionals "mess up more often," Crowley says. It can be challenging to determine and perpetually monitor who gets access to which data resources, how they should be able to access them, and under which conditions they are granted access, for starters. But inconsistent or weak authorization protocols can create security holes that need to be identified and plugged as quickly as possible.

Speaking of monitoring: However your organization chooses to implement access control, it must be constantly monitored, says Chesla, both in terms of compliance to your corporate security policy as well as operationally, to identify any potential security holes. "You should periodically perform a governance, risk and compliance review," he says. "You need recurring vulnerability scans against any application running your access control functions, and you should collect and monitor logs on each access for violations of the policy."

In today's complex IT environments, access control must be regarded as "a living technology infrastructure that uses the most sophisticated tools, reflects changes in the work environment such as increased mobility, recognizes the changes in the devices we use and their inherent risks, and takes into account the growing movement toward the cloud," Chesla says.

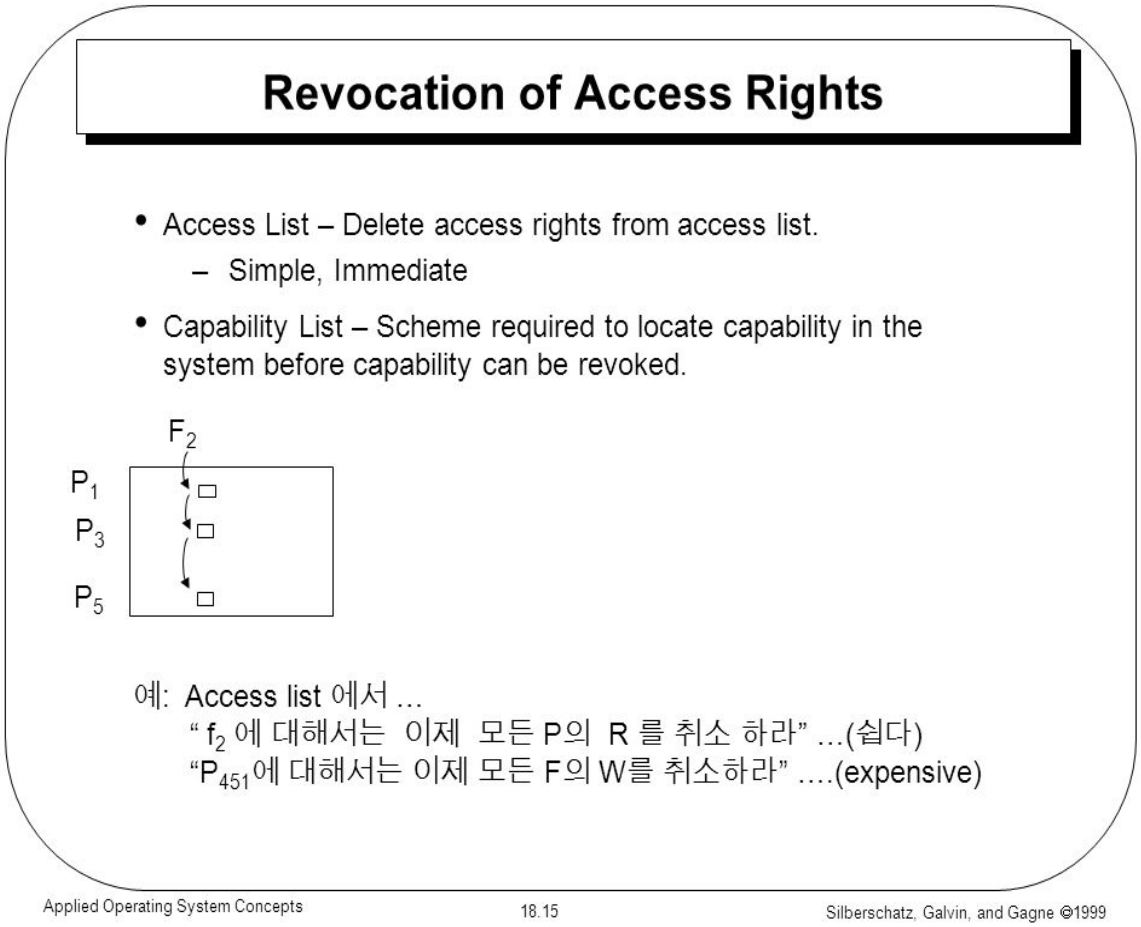
Revocation of Access Rights

Revocation of Access Rights

In a dynamic protection system, we may sometimes need to revoke access rights to objects shared by different users. Various questions about revocation may arise:

• Immediate versus delayed. Does revocation occur immediately/ or is it delayed? If revocation is delayed, can we find out when it will take place?

• Selective versus general. When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a select group of users whose access rights should be revoked?



• Partial versus total. Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?

• Temporary versus permanent. Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again? With an access-list scheme, revocation is easy.

The access list is searched for any access rights to be revoked, and they are deleted from the list. Revocation is immediate and can be general or selective, total or partial, and permanent or temporary. Capabilities, however, present a much more difficult revocation problem. Since the capabilities are distributed throughout the system, we must find them before we can revoke them. Schemes that implement revocation for capabilities include the following:

Reacquisition. Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability.

• Back-pointers. A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, we can follow these pointers, changing the capabilities as necessaryy. This scheme was adopted in the MULTICS system. It is quite general, but its implementation is costly.

• Indirection. The capabilities point indirectly, not directly, to the objects. Each capability points to a unique entry in a global table, which in turn points to the object. We implement revocation by searching the global table for the desired entry and deleting it. Then, when an access is attempted, the capability is found to point to an illegal table entry.

Table entries can be reused for other capabilities without difficulty, since both the capability and the table entry contain the unique name of the object. The object for a 14.8 Capability-Based Systems 547 capability and its table entry must match. This scheme was adopted in the CAL system. It does not allow selective revocation. Keys. A key is a unique bit pattern that can be associated with a capability. Tliis key is defined when the capability is created, and it can be neither modified nor inspected by the process owning the capability.

A master key is associated with each object; it can be defined or replaced with the set-key operation. When a capability is created, the current value of the master key is associated with the capability. When the capability is exercised, its key is compared with the master key. If the keys match, the operation is allowed to continue; otherwise, an exception condition is raised.

Revocation replaces the master key with a new value via the set-key operation, invalidating all previous capabilities for this object. This scheme does not allowr selective revocation, since only one master key is associated with each object. If

we associate a list of keys with each object, then selective revocation can be implemented.

Finally, we can group all keys into one global table of keys. A capability is valid only if its key matches some key in the global table. We implement revocation by removing the matching key from the table. With this scheme, a key can be associated with several objects, and several keys can be associated with each object, providing maximum flexibility. In key-based schemes, the operations of defining keys, inserting them into lists, and deleting them from lists should not be available to all users. In particular, it would be reasonable to allow only the owner of an object to set the keys for that object. This choice, however, is a policy decision that the protection system can implement but should not define.

Threats

Threats can be classified into the following two categories:

1. **Program Threats:**
   A program written by a cracker to hijack the security or to change the behaviour of a normal process.

2. **System Threats:**
   These threats involve the abuse of system services. They strive to create a situation in which operating-system resources and user files are misused. They are also used as a medium to launch program threats.

   System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

Types of Program Threats –

1. **Virus:**
   An infamous threat, known most widely. It is a self-replicating and a

malicious thread which attaches itself to a system file and then rapidly replicates itself, modifying and destroying essential files leading to a system breakdown.

Further, Types of computer viruses can be described briefly as follows:
– file/parasitic – appends itself to a file
– boot/memory – infects the boot sector
– macro – written in a high-level language like VB and affects MS Office files
– source code – searches and modifies source codes
– polymorphic – changes in copying each time
– encrypted – encrypted virus + decrypting code
– stealth – avoids detection by modifying parts of the system that can be used to detect it, like the read system
call
– tunneling – installs itself in the interrupt service routines and device drivers
– multipartite – infects multiple parts of the system

2. **Trojan Horse:**
   A code segment that misuses its environment is called a Trojan Horse. They seem to be attractive and harmless cover program but are a really harmful hidden program which can be used as the virus carrier. In one of the versions of Trojan, User is fooled to enter its confidential login details on an application. Those details are stolen by a login emulator and can be further used as a way of information breaches.

Another variance is Spyware, Spyware accompanies a program that the user has chosen to install and downloads ads to display on the user's system, thereby creating pop-up browser windows and when certain sites are visited by the user, it captures essential information and sends it over to the remote server. Such attacks are also known as **Covert Channels**.

3. **Trap Door:**
   The designer of a program or system might leave a hole in the software that only he is capable of using, the Trap Door works on the similar

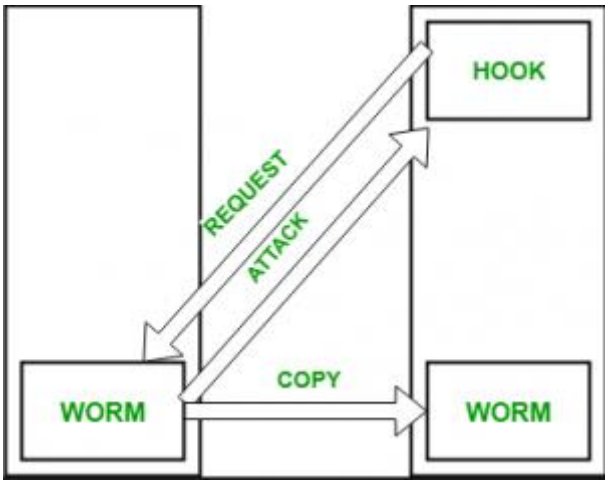
principles. Trap Doors are quite difficult to detect as to analyze them, one needs to go through the source code of all the components of the system.

4. **Logic Bomb:**
   A program that initiates a security attack only under a specific situation.

Types of System Threats –
Aside from the program threats, various system threats are also endangering the security of our system:

1. **Worm:**
   An infection program which spreads through networks. Unlike a virus, they target mainly LANs. A computer affected by a worm attacks the target system and writes a small program "hook" on it. This hook is further used to copy the worm to the target computer. This process repeats recursively, and soon enough all the systems of the LAN are affected. It uses the spawn mechanism to duplicate itself. The worm spawns copies of itself, using up a majority of system resources and also locking out all other processes.

The basic functionality of a the worm can be represented as:



2. **Port Scanning:**
   It is a means by which the cracker identifies the vulnerabilities of the system to attack. It is an automated process which involves creating a TCP/IP connection to a specific port. To protect the identity of the attacker,

port scanning attacks are launched from **Zombie Systems**, that is systems which were previously independent systems that are also serving their owners while being used for such notorious purposes.

3. **Denial of Service:**
   Such attacks aren't aimed for the purpose of collecting information or destroying system files. Rather, they are used for disrupting the legitimate use of a system or facility.
   These attacks are generally network based. They fall into two categories:
   – Attacks in this first category use so many system resources that no useful work can be performed.

For example, downloading a file from a website that proceeds to use all available CPU time.
– Attacks in the second category involves disrupting the network of the facility. These attacks are a result of the abuse of some fundamental TCP/IP principles. fundamental functionality of TCP/IP.

Security Measures Taken –
To protect the system, Security measures can be taken at the following levels:

- **Physical:**
  The sites containing computer systems must be physically secured against armed and malicious intruders. The workstations must be carefully protected.

- **Human:**
  Only appropriate users must have the authorization to access the system. Phishing(collecting confidential information) and Dumpster Diving(collecting basic information so as to gain unauthorized access) must be avoided.

- **Operating system:**
  The system must protect itself from accidental or purposeful security breaches.

- **Networking System:**
  Almost all of the information is shared between different systems via a network. Intercepting these data could be just as harmful as breaking into a computer. Henceforth, Network should be properly secured against such attacks.

Usually, Anti Malware programs are used to periodically detect and remove such viruses and threats. Additionally, to protect the system from the Network Threats, Firewall is also be used.

Network Security Threats

Types of Network Threats

Network-delivered threats are typically of two basic types:

- **Passive Network Threats**: Activities such as wiretapping and idle scans that are designed to intercept traffic traveling through the network.
- **Active Network Threats**: Activities such as Denial of Service (DoS) attacks and SQL injection attacks where the attacker is attempting to execute commands to disrupt the network's normal operation.

To execute a successful network attack, attackers must typically actively hack a company's infrastructure to exploit software vulnerabilities that allow them to remotely execute commands on internal operating systems. DoS attacks and shared network hijacking (example: when corporate user is on a public WiFi network) of communications are exceptions.

Attackers typically gain access to internal operating systems via email-delivered network threats which first compromise a set of machines, then install attacker controlled malware, and so provide ability for the attacker to move laterally. This increases the likelihood of not being detected up front while providing an almost effortless entry point for the attacker.

According to a recent Microsoft security intelligence report, **more than 45% of malware requires some form of user interaction**, suggesting that user-targeted

email, designed to trick users, is a primary tactic used by attackers to establish their access.

Some threats are designed to disrupt an organisation's operations rather than silently gather information for financial gain or espionage. The most popular approach is called a Denial of Service (DoS) attack. These attacks overwhelm network resources such as web and email gateways, routers, switches, etc. and prevent user and application access, ultimately taking a service offline or severely degrading the quality of a service. These do not necessarily require active hacking, but instead rely on attackers' ability to scale traffic towards an organisation to take advantage of misconfigured and poorly protected infrastructure. This means they often make use of a network of compromised computer systems that work in tandem to overwhelm the target, known as a Distributed Denial of Service (DDoS) attack. In many cases, attackers will launch DoS and DDoS attacks while attempting active hacking or sending in malicious email threats to camouflage their real motives from the information security teams by creating distractions.

While detection, perimeter hardening, and patching processes are required to mitigate network threats from active and passive network threats, as a basic starting point organisations need to protect themselves especially from the email-delivered security threats that subsequently enable network-threats to be successful.

Cryptography

Cryptography is the science to encrypt and decrypt data that enables the users to store sensitive information or transmit it across insecure networks so that it can be read only by the intended recipient.

Data which can be read and understood without any special measures is called **plaintext**, while the method of disguising plaintext in order to hide its substance is called **encryption**.

Encrypted plaintext is known as cipher text and process of reverting the encrypted data back to plain text is known as **decryption**.

- The science of analyzing and breaking secure communication is known as cryptanalysis. The people who perform the same also known as attackers.
- Cryptography can be either strong or weak and the strength is measured by the time and resources it would require to recover the actual plaintext.
- Hence an appropriate decoding tool is required to decipher the strong encrypted messages.
- There are some cryptographic techniques available with which even a billion computers doing a billion checks a second, it is not possible to decipher the text.
- As the computing power is increasing day by day, one has to make the encryption algorithms very strong in order to protect data and critical information from the attackers.

How Encryption Works?

A cryptographic algorithm works in combination with a key (can be a word, number, or phrase) to encrypt the plaintext and the same plaintext encrypts to different cipher text with different keys.

Hence, the encrypted data is completely dependent couple of parameters such as the strength of the cryptographic algorithm and the secrecy of the key.

Cryptography Techniques

Symmetric Encryption − Conventional cryptography, also known as conventional encryption, is the technique in which only one key is used for both encryption and decryption. For example, DES, Triple DES algorithms, MARS by IBM, RC2, RC4, RC5, RC6.

Asymmetric Encryption − It is Public key cryptography that uses a pair of keys for encryption: a public key to encrypt data and a private key for decryption. Public key is published to the people while keeping the private key secret. For example, RSA, Digital Signature Algorithm (DSA), Elgamal.

Hashing − Hashing is ONE-WAY encryption, which creates a scrambled output that cannot be reversed or at least cannot be reversed easily. For example, MD5 algorithm. It is used to create Digital Certificates, Digital signatures, Storage of passwords, Verification of communications, etc.

Authentication

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

- **Username / Password** − User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** − User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** − User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways.

- **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.

- **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.
- **Network password** − Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

Program Threats

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as **Program Threats**. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

- **Trojan Horse** − Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.
- **Trap Door** − If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
- **Logic Bomb** − Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.
- **Virus** − Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generatlly a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

Computer Security Classifications

As per the U.S. Department of Defense Trusted Computer System's Evaluation Criteria there are four security classifications in computer systems: A, B, C, and D. This is widely used specifications to determine and model the security of systems and of security solutions. Following is the brief description of each classification.

S.N.	Classification Type & Description
1	**Type A** Highest Level. Uses formal design specifications and verification techniques. Grants a high degree of assurance of process security.
2	**Type B** Provides mandatory protection system. Have all the properties of a class C2 system. Attaches a sensitivity label to each object. It is of three types. • **B1** − Maintains the security label of each object in the system. Label is used for making decisions to access control. • **B2** − Extends the sensitivity labels to each system resource, such as storage objects, supports covert channels and auditing of events. • **B3** − Allows creating lists or user groups for access-control to grant access or revoke access to a given named object.

3	**Type C** Provides protection and user accountability using audit capabilities. It is of two types. • **C1** – Incorporates controls so that users can protect their private information and keep other users from accidentally reading / deleting their data. UNIX versions are mostly Cl class. • **C2** – Adds an individual-level access control to the capabilities of a Cl level system.
4	**Type D** Lowest level. Minimum protection. MS-DOS, Window 3.1 fall in this category.

Virtual Machine

A **Virtual Machine** (VM) is a compute resource that uses software instead of a physical computer to run programs and deploy apps. One or more virtual "guest" machines run on a physical "host" machine. Each virtual machine runs its own operating system and functions separately from the other VMs, even when they are all running on the same host. This means that, for example, a virtual MacOS virtual machine can run on a physical PC.

Virtual machine technology is used for many use cases across on-premises and cloud environments. More recently, public cloud services are using virtual machines to provide virtual application resources to multiple users at once, for even more cost efficient and flexible compute.

What are virtual machines used for?

Virtual machines (VMs) allow a business to run an operating system that behaves like a completely separate computer in an app window on a desktop. VMs may

be deployed to accommodate different levels of processing power needs, to run software that requires a different operating system, or to test applications in a safe, sandboxed environment.

Virtual machines have historically been used for server virtualization, which enables IT teams to consolidate their computing resources and improve efficiency. Additionally, virtual machines can perform specific tasks considered too risky to carry out in a host environment, such as accessing virus-infected data or testing operating systems. Since the virtual machine is separated from the rest of the system, the software inside the virtual machine cannot tamper with the host computer.

How do virtual machines work?

The virtual machine runs as a process in an application window, similar to any other application, on the operating system of the physical machine. Key files that make up a virtual machine include a log file, NVRAM setting file, virtual disk file and configuration file.

Advantages of virtual machines

Virtual machines are easy to manage and maintain, and they offer several advantages over physical machines:

- VMs can run multiple operating system environments on a single physical computer, saving physical space, time and management costs.
- Virtual machines support legacy applications, reducing the cost of migrating to a new operating system. For example, a Linux virtual machine running a distribution of Linux as the guest operating system can exist on a host server that is running a non-Linux operating system, such as Windows.
- VMs can also provide integrated disaster recovery and application provisioning options.

Disadvantages of virtual machines

While virtual machines have several advantages over physical machines, there are also some potential disadvantages:

- Running multiple virtual machines on one physical machine can result in unstable performance if infrastructure requirements are not met.
- Virtual machines are less efficient and run slower than a full physical computer. Most enterprises use a combination of physical and virtual infrastructure to balance the corresponding advantages and disadvantages.

The two types of virtual machines

Users can choose from two different types of virtual machines—process VMs and system VMs:

A process virtual machine allows a single process to run as an application on a host machine, providing a platform-independent programming environment by masking the information of the underlying hardware or operating system. An example of a process VM is the Java Virtual Machine, which enables any operating system to run Java applications as if they were native to that system.

A system virtual machine is fully virtualized to substitute for a physical machine. A system platform supports the sharing of a host computer's physical resources between multiple virtual machines, each running its own copy of the operating system. This virtualization process relies on a hypervisor, which can run on bare hardware, such as VMware ESXi, or on top of an operating system.

What are 5 types of virtualization?

All the components of a traditional data center or IT infrastructure can be virtualized today, with various specific types of virtualization:

- **Hardware virtualization:** When virtualizing hardware, virtual versions of computers and operating systems (VMs) are created and consolidated into a single, primary, physical server. A hypervisor communicates directly with

a physical server's disk space and CPU to manage the VMs. Hardware virtualization, which is also known as server virtualization, allows hardware resources to be utilized more efficiently and for one machine to simultaneously run different operating systems.

- **Software virtualization:** Software virtualization creates a computer system complete with hardware that allows one or more guest operating systems to run on a physical host machine. For example, Android OS can run on a host machine that is natively using a Microsoft Windows OS, utilizing the same hardware as the host machine does. Additionally, applications can be virtualized and delivered from a server to an end user's device, such as a laptop or smartphone. This allows employees to access centrally hosted applications when working remotely.
- **Storage virtualization:** Storage can be virtualized by consolidating multiple physical storage devices to appear as a single storage device. Benefits include increased performance and speed, load balancing and reduced costs. Storage virtualization also helps with disaster recovery planning, as virtual storage data can be duplicated and quickly transferred to another location, reducing downtime.
- **Network virtualization:** Multiple sub-networks can be created on the same physical network by combining equipment into a single, software-based virtual network resource. Network virtualization also divides available bandwidth into multiple, independent channels, each of which can be assigned to servers and devices in real time. Advantages include increased reliability, network speed, security and better monitoring of data usage. Network virtualization can be a good choice for companies with a high volume of users who need access at all times.
- **Desktop virtualization:** This common type of virtualization separates the desktop environment from the physical device and stores a desktop on a remote server, allowing users to access their desktops from anywhere on

any device. In addition to easy accessibility, benefits of virtual desktops include better data security, cost savings on software licenses and updates, and ease of management.

Container vs virtual machine

Like virtual machines, container technology such as Kubernetes is similar in the sense of running isolated applications on a single platform. While virtual machines virtualize the hardware layer to create a "computer," containers package up just a single app along with its dependencies. Virtual machines are often managed by a hypervisor, whereas container systems provide shared operating system services from the underlying host and isolate the applications using virtual-memory hardware.

A key benefit of containers is that they have less overhead compared to virtual machines. Containers include only the binaries, libraries and other required dependencies, and the application. Containers that are on the same host share the same operating system kernel, making containers much smaller than virtual machines. As a result, containers boot faster, maximize server resources, and make delivering applications easier. Containers have become popluar for use cases such as web applications, DevOps testing, microservices and maximizing the number of apps that can be deployed per server.

Virtual machines are larger and slower to boot than containers. They are logically isolated from one another, with their own operating system kernel, and offer the benefits of a completely separate operating system. Virtual machines are best for running multiple applications together, monolithic applications, isolation between apps, and for legacy apps running on older operating systems. Containers and virtual machines may also be used together.

Virtualization

With the help of OS virtualization nothing is pre-installed or permanently loaded on the local device and no-hard disk is needed. Everything runs from the network using a kind of virtual disk. This virtual disk is actually a disk image file stored on a remote server, SAN (Storage Area Network) or NAS (Non-volatile Attached Storage). The client will be connected by the network to this virtual disk and will boot with the Operating System installed on the virtual disk.

How does OS Virtualization works?

Components needed for using OS Virtualization in the infrastructure are given below:

The first component is the OS Virtualization server. This server is the center point in the OS Virtualization infrastructure. The server manages the streaming of the information on the virtual disks for the client and also determines which client will be connected to which virtual disk (using a database, this information is stored). Also the server can host the storage for the virtual disk locally or the server is connected to the virtual disks via a SAN (Storage Area Network). In high availability environments there can be more OS Virtualization servers to create no redundancy and load balancing. The server also ensures that the client will be unique within the infrastructure.

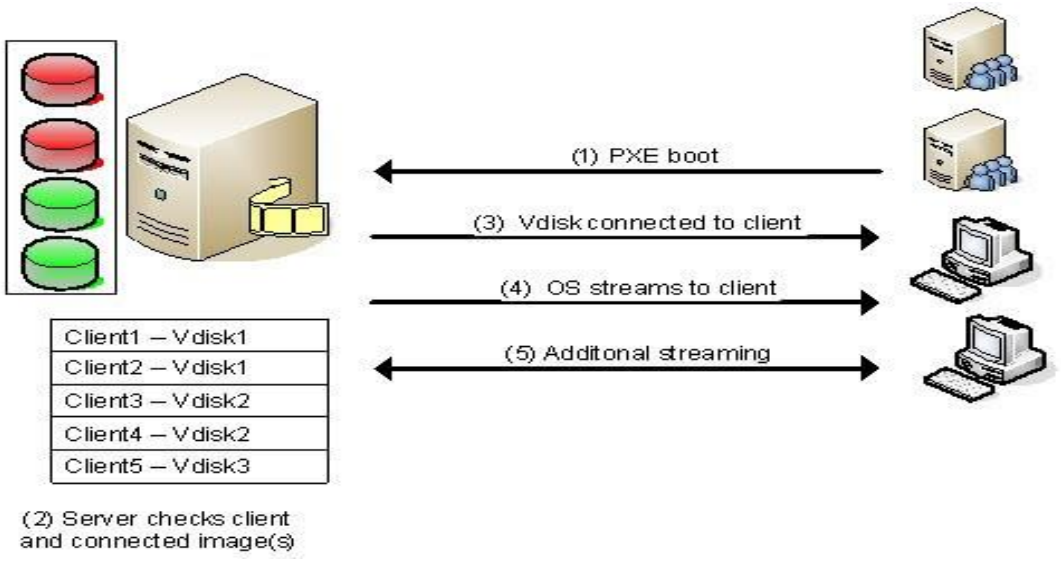
Secondly, there is a client which will contact the server to get connected to the virtual disk and asks for components stored on the virtual disk for running the operating system.

The available supporting components are database for storing the configuration and settings for the server, a streaming service for the virtual disk content, a (optional) TFTP service and a (also optional) PXE boot service for connecting the client to the OS Virtualization servers.

As it is already mentioned that the virtual disk contains an image of a physical disk from the system that will reflect to the configuration and the settings of those systems which will be using the virtual disk. When the virtual disk is created then that disk needs to be assigned to the client that will be using this disk for starting.

The connection between the client and the disk is made through the administrative tool and saved within the database. When a client has a assigned disk, the machine can be started with the virtual disk using the following process

Figure: as displayed in the given below



1) Connecting to the OS Virtualization server:

First we start the machine and set up the connection with the OS Virtualization server. Most of the products offer several possible methods to connect with the server. One of the most popular and used methods is using a PXE service, but also a boot strap is used a lot (because of the disadvantages of the PXE service). Although each method initializes the network interface card (NIC), receiving a (DHCP-based) IP address and a connection to the server.

2) Connecting the Virtual Disk:

When the connection is established between the client and the server, the server will look into its database for checking the client is known or unknown and which virtual disk is assigned to the client. When more than one virtual disk are connected then a boot menu will be displayed on the client side. If only one disk is assigned, that disk will be connected to the client which is mentioned in step number 3.

3) VDisk connected to the client:

After the desired virtual disk is selected by the client, that virtual disk is connected through the OS Virtualization server . At the back-end, the OS Virtualization server makes sure that the client will be unique (for example computer name and identifier) within the infrastructure.

4) OS is "streamed" to the client:

As soon the disk is connected the server starts streaming the content of the virtual disk. The software knows which parts are necessary for starting the operating system smoothly, so that these parts are streamed first. The information streamed in the system should be stored somewhere (i.e. cached). Most products offer several ways to cache that information. For examples on the client hard disk or on the disk of the OS Virtualization server.

5) Additional Streaming:

After that the first part is streamed then the operating system will start to run as expected. Additional virtual disk data will be streamed when required for running or starting a function called by the user (for example starting an application available within the virtual disk).

Linux

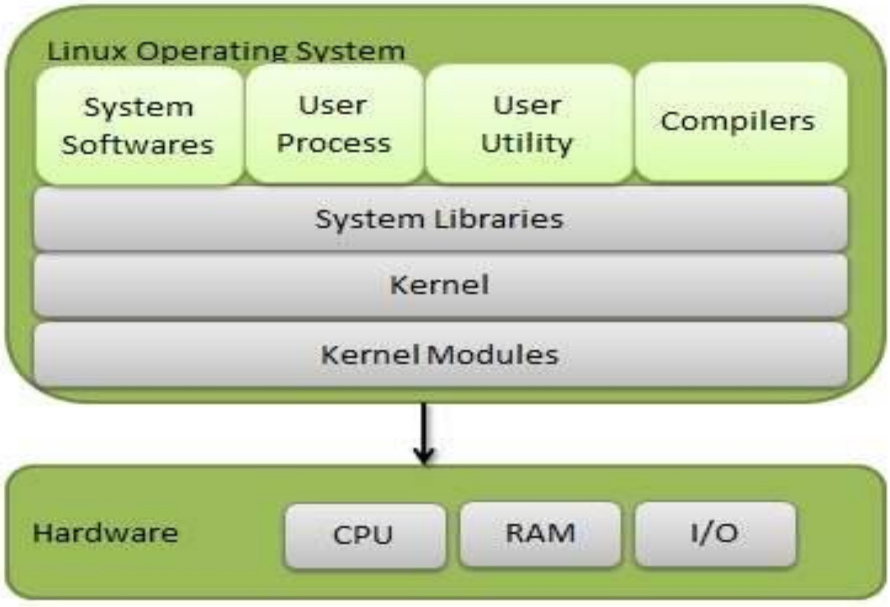
Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux System

Linux Operating System has primarily three components

- **Kernel** − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.

- **System Library** – System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.
- **System Utility** – System Utility programs are responsible to do specialized, individual level tasks.



Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called **kernel mode** with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in **User Mode** which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

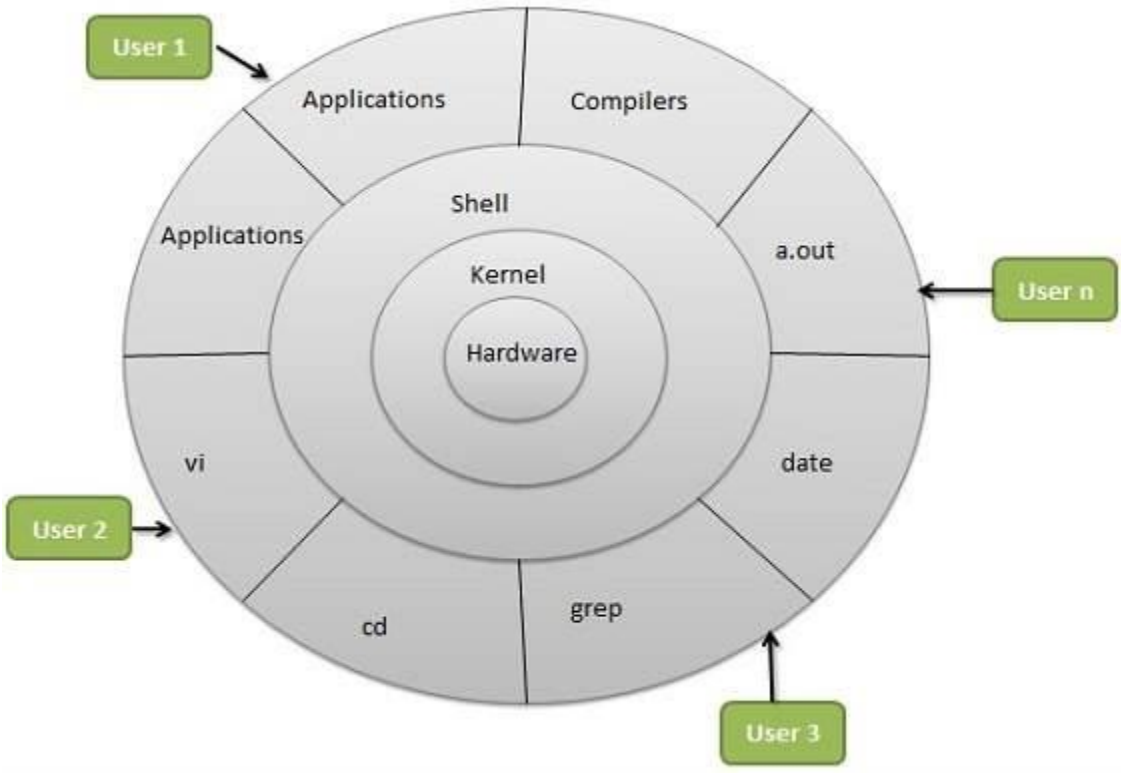
Basic Features

Following are some of the important features of Linux Operating System.

- **Portable** − Portability means software can works on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** − Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** − Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** − Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Architecture

The following illustration shows the architecture of a Linux system −



The architecture of a Linux System consists of the following layers −

- **Hardware layer** − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** − It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** − An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.
- **Utilities** − Utility programs that provide the user most of the functionalities of an operating systems.

Linux Design Principles

In its overall design, Linux resembles another **non-microkernel UNIX** implementation. It is a multi-user, multi-tasking system with complete UNIX-compatible tools. The Linux file system follows the traditional UNIX

semantics, and the UNIX standard network model is implemented as a whole. The internal characteristics of Linux design have been influenced by the history of the development of this operating system.

Although Linux can run on a variety of platforms, at first it was developed exclusively on PC architecture. Most of the initial development was carried out by individual enthusiasts, not by large funded research facilities, so that from the start Linux tried to include as much functionality as possible with very limited funds. Currently, Linux can run well on multi-processor machines with very large main memory and disk space that is also very large, but still capable of operating in a useful amount of RAM smaller than 4 MB.

1. Linux Design Principles

As a result of the development of PC technology, the Linux kernel is also becoming more complete in implementing UNIX functions. Fast and efficient are important design goals, but lately the concentration of Linux development has focused more on the third design goal, standardization. The POSIX standard consists of a collection of specifications from different aspects of operating system behavior. There are POSIX documents for ordinary operating system functions and for extensions such as processes for threads and real-time operations. Linux is designed to fit the relevant POSIX documents; there are at least two Linux distributions that have received POSIX official certification.

Because Linux provides a standard interface to programmers and users, Linux does not make many surprises to anyone who is familiar with UNIX. But the Linux programming interface refers to the UNIX SVR4 semantics rather than BSD behavior. A different collection of libraries is available to implement the BSD semantics in places where the two behaviors are very different.

There are many other standards in the UNIX world, but Linux's full certification of other UNIX standards sometimes becomes slow because it is more often available at a certain price (not freely), and there is a price to pay if it involves certification of approval or compatibility of an operating system with most standards . Supporting broad applications is important for all operating systems so that the

implementation of the standard is the main goal of developing Linux even though its implementation is not formally valid. In addition to the POSIX standard, Linux currently supports POSIX thread extensions and subsets of extensions for POSIX real-time process control.

2. Linux System Components

The Linux system consists of three important code parts:

1. Kernel: Responsible for maintaining all important abstractions of the operating system, including things like processes and virtual memory.
2. System library: defines a set of standard functions where applications can interact with the kernel, and implements almost all operating system functions that do not require full rights to the kernel.
3. System Utility: is a program that performs management work individually and specifically.

2.1. Kernel

Although various modern operating systems have adopted a message-passing architecture for their internal kernel, Linux uses the historical UNIX model: the kernel was created as a single, monolithic binary. The main reason is to improve performance: Because all data structures and kernel code are stored in one address space, context switching is not needed when a process calls an operating system function or when a hardware interrupt is sent. Not only scheduling core and virtual memory code occupies this address space; all kernel code, including all device drivers, file systems, and network code, come in the same address space.

The Linux kernel forms the core of the Linux operating system. It provides all the functions needed to run the process, and is provided with system services to provide settings and protection for access to hardware resources. The kernel implements all the features needed to work as an operating system. However, if alone, the operating system provided by the Linux kernel is not at all similar to UNIX systems. It does not have many extra UNIX features, and the features

provided are not always in the format expected by the UNIX application. The interface of the operating system that is visible to the running application is not maintained directly by the kernel. Instead, the application makes calls to the system library, which then invokes the operating system services that are needed.

2.2. System Library

The system library provides many types of functions. At the easiest level, they allow applications to make requests to the kernel system services. Making a system call involves transferring controls from non-essential user mode to important kernel mode; the details of this transfer are different for each architecture. The library has the duty to collect system-call arguments and, if necessary, arrange those arguments in the special form needed to make system calls.

Libraries can also provide more complex versions of basic system calls. For example, the buffered file-handling functions of the C language are all implemented in the system library, which results in better control of file I / O than those provided by the basic kernel system call. The library also provides routines that have nothing to do with system calls, such as sorting algorithms, mathematical functions, and string manipulation routines. All functions needed to support the running of UNIX or POSIX applications are implemented in the system library.

2.3. System Utilities

Linux systems contain many user-mode programs: system utilities and user utilities. The system utilities include all the programs needed to initialize the system, such as programs for configuring network devices or for loading kernel modules. Server programs that are running continuously are also included as system utilities; This kind of program manages user login requests, incoming network connections, and printer queues.

Not all standard utilities perform important system administration functions. The UNIX user environment contains a large number of standard utilities for doing

daily work, such as making directory listings, moving and deleting files, or showing the contents of a file. More complex utilities can perform text-processing functions, such as compiling textual data or performing pattern-searches on text input. When combined, these utilities form the standard toolset expected by users on any UNIX system; even if it doesn't perform any operating system functions, utilities are still an important part of a basic Linux system.

Linux Process Management

In this article we will cover the basics of process management in Linux. This topic is of particular importance if you are responsible for administering a system which has not yet been proven stable, that is not fully tested in its configuration. You may find that as you run software, problems arise requiring administrator intervention. This is the world of process management

Process Management

Any application that runs on a Linux system is assigned a process ID or PID. This is a numerical representation of the instance of the application on the system. In most situations this information is only relevant to the system administrator who may have to debug or terminate processes by referencing the PID. Process Management is the series of tasks a System Administrator completes to monitor, manage, and maintain instances of running applications.

Multitasking

Process Management beings with an understanding concept of Multitasking. Linux is what is referred to as a preemptive multitasking operating system. Preemptive multitasking systems rely on a scheduler. The function of the scheduler is to control the process that is currently using the CPU. In contrast, symmetric multitasking systems such as Windows 3.1 relied on each running process to voluntary relinquish control of the processor. If an application in this system hung or stalled, the entire computer system stalled. By making use of an additional component to pre-empt each process when its "turn" is up, stalled programs do not affect the overall flow of the operating system.

Each “turn” is called a time slice, and each time slice is only a fraction of a second long. It is this rapid switching from process to process that allows a computer to “appear’ to be doing two things at once, in much the same way a movie “appears” to be a continuous picture.

Types of Processes

There are generally two types of processes that run on Linux. Interactive processes are those processes that are invoked by a user and can interact with the user. VI is an example of an interactive process. Interactive processes can be classified into foreground and background processes. The foreground process is the process that you are currently interacting with, and is using the terminal as its stdin (standard input) and stdout (standard output). A background process is not interacting with the user and can be in one of two states – paused or running.

The following exercise will illustrate foreground and background processes.
1. Logon as root.
2. Run [cd \]
3. Run [vi]
4. Press [ctrl + z]. This will pause vi
5. Type [jobs]
6. Notice vi is running in the background
7. Type [fg %1]. This will bring the first background process to the foreground.
8. Close vi.

The second general type of process that runs on Linux is a system process or Daemon (day-mon). Daemon is the term used to refer to process’ that are running on the computer and provide services but do not interact with the console. Most server software is implemented as a daemon. Apache, Samba, and inn are all examples of daemons.

Any process can become a daemon as long as it is run in the background, and does not interact with the user. A simple example of this can be achieved using the [ls –R] command. This will list all subdirectories on the computer, and is similar to the [dir /s] command on Windows. This command can be set to run in

the background by typing [ls –R &], and although technically you have control over the shell prompt, you will be able to do little work as the screen displays the output of the process that you have running in the background. You will also notice that the standard pause (ctrl+z) and kill (ctrl+c) commands do little to help you.

Linux Scheduling

The Linux scheduler is a priority based scheduler that schedules tasks based upon their static and dynamic priorities. When these priorities are combined they form a task's goodness . Each time the Linux scheduler runs, every task on the run queue is examined and its goodness value is computed. The task with the highest goodness is chosen to run next.

When there are cpu bound tasks running in the system, the Linux scheduler may not be called for intervals of up to .40 seconds. This means that the currently running task has the CPU to itself for periods of up to .40 seconds (how long depends upon the task's priority and whether it blocks or not). This is good for throughput because there are few computationally uneccessary context switches. However it can kill interactivity because Linux only reschedules when a task blocks or when the task's dynamic priority (counter) reaches zero. Thus under Linux's default priority based scheduling method, long scheduling latencies can occur.

Looking at the scheduling latency in finer detail, the Linux scheduler makes use of a timer that interrupts every 10 msec. This timer erodes the currently running task's dynamic priority (decrements its counter). A task's counter starts out at the same value its priority contains. Once its dynamic priority (counter) has eroded to 0 it is again reset to that of its static priority (priority). It is only after the counter reaches 0 that a call to schedule() is made. Thus a task with the default priority of 20 may run for .200 secs (200 msecs) before any other task in the system gets a chance to run. A task at priority 40 (the highest priority allowed) can run for .400 secs without any scheduling occurring as long as it doesn't block or yield.

Linux scheduler has been gone through some big improvements since kernel version 2.4. There were a lot of complaints about the interactivity of the

scheduler in kernel 2.4. During this version, the scheduler was implemented with one running queue for all available processors. At every scheduling, this queue was locked and every task on this queue got its timeslice update. This implementation caused poor performance in all aspects. The scheduler algorithm and supporting code went through a large rewrite early in the 2.5 kernel development series. The new scheduler was arisen to achieveO(1) run-time regardless number of runnable tasks in the system. To achieve this, each processor has its own running queue. This helps a lot in reducing lock contention. The priority array was introduced which used active array and expired array to keep track running tasks in the system. TheO(1) running time is primarily drawn from this new data structure. The scheduler puts all expired processes into expired array. When there is no active process available in active array, it swaps active array with expired array, which makes active array becomes expired array and expired array becomes active array. There were some twists made into this scheduler to optimize further by putting expired task back to active array instead of expired array in some cases.O(1) scheduler uses a heuristic calculation to update dynamic priority of tasks based on their interactivity (I/O bound versus CPU bound) The industry was happy with this new scheduler until Con Kolivas introduced his new scheduler named Rotating Staircase Deadline (RSDL) and then later Staircase Deadline (SD). His new schedulers proved the fact that fair scheduling among processes can be achieved without any complex computation. His scheduler was designed to run inO(n) but its performance exceeded the currentO(1) scheduler.

The result achieved from SD scheduler surprised all kernel developers and designers. The fair scheduling approach in SD scheduler encouraged Igno Molnar to re-implement the new Linux scheduler named Completely Fair Scheduler (CFS). CFS scheduler was a big improvement over the existing scheduler not only in its performance and interactivity but also in simplifying the scheduling logic and putting more modularized code into the scheduler. CFS scheduler was merged into mainline version 2.6.23. Since then, there have been some minor improvements made to CFS scheduler in some areas such as optimization, load balancing and group scheduling feature.

Kernel 2.4 Major Features

- An O(n) scheduler - Goes through the entire " global runqueue" to determine the next task to be run. This is an O(n) algorithm where 'n' is the number of processes. The time taken was proportional to the number of active processes in the system
- A Global runqueue - All CPUs had to wait for other CPUs to finish execution. A Global runqueue for all processors in a symmetric multiprocessing system (SMP). This meant a task could be scheduled on any processor -- which can be good for load balancing but bad for memory caches. For example, suppose a task executed on CPU-1, and its data was in that processor's cache. If the task got rescheduled to CPU-2, its data would need to be invalidated in CPU-1 and brought into CPU-2 .
- This lead to large performance hits during heavy workload

Kernel 2.4 Scheduler Policies:

- SCHED_FIFO - A First-In, First-Out real-time process. When the scheduler assigns the CPU to the process, it leaves the process descriptor in its current position in the runqueue list. If no other higher-priority realtime process is runnable, the process will continue to use the CPU as long as it wishes, even if other real-time processes having the same priority are runnable
- SCHED_RR - A Round Robin real-time process. When the scheduler assigns the CPU to the process, it puts the process descriptor at the end of the runqueue list. This policy ensures a fair assignment of CPU time to all SCHED_RR real-time processes that have the same priority
- SCHED_OTHER - A conventional, time-shared process. The policy field also encodes a SCHED_YIELD binary flag. This flag is set when the process invokes the sched_ yield( ) system call (a way of voluntarily relinquishing the processor without the need to start an I/O operation or go to sleep. The scheduler puts the process descriptor at the bottom of the runqueue list.

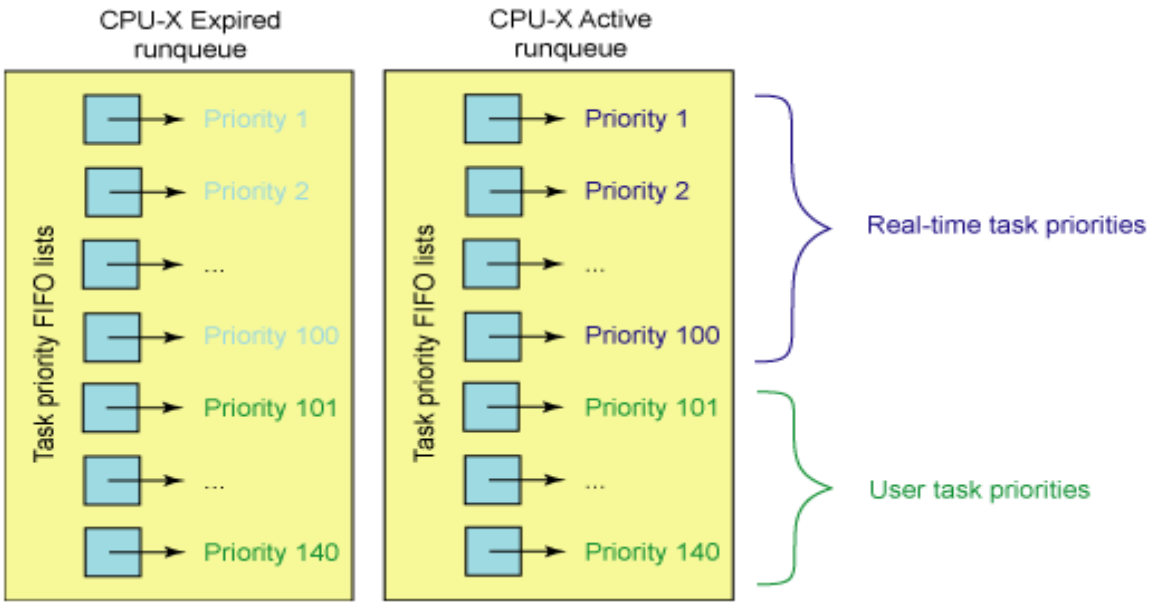
O(1) Algorithm (Constant time algorithm)

- Choose the task on the highest priority list to execute
- To make this process more efficient, a bitmap is used to define when tasks are on a given priority list
- On most architectures, a find-first-bit-set instruction is used to find the highest priority bit set in one of five 32-bit words (for the 140 priorities
- The time it takes to find a task to execute depends not on the number of active tasks but instead on the number of priorities
- This makes the 2.6 scheduler an O(1) process because the time to schedule is both fixed and deterministic regardless of the number of active tasks

Kernel 2.6 - Major Features

- The 2.6 scheduler was designed and implemented by Ingo Molnar. His motivation in working on the new scheduler was to create a completely O(1) scheduler for wakeup, context-switch, and timer interrupt overhead
- One of the issues that triggered the need for a new scheduler was the use of Java virtual machines (JVMs). The Java programming model uses many threads of execution, which results in lots of overhead for scheduling in an O(n) scheduler
- Each CPU has a runqueue made up of 140 priority lists that are serviced in FIFO order. Tasks that are scheduled to execute are added to the end of their respective runqueue's priority list
- Each task has a time slice that determines how much time it's permitted to execute
- The first 100 priority lists of the runqueue are reserved for real-time tasks, and the last 40 are used for user tasks (MAX_RT_PRIO=100 and MAX_PRIO=140)

- In addition to the CPU's runqueue, which is called the active runqueue, there's also an expired runqueue
- When a task on the active runqueue uses all of its time slice, it's moved to the expired runqueue. During the move, its time slice is recalculated (and so is its priority)
- If no tasks exist on the active runqueue for a given priority, the pointers for the active and expired runqueues are swapped, thus making the expired priority list the active one



Kernel 2.6 Scheduler Policies:

- SCHED_NORMAL - A conventional, time-shared process (used to be called SCHED_OTHER), for normal tasks

1. Each task assigned a “Nice” value
2. PRIO = MAX_RT_PRIO + NICE + 20
3. Assigned a time slice
4. Tasks at the same prio(rity) are round-robined
5. Ensures Priority + Fairness

- SCHED_FIFO - A First-In, First-Out real-time process

1. Run until they relinquish the CPU voluntarily
2. Priority levels maintained
3. Not pre-empted !!

- SCHED_RR - A Round Robin real-time process

1. Assigned a timeslice and run till the timeslice is exhausted.
2. Once all RR tasks of a given prio(rity) level exhaust their timeslices, their timeslices are refilled and they continue running
3. Prio(rity) levels are maintained

- SCHED_BATCH - for "batch" style execution of processes

1. For computing-intensive tasks
2. Timeslices are long and processes are round robin scheduled
3. lowest priority tasks are batch-processed (nice +19)

- SCHED_IDLE - for running very low priority background job

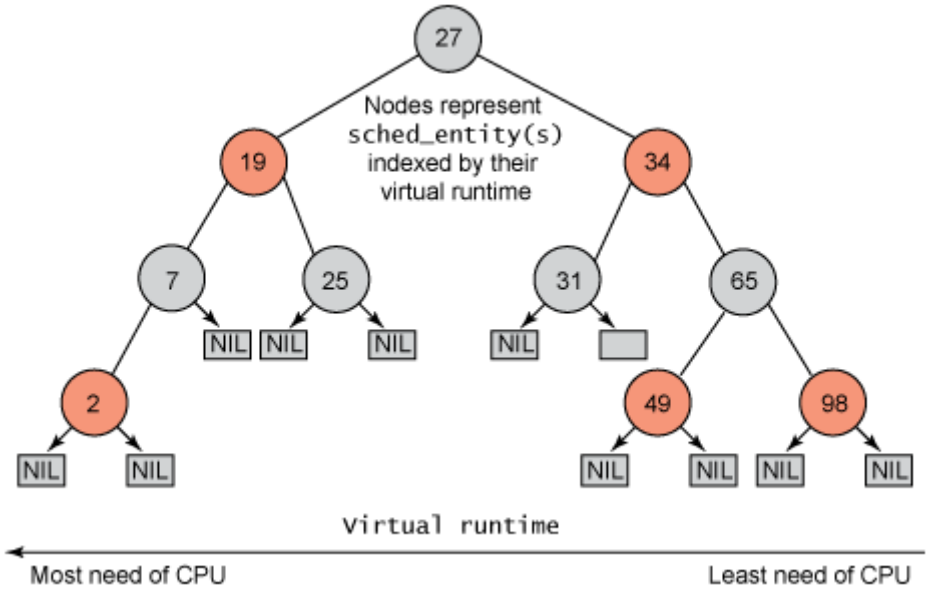
1. nice value has no influence for this policy
2. extremely low priority (lower than +19 nice)

Completely Fair Scheduler (CFS)

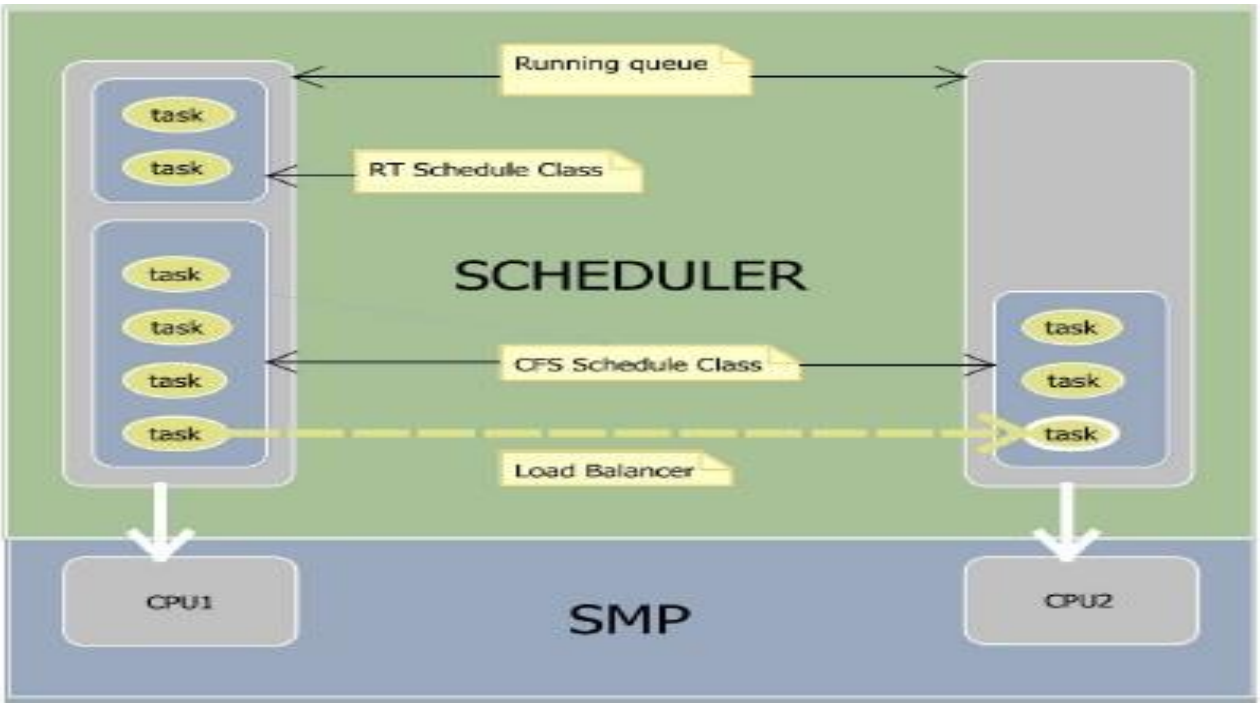
- The main idea behind the CFS is to maintain balance (fairness) in providing processor time to tasks. This means processes should be given a fair amount of the processor. When the time for tasks is out of balance (meaning that one or more tasks are not given a fair amount of time relative to others), then those out-of-balance tasks should be given time to execute.

To determine the balance, the CFS maintains the amount of time provided to a given task in what's called the virtual runtime. The smaller a task's virtual runtime—meaning the smaller amount of time a task has been permitted access to the processor—the higher its need for the processor. The CFS also includes the concept of sleeper fairness to ensure that tasks that are not currently runnable (for example, waiting for I/O) receive a comparable share of the processor when they eventually need it.

But rather than maintain the tasks in a run queue, as has been done in prior Linux schedulers, the CFS maintains a time-ordered red-black tree (see Figure below). A red-black tree is a tree with a couple of interesting and useful properties. First, it's self-balancing, which means that no path in the tree will ever be more than twice as long as any other. Second, operations on the tree occur in O(log n) time (where n is the number of nodes in the tree). This means that you can insert or delete a task quickly and efficiently.



Concrete view of Linux Kernel Scheduler



Linux scheduler contains:

- **A Running Queue :** A running queue (rq) is created for each processor (CPU). It is defined in kernel/sched.c as struct _runqueue. Each rq contains a list of runnable processes on a given processor. Thestruct_runqueue is defined in sched.c notsched.h to abstract the internal data structure of the scheduler.

- **Schedule Class :** schedule class was introduced in 2.6.23. It is an extensible hierarchy of scheduler modules. These modules encapsulate scheduling policy details and are called from the scheduler core without the core code assuming too much about them. Scheduling classes are implemented through thesched_class structure, which contains hooks to functions that must be called whenever an interesting event occurs. Tasks refer to their schedule policy through struct task_struct and sched_class. There are two schedule classes implemented in 2.6.32:

1. **Completely Fair Schedule class:** schedules tasks following Completely Fair Scheduler (CFS) algorithm. Tasks which have policy set to SCHED_ NORMA L

(SCHED_OTHER), SCHED_BATCH, SCHED_IDLE are scheduled by this schedule class. The implementation of this class is in kernel /sched_fai r.c

2. **RT schedule class:** schedules tasks following real-time mechanism defined in POSIX standard. Tasks which have policy set to SCHED_FIFO, SCHED_RR are scheduled using this schedule class. The implementation of this class is kernel/sched_rt.c

- **Load balancer:** In SMP environment, each CPU has its own rq. These queues might be unbalanced from time to time. A running queue with empty task pushes its associated CPU to idle, which does not take full advantage of symmetric multiprocessor systems. Load balancer is to address this issue. It is called every time the system requires scheduling tasks. If running queues are unbalanced, load balancer will try to pull idle tasks from busiest processors to idle processor.

Interactivity

Interactivity is an important goal for the Linux scheduler, especially given the growing effort to optimize Linux for desktop environments. Interactivity often flies in the face of efficiency, but it is very important nonetheless. An example of interactivity might be a keystroke or mouse click. Such events usually require a quick response (i.e. the thread handling them should be allowed to execute very soon) because users will probably notice and be annoyed if they do not see some result from their action almost immediately. Users don’t expect a quick response when, for example, they are compiling programs or rendering high-resolution images. They are unlikely to notice if something like compiling the Linux kernel takes an extra twenty seconds. Schedulers used for interactive computing should be designed in such a way that they respond to user interaction within a certain time period. Ideally, this should be a time period that is imperceptible to users and thus gives the impression of an immediate response.

Interactivity estimator

- Dynamically scales a tasks priority based on it's interactivity

- Interactive tasks receive a prio bonus
- Hence a larger timeslice
- CPU bound tasks receive a prio penalty
- Interactivity estimated using a running sleep average.
- Interactive tasks are I/O bound. They wait for events to occur.
- Sleeping tasks are I/O bound or interactive !!
- Actual bonus/penalty is determined by comparing the sleep average against a constant maximum sleep average.
- Does not apply to RT tasks

When a task finishes it's timeslice :

- It's interactivity is estimated
- Interactive tasks can be inserted into the 'Active' array again
- Else, priority is recalculated
- Inserted into the NEW priority level in the 'Expired' array

Re-inserting interactive tasks

- To avoid delays, interactive tasks may be re-inserted into the 'active' array after their timeslice has expired
- Done only if tasks in the 'expired' array have run recently
- Done to prevent starvation of tasks
- Decision to re-insert depends on the task's priority level

Timeslice distribution:

- Priority is recalculated only after expiring a timeslice

- Interactive tasks may become non-interactive during their LARGE timeslices, thus starving other processes
- To prevent this, time-slices are divided into chunks of 20ms
- A task of equal priority may preempt the running task every 20ms
- The preempted task is requeued and is round-robined in it's priority level.
- Also, priority recalculation happens every 20ms

Memory Management

Linux memory management subsystem is responsible, as the name implies, for managing the memory in the system. This includes implemnetation of virtual memory and demand paging, memory allocation both for kernel internal structures and user space programms, mapping of files into processes address space and many other cool things.

The memory management in Linux is a complex system that evolved over the years and included more and more functionality to support a variety of systems from MMU-less microcontrollers to supercomputers. The memory management for systems without an MMU is called nommu and it definitely deserves a dedicated document, which hopefully will be eventually written. Yet, although some of the concepts are the same, here we assume that an MMU is available and a CPU can translate a virtual address to a physical address.

Linux File System

A Linux file system is a structured collection of files on a disk drive or a partition. A partition is a segment of memory and contains some specific data. In our machine, there can be various partitions of the memory. Generally, every partition contains a file system.

The general-purpose computer system needs to store data systematically so that we can easily access the files in less time. It stores the data on hard disks (HDD) or

some equivalent storage type. There may be below reasons for maintaining the file system:

- Primarily the computer saves data to the RAM storage; it may lose the data if it gets turned off. However, there is non-volatile RAM (Flash RAM and SSD) that is available to maintain the data after the power interruption.
- Data storage is preferred on hard drives as compared to standard RAM as RAM costs more than disk space. The hard disks costs are dropping gradually comparatively the RAM.

The Linux file system contains the following sections:

- The root directory (/)
- A specific data storage format (EXT3, EXT4, BTRFS, XFS and so on)
- A partition or logical volume having a particular file system.

What is the Linux File System?

Linux file system is generally a built-in layer of a Linux operating system used to handle the data management of the storage. It helps to arrange the file on the disk storage. It manages the file name, file size, creation date, and much more information about a file.

If we have an unsupported file format in our file system, we can download software to deal with it.

Linux File System Structure

Linux file system has a hierarchal file structure as it contains a root directory and its subdirectories. All other directories can be accessed from the root directory. A partition usually has only one file system, but it may have more than one file system.

A file system is designed in a way so that it can manage and provide space for non-volatile storage data. All file systems required a namespace that is a naming

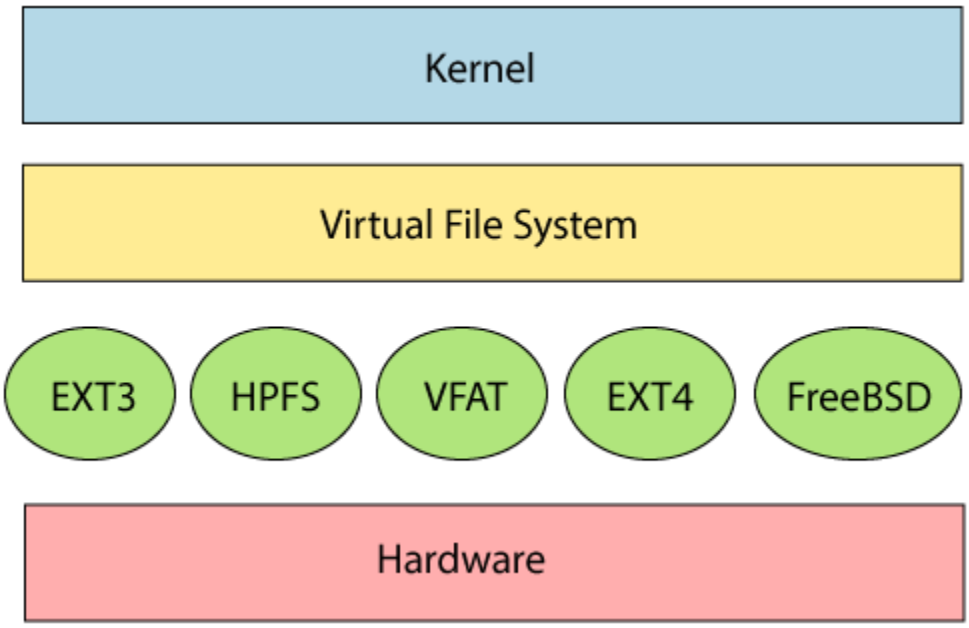
and organizational methodology. The namespace defines the naming process, length of the file name, or a subset of characters that can be used for the file name. It also defines the logical structure of files on a memory segment, such as the use of directories for organizing the specific files. Once a namespace is described, a Metadata description must be defined for that particular file.

The data structure needs to support a hierarchical directory structure; this structure is used to describe the available and used disk space for a particular block. It also has the other details about the files such as file size, date & time of creation, update, and last modified.

Also, it stores advanced information about the section of the disk, such as partitions and volumes.

The advanced data and the structures that it represents contain the information about the file system stored on the drive; it is distinct and independent of the file system metadata.

Linux file system contains two-part file system software implementation architecture. Consider the below image:



The file system requires an API (Application programming interface) to access the function calls to interact with file system components like files and directories. API facilitates tasks such as creating, deleting, and copying the files. It facilitates an algorithm that defines the arrangement of files on a file system.

The first two parts of the given file system together called a **Linux virtual file system**. It provides a single set of commands for the kernel and developers to access the file system. This virtual file system requires the specific system driver to give an interface to the file system.

Linux File System Features

In Linux, the file system creates a tree structure. All the files are arranged as a tree and its branches. The topmost directory called the **root (/) directory**. All other directories in Linux can be accessed from the root directory.

Some key features of Linux file system are as following:

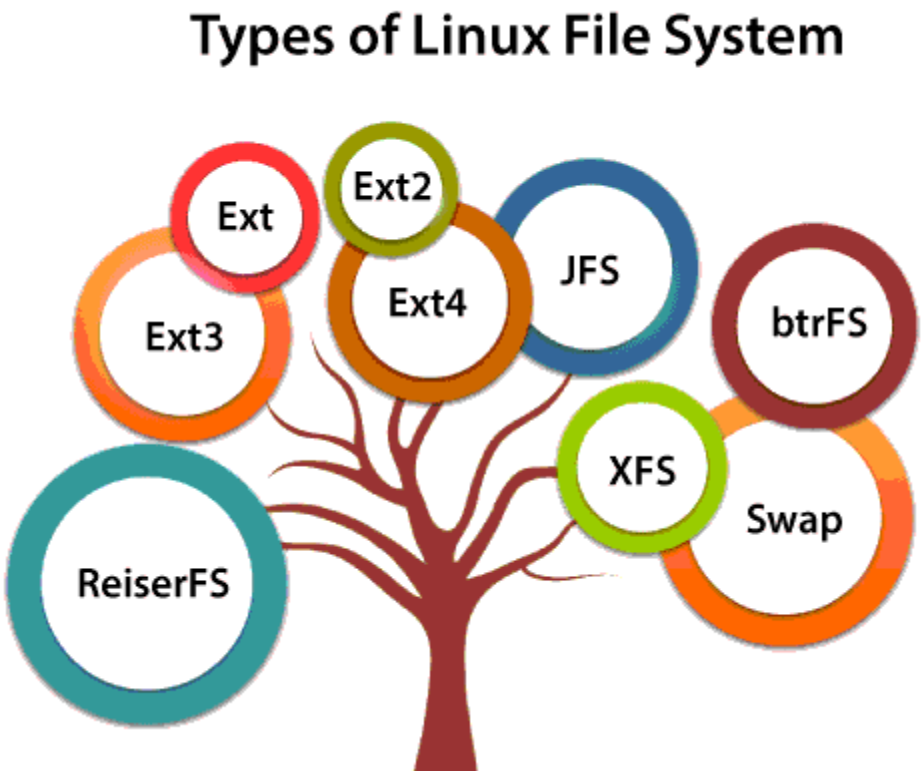
- **Specifying paths:** Linux does not use the backslash (\) to separate the components; it uses forward slash (/) as an alternative. For example, as in Windows, the data may be stored in C:\ My Documents\ Work, whereas, in Linux, it would be stored in /home/ My Document/ Work.
- **Partition, Directories, and Drives:** Linux does not use drive letters to organize the drive as Windows does. In Linux, we cannot tell whether we are addressing a partition, a network device, or an "ordinary" directory and a Drive.
- **Case Sensitivity:** Linux file system is case sensitive. It distinguishes between lowercase and uppercase file names. Such as, there is a difference between test.txt and Test.txt in Linux. This rule is also applied for directories and Linux commands.
- **File Extensions:** In Linux, a file may have the extension '.txt,' but it is not necessary that a file should have a file extension. While working with Shell, it creates some problems for the beginners to differentiate between files

and directories. If we use the graphical file manager, it symbolizes the files and folders.

- **Hidden files:** Linux distinguishes between standard files and hidden files, mostly the configuration files are hidden in Linux OS. Usually, we don't need to access or read the hidden files. The hidden files in Linux are represented by a dot (.) before the file name (e.g., .ignore). To access the files, we need to change the view in the file manager or need to use a specific command in the shell.

Types of Linux File System

When we install the Linux operating system, Linux offers many file systems such as **Ext, Ext2, Ext3, Ext4, JFS, ReiserFS, XFS, btrfs,** and **swap**.



1. Ext, Ext2, Ext3 and Ext4 file system

The file system Ext stands for **Extended File System**. It was primarily developed for **MINIX OS**. The Ext file system is an older version, and is no longer used due to some limitations.

Ext2 is the first Linux file system that allows managing two terabytes of data. Ext3 is developed through Ext2; it is an upgraded version of Ext2 and contains backward compatibility. The major drawback of Ext3 is that it does not support servers because this file system does not support file recovery and disk snapshot.

Ext4 file system is the faster file system among all the Ext file systems. It is a very compatible option for the SSD (solid-state drive) disks, and it is the default file system in Linux distribution.

2. JFS File System

JFS stands for **Journaled File System**, and it is developed by **IBM for AIX Unix**. It is an alternative to the Ext file system. It can also be used in place of Ext4, where stability is needed with few resources. It is a handy file system when CPU power is limited.

3. ReiserFS File System

ReiserFS is an alternative to the Ext3 file system. It has improved performance and advanced features. In the earlier time, the ReiserFS was used as the default file system in SUSE Linux, but later it has changed some policies, so SUSE returned to Ext3. This file system dynamically supports the file extension, but it has some drawbacks in performance.

4. XFS File System

XFS file system was considered as high-speed JFS, which is developed for parallel I/O processing. NASA still using this file system with its high storage server (300+ Terabyte server).

5. Btrfs File System

Btrfs stands for the **B tree file system**. It is used for fault tolerance, repair system, fun administration, extensive storage configuration, and more. It is not a good suit for the production system.

6. Swap File System

The swap file system is used for memory paging in Linux operating system during the system hibernation. A system that never goes in hibernate state is required to have swap space equal to its RAM size.

Linux-Input & output

Input and Output

To the user, the I/O system in Linux looks much like that in any UNIX system. That is, to the extent possible, all device drivers appear as normal files. A user can open an access channel to a device in the same way she opens any other file—devices can appear as objects within the file system.

The system administrator can create special files within a file system that contain references to a specific device driver, and a user opening such a file will be able to read from and write to the device referenced. By using the normal file-protection system, which determines who can access which file, the administrator can set access permissions for each device. Linux splits all devices into three classes: block devices, character devices, and network devices.

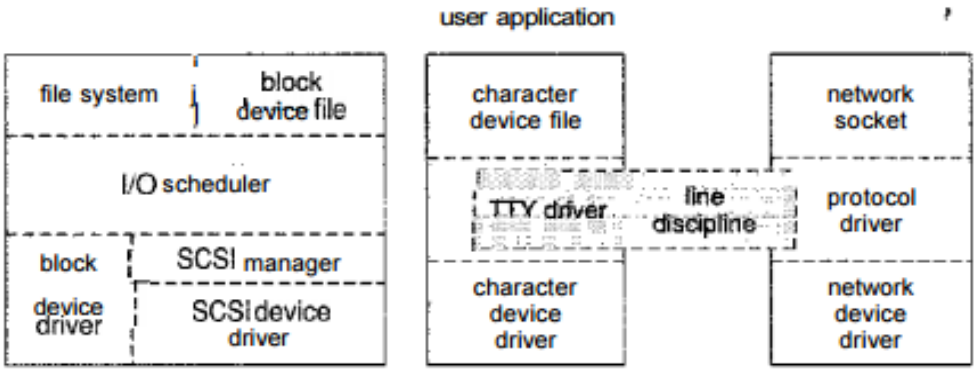


Figure 21.10 Device-driver block structure.

Figure 21.10 illustrates the overall structure of the device-driver system. Block devices include all devices that allow random access to completely independent, fixed-sized blocks of data, including hard disks and floppy disks, CD-ROMs, and flash memory. Block devices are typically used to store file systems, but direct access to a block device is also allowed so that programs can create and repair the file system that the device contains.

Applications can also access these block devices directly if they wish; for example, a database application may prefer to perform its own, fine-tuned laying out of data onto the disk, rather than using the general-purpose file system. Character devices include most other devices, such as mice and keyboards. The fundamental difference between block and character devices is random access—block devices may be accessed randomly, while character devices are only accessed serially.

For example, seeking to a certain position in a file might be supported for a DVD but makes no sense to a pointing device such as a mouse. Network devices are dealt with differently from block and character devices. Users cannot directly transfer data to network devices; instead, they must communicate indirectly by opening a connection to the kernel's networking subsystem. We discuss the interface to network devices separately in Section 21.10.

Block Devices

Block devices provide the main interface to all disk devices in a system. Performance is particularly important for disks, and the block-device system must provide functionality to ensure that disk access is as fast as possible. This functionality is achieved through the scheduling of I/O operations In the context of block devices, a block represents the unit with which the kernel performs I/O. When a block is read into memory, it is stored in a buffer. The request manager is the layer of software that manages the reading and writing of buffer contents to and from a block-device driver. A separate list of requests is kept for each block-device driver. Traditionally, these requests have been scheduled according to a unidirectional-elevator (C-SCAN) algorithm that exploits the order in which requests are inserted in and removed from the per-device lists. The request lists

are maintained in sorted order of increasing starting-sector number. When a request is accepted for processing by a block-device driver, it is not removed from the list. It is removed only after the I/O is complete, at which point the driver continues with the next request in the list, even if new requests have been inserted into the list before the active request. As new I/O requests are made, the request manager attempts to merge requests in the per-device lists. The scheduling of I/O operations changed somewhat with version 2.6 of the kernel. The fundamental problem with the elevator algorithm is that I/O operations concentrated in a specific region of the disk can result in starvation of requests that need to occur in other regions of the disk.

The deadline I/O scheduler used in version 2.6 works similarly to the elevator algorithm except that it also associates a deadline with each request, thus addressing the starvation issue. By default, the deadline for read requests is 0.5 second and that for write requests is 5 seconds. The deadline scheduler maintains a sorted queue of pending I/O operations sorted by sector number. However, it also maintains two other queues—a read queue for read operations and a write queue for write operations. These two queues are ordered according to deadline.

Every I/O request is placed in both the sorted queue and either the read or the write queue, as appropriate. Ordinarily, I/O operations occur from the sorted queue. However, if a deadline expires for a request in either the read or the write queue, I/O operations are scheduled from the queue containing the expired request. This policy ensures that an I/O operation will wait no longer than its expiration time

Character Devices

A character-device driver can be almost any device driver that does not offer random access to fixed blocks of data. Any character-device drivers registered to the Linux kernel must also register a set of functions that implement the file I/O operations that the driver can handle. The kernel performs almost no preprocessing of a file read or write request to a character device; it simply passes the request to the device in question and lets the device deal with the request.

The main exception to this rule is the special subset of character-device drivers that implement terminal devices. The kernel maintains a standard interface to these drivers by means of a set of tty_struc t structures. Each of these structures provides buffering and flow control on the data stream from the terminal device and feeds those data to a line discipline.

A line discipline is an interpreter for the information from the terminal device. The most common line discipline is the tt y discipline, which glues the terminal's data stream onto the standard input and output streams of a user's running processes, allowing those processes to communicate directly with user's terminal. This job is complicated by the fact that several such processes may be running simultaneously, and the tt y line discipline is responsible for attaching and detaching the terminal's input and output from the various processes connected to it as those processes are suspended or awakened by the user.

Other line disciplines also are implemented that have nothing to do with I/O to a user process. The PPP and SLIP networking protocols are ways of encoding a networking connection over a terminal device such as a serial line. These protocols are implemented under Linux as drivers that at one end appear to the terminal system as line disciplines and at the other end appear to the networking system as network-device drivers. After one of these line disciplines has been enabled on a terminal device, any data appearing on that terminal will be routed directly to the appropriate network-device driver.

Methods in Interprocess Communication

Inter-process communication (IPC) is set of interfaces, which is usually programmed in order for the programs to communicate between series of processes. This allows running programs concurrently in an Operating System. These are the methods in IPC:

1. **Pipes (Same Process) –**
   This allows flow of data in one direction only. Analogous to simplex systems (Keyboard). Data from the output is usually buffered until input process receives it which must have a common origin.

2. **Names Pipes (Different Processes) –**
   This is a pipe with a specific name it can be used in processes that don't have a shared common process origin. E.g. is FIFO where the details written to a pipe is first named.

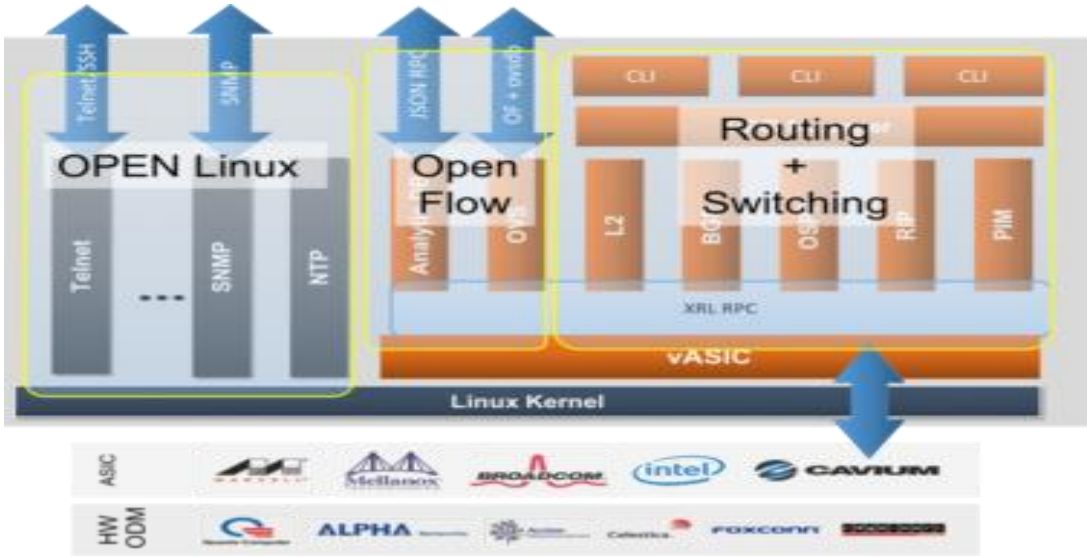
3. **Message Queuing –**
   This allows messages to be passed between processes using either a single queue or several message queue. This is managed by system kernel these messages are coordinated using an API.

4. **Semaphores –**
   This is used in solving problems associated with synchronization and to avoid race condition. These are integer values which are greater than or equal to 0.

5. **Shared memory –**
   This allows the interchange of data through a defined area of memory. Semaphore values have to be obtained before data can get access to shared memory.

6. **Sockets –**
   This method is mostly used to communicate over a network between a client and a server. It allows for a standard connection which is computer and OS independent.

Network Structure in Linux Operating System



Discussing the network structure in a Linux operating system gets a bit complicated. By itself, Linux does not address networking; it is, after all, a server operating system intended to run applications, not networks. OpenStack, however, does provide a networking service that's meant to be used with Linux.

OpenStack is a combination of open source software tools for building and managing virtualized cloud computing services, providing services including compute, storage and identity management. There's also a networking component, Neutron, which enables all the other OpenStack components to communicate with one another. Given that OpenStack was designed to run on a Linux kernel, it could be said that Neutron is a networking service for Linux – but only when used in an OpenStack cloud environment.

Neutron enables network virtualization in an OpenStack environment, providing software-defined network services through a series of plug-ins. It is intended to enable organizations to spin up network services on demand, including virtual LANS and virtual private networks (VPNs), as well as services such as firewalls, intrusion detection and load balancing.

In practice, the networking capabilities of Neutron are somewhat limited, with its main drawback being a lack of scalability. While companies may use Neutron in a lab environment, when it comes to production they typically look for other options.

A number of companies have developed SDN and network virtualization software that is more enterprise-ready. Pica8, for example, offers PICOS, an open network operating system built on a Debian Linux kernel. PICOS is a white box NOS intended to run on white box network switches and be used in a virtualized, SDN environment. But it provides the scalability required to extend to hundreds or thousands of white box switches, making it a viable option for enterprise use.

Windows Operating Systems

Windows is a graphical **operating system** developed by **Microsoft**. It allows users to view and store files, run the software, play games, watch videos, and provides a way to connect to the internet. It was released for both home computing and professional works.

Windows is a general name for Microsoft Windows. It is developed and marketed by an American multinational company Microsoft. Microsoft Windows is a collection of several proprietary graphical **operating systems** that provide a simple method to store files, run the software, play games, watch videos, and connect to the Internet.

What is an Operating System?

An operating system or OS is system software. It provides an interface between computer user and computer hardware. An operating system is used to perform all the basic tasks like file management, process management, memory management, handling input and output devices, and controlling peripheral devices such as disk drives and printers.

History of Windows

Windows was first introduced by Microsoft on 20 November 1985. After that, it was gaining popularity day by day. Now, it is the most dominant desktop operating system around the world, with a market share of around 82.74%. The macOS Operating system by Apple Inc. is the second most popular with the share of 13.23%, and all varieties of Linux operating systems are collectively in third place with the market share of 1.57%.

Early History

Bill Gates is known as the founder of Windows. Microsoft was founded by Bill Gates and Paul Allen, the childhood friends on 4 April 1975 in Albuquerque, New Mexico, U.S.

The first project towards the making of Windows was **Interface Manager**. Microsoft was started to work on this program in 1981, and in November 1983, it was announced under the name "Windows," but Windows 1.0 was not released until November 1985. It was the time of Apple's Macintosh, and that's the reason Windows 1.0 was not capable of competing with Apple's operating system, but it achieved little popularity. Windows 1.0 was just an extension of MS-DOS (an already released Microsoft's product), not a complete operating system. The first Microsoft Windows was a graphical user interface for MS-DOS. But, in the later 1990s, this product was evolved as a fully complete and modern operating system.

Windows Versions

The versions of Microsoft Windows are categorized as follows:

Early versions of Windows

The first version of Windows was Windows 1.0. It cannot be called a complete operating system because it was just an extension of MS-DOS, which was already developed by Microsoft. The shell of Windows 1.0 was a program named MS-DOS Executive. Windows 1.0 had introduced some components like Clock, Calculator, Calendar, Clipboard viewer, Control Panel, Notepad, Paint, Terminal, and Write, etc.

In December 1987, Microsoft released its second Windows version as Windows 2.0. It got more popularity than its previous version Windows 2.0. Windows 2.0 has some improved features in user interface and memory management.

The early versions of Windows acted as graphical shells because they ran on top of MS-DOS and used it for file system services.

Windows 3.x

The third major version of Windows was Windows 3.0. It was released in 1990 and had an improved design. Two other upgrades were released as Windows 3.1 and Windows 3.2 in 1992 and 1994, respectively. Microsoft tasted its first broad commercial success after the release of Windows 3.x and sold 2 million copies in just the first six months of release.

Windows 9x (Windows 95, Windows 98)

Windows 9x was the next release of Windows. Windows 95 was released on 24 August 1995. It was also the MS-DOS-based Windows but introduced support for native 32-bit applications. It provided increased stability over its predecessors, added plug and play hardware, preemptive multitasking, and also long file names of up to 255 characters.

It had two major versions Windows 95 and Windows 98





Windows NT (3.1/3.5/3.51/4.0/2000)

Windows NT was developed by a new development team of Microsoft to make it a secure, multi-user operating system with POSIX compatibility. It was designed with a modular, portable kernel with preemptive multitasking and support for multiple processor architectures.

Windows XP

Windows XP was the next major version of Windows NT. It was first released on 25 October 2001. It was introduced to add security and networking features.

It was the first Windows version that was marketed in two main editions: the "Home" edition and the "Professional" edition.

The "Home" edition was targeted towards consumers for personal computer use, while the "Professional" edition was targeted towards business environments and power users. It included the "Media Center" edition later, which was designed for home theater PCs and provided support for DVD playback, TV tuner cards, DVR functionality, and remote controls, etc.

Windows XP was one of the most successful versions of Windows.

Windows Vista

After Windows XP's immense success, Windows Vista was released on 30 November 2006 for volume licensing and 30 January 2007 for consumers. It had included a lot of new features such as a redesigned shell and user interface to significant technical changes. It extended some security features also.

Windows 7

Windows 7 and its Server edition Windows Server 2008 R2 were released as RTM on 22 July 2009. Three months later, Windows 7 was released to the public. Windows 7 had introduced a large number of new features, such as a redesigned Windows shell with an updated taskbar, multi-touch support, a home networking system called HomeGroup, and many performance improvements.

Windows 7 was supposed to be the most popular version of Windows to date.

Windows 8 and 8.1

Windows 8 was released as the successor to Windows 7. It was released on 26 October, 2012. It had introduced a number of significant changes such as the introduction of a user interface based around Microsoft's Metro design language with optimizations for touch-based devices such as tablets and all-in-one PCs. It was more convenient for touch-screen devices and laptops.

Microsoft released its newer version Windows 8.1 on 17 October 2013 and includes features such as new live tile sizes, deeper OneDrive integration, and many other revisions.

Windows 8 and Windows 8.1 were criticized for the removal of the Start menu.

Windows 10

Microsoft announced Windows 10 as the successor to Windows 8.1 on 30 September 2014. Windows 10 was released on 29 July 2015. Windows 10 is the part of the Windows NT family of operating systems.

Microsoft has not announced any newer version of Windows after Windows 10.

Design Principles

Microsoft's design goals for Windows XP include security, reliability, Windows and POSIX application compatibility, high performance, extensibility, portability, and international support.

Security

Windows XP security goals required more than just adherence to the design standards that enabled Windows NT 4.0 to receive a C-2 security classification from the U.S. government (which signifies a moderate level of protection from defective software and malicious attacks).

Extensive code review and testing were combined with sophisticated automatic analysis tools to identify and investigate potential defects that might represent security vulnerabilities.

Reliability

Windows 2000 was the most reliable, stable operating system Microsoft had ever shipped to that point. Much of this reliability came from maturity in the source code, extensive stress testing of the system, and automatic detection of many serious errors in drivers.

The reliability requirements for Windows XP were even more stringent. Microsoft used extensive manual and automatic code review to identify over 63,000 lines in the source files that might contain issues not detected by testing and then set about reviewing each area to verify that the code was indeed correct.

Windows XP extends driver verification to catch more subtle bugs, improves the facilities for catching programming errors in user-level code, and subjects third-party applications, drivers, and devices to a rigorous certification process.

Furthermore, Windows XP adds new facilities for monitoring the health of the PC, including downloading fixes for problems before they are encountered by users. The perceived reliability of Windows XP was also improved by making the graphical user interface easier to use through better visual design, simpler menus, and measured improvements in the ease with which users can discover how to perform common tasks.

Windows and POSIX Application Compatibility

Windows XP is not only an update of Windows 2000; it is a replacement for Windows 95/98. Windows 2000 focused primarily on compatibility for business applications. The requirements for Windows XP include a much higher compatibility with consumer applications that run on Windows 95/98. Application compatibility is difficult to achieve because each application checks for a particular version of Windows, may have some dependence on the quirks of the implementation of APIs, may have latent application bugs that were masked in the previous system, and so forth.

Windows XP introduces a compatibility layer that falls between applications and the Win32 APIs. This layer makes Windows XP look (almost) bug-for-bug compatible with previous versions of Windows. Windows XP, like earlier NT releases, maintains support for running many 16-bit applications using a thunking, or conversion, layer that translates 16-bit API calls into equivalent 32-bit calls. Similarly, the 64-bit version of Windows XP provides a thunking layer that translates 32-bit API calls into native 64-bit calls.

POSIX support in Windows XP is much improved. A new POSIX subsystem called Interix is now available. Most available UNIX-compatible software compiles and runs under Interix without modification.

High Performance

Windows XP is designed to provide high performance on desktop systems (which are largely constrained by I/O performance), server systems (where the CPU is often the bottleneck), and large multithreaded and multiprocessor environments (where locking and cache-line management are key to scalability). High performance has been an increasingly important goal for Windows XP. Windows 2000 with SQL 2000 on Compaq hardware achieved top TPC-C numbers at the time it shipped.

To satisfy performance requirements, NT uses a variety of techniques, such as asynchronous I/O, optimized protocols for networks (for example, optimistic locking of distributed data, batching of requests), kernel-based graphics, and sophisticated caching of file-system data. The memory-management and synchronization algorithms are designed with an awareness of the performance considerations related to cache lines and multiprocessors.

Windows XP has further improved performance by reducing the code-path length in critical functions, using better algorithms and per-processor data structures, using memory coloring for NUMA (non-uniform memory access) machines, and implementing more scalable locking protocols, such as queued spinlocks. The new locking protocols help reduce system bus cycles and include lock-free lists and queues, use of atomic read-modify-write operations (like interlocked increment), and other advanced locking techniques.

The subsystems that constitute Windows XP communicate with one another efficiently by a local procedure call (LPC) facility that provides highperformance message passing. Except while executing in the kernel dispatcher, threads in the subsystems of Windows XP can be preempted by higher-priority threads. Thus, the system responds quickly to external events. In addition, Windows XP is

designed for symmetrical multiprocessing; on a multiprocessor computer, several threads can run at the same time.

Extensibility

Extensibility refers to the capacity of an operating system to keep up with advances in computing technology. So that changes over time are facilitated, the developers implemented Windows XP using a layered architecture. The Windows XP executive runs in kernel or protected mode and provides the basic system services. On top of the executive, several server subsystems operate in user mode. Among them are environmental subsystems that emulate different operating systems. Thus, programs written for MS-DOS, Microsoft Windows, and POSIX all run on Windows XP in the appropriate environment. Because of the modular structure, additional environmental subsystems can be added without affecting the executive.

In addition, Windows XP uses loadable drivers in the I/O system, so new file systems, new kinds of I/O devices, and new kinds of networking can be added while the system is running. Windows XP uses a client-server model like the Mach operating system and supports distributed processing by remote procedure calls (RPCs) as defined by the Open Software Foundation.

Portability

An operating system is portable if it can be moved from one hardware architecture to another with relatively few changes. Windows XP is designed to be portable. As is true of the UNIX operating system, the majority of the system is written in C and C++. Most processor-dependent code is isolated in a dynamic link library (DLL) called the hardware-abstraction layer (HAL).

A DLL is a file that is mapped into a process's address space such that any functions in the DLL appear to be part of the process. The upper layers of the Windows XP kernel depend on the HAL interfaces rather than on the underlying hardware, bolstering Windows XP portability. The HAL manipulates hardware

directly, isolating the rest of Windows XP from hardware differences among the platforms on which it runs.

Although for market reasons Windows 2000 shipped only on Intel IA32-compatible platforms, it was also tested on IA32 and DEC Alpha platforms until just prior to release to ensure portability. Windows XP runs on IA32-compatible and IA64 processors. Microsoft recognizes the importance of multiplatform development and testing, since, as a practical matter, maintaining portability is a matter of use it or lose it.

International Support

Windows XP is also designed for international and multinational use. It provides support for different locales via the national-language-support (NLS) API. The NLS API provides specialized routines to format dates, time, and money in accordance with various national customs.

String comparisons are specialized to account for varying character sets. UNICODE is Windows XP's native character code. Windows XP supports ANSI characters by converting them to UNICODE characters before manipulating them (8-bit to 16-bit conversion). System text strings are kept in resource files that can be replaced to localize the system for different languages. Multiple locales can be used concurrently, which is important to multilingual individuals and businesses.

Main Components of Windows

The main components of the Windows Operating System are the following:

- Configuration and maintenance
- User interface
- Applications and utilities
- Windows Server components
- File systems
- Core components
- Services
- DirectX

- Networking
- Scripting and command-line
- Kernel
- NET Framework
- Security
- Deprecated components and apps
- APIs

Fast User Switching

Fast user switching is a feature Microsoft implemented within Windows XP Home Edition and Windows XP Professional that allows users to switch between users without having to log out of their user accounts. It allows applications to remain open and in the same state, despite someone new logging in. This allows the previous user to get back to his or her tasks more quickly.

Explained Fast User Switching

Windows allows multiple users to have their own profile set-ups associated with their account. These profiles are usually password protected, and their settings, files and other information are set up to meet each individual user's needs. Setting up different profiles allows multiple users to share one computer. However, there is often a delay when logging in and out of different accounts. This is where fast user switching comes in.

Fast users switching allows multiple users to be logged in simultaneously and switch between their open accounts while other applications are running and network connections are preserved.

Terminal Services

Terminal Services is a component in Microsoft Windows that allows a user to access applications and data on a remote computer over a network. **Terminal Services** is a thin-client terminal server sort of computing environment developed by Microsoft.

Terminal Services allows Windows applications or even the entire desktop of a computer running terminal services to be accessible from a remote client computer.

Widely used these days with **Microsoft Windows Server 2003**, **Terminal Services** provides the ability to host multiple, simultaneous client sessions.

What is Terminal Services good for?

Terminal Services lets administrators install, configure, manage, and maintain applications centrally on a few servers.

Time is money...

This goes in hand with IT budgets and staffing. Managing software in a central location is usually much faster, easier, and cheaper than deploying applications to end-users' desktops. Centrally-deployed applications are also easier to maintain, especially as related to patching and upgrading.

Running applications from one central location also can be beneficial for the configuration of desktops. Since a terminal server hosts all the application logic which also runs on the server, the processing and storage requirements for client machines are minimal.

Terminal Services history

Terminal Services was first introduced in **Windows NT 4.0** Terminal Server Edition. Unfortunately, this early implementation of Terminal Services in Windows NT did not gain too much popularity. Terminal Services has been significantly improved in Windows 2000 and even more in **Windows Server 2003**.

Both the underlying protocol as well as the service was again fundamentally overhauled for Windows Vista and Windows Server 2008.

Are there any limitations on the network connection?

In general, there are no explicit limitations for network connectivity related to terminal services. It can be used over a Local Area Network (LAN) as well as over a Wide Area Network (WAN).

Terminal Services in Windows XP

Windows includes the following two client applications which utilize Terminal Services:

- Remote Assistance
- Remote Desktop

The **Remote Assistance** component is available in all versions of Windows. Remote Assistance allows one user to assist another user.

The **Remote Desktop** application is available in Windows XP Professional, Media Center Edition, Windows Vista Business, Enterprise, and Ultimate. Remote Desktop allows a user to log into a remote system and access the desktop, applications, and data. Remote Desktop can also be used to control the system remotely.

Terminal Services on client versions of Windows versus server Windows

In the client versions of Windows, that is for example Windows XP, **Terminal Services** supports only one logged in user at a time. On the other hand, concurrent remote sessions are allowed in a server Windows operating system, for example the Microsoft Windows Server 2003.

What are the disadvantages of Terminal Services?

As one may expect, running an application from a central location also has some disadvantages.

- The terminal server needs to be powerful enough to be able to handle all connections.

- The network needs to be sized appropriately so that it is not the bottleneck when terminal server sessions are established.
- The terminal server is the major source of risk of downtime. If the terminal server fails, the whole system fails unless a fail-over terminal server is in place.
- The functionality of the system as a whole is also affected by the network reliability. If the network is down, the whole system is down as well.
- Running applications from a terminal server can also be an issue from performance perspective. In some cases, no matter how good the network is, the performance associated with running an application locally on a desktop workstation can still overshadow the benefits of a terminal server environment.

Another disadvantage can be the availability of skilled administrator. Support for a terminal server needs to have the necessary knowledge and be available as the business needs commands.

What Microsoft Terminal Services has to offer?

Terminal Services is a built-in component in **Windows Server 2003**. Terminal Services provides especially the following functionality:

Terminal Services and Group Policy

Terminal Services can be configured and managed through **Group Policy** settings. This is a new feature in Windows Server 2003 which allows administrators to take advantage of the flexibility and power of the Group Policy component to simplify the configuration and management of Windows Terminal servers. User accounts can be assigned permissions based on group policies.

Remote Administration Built in...

The remote administration mode is already built into the operating system and no longer requires installation of additional components. Allowing users to remotely

connect to a server requires just a simple step of selecting a checkbox on the **Remote tab** of the **System** tool in **Control Panel**.

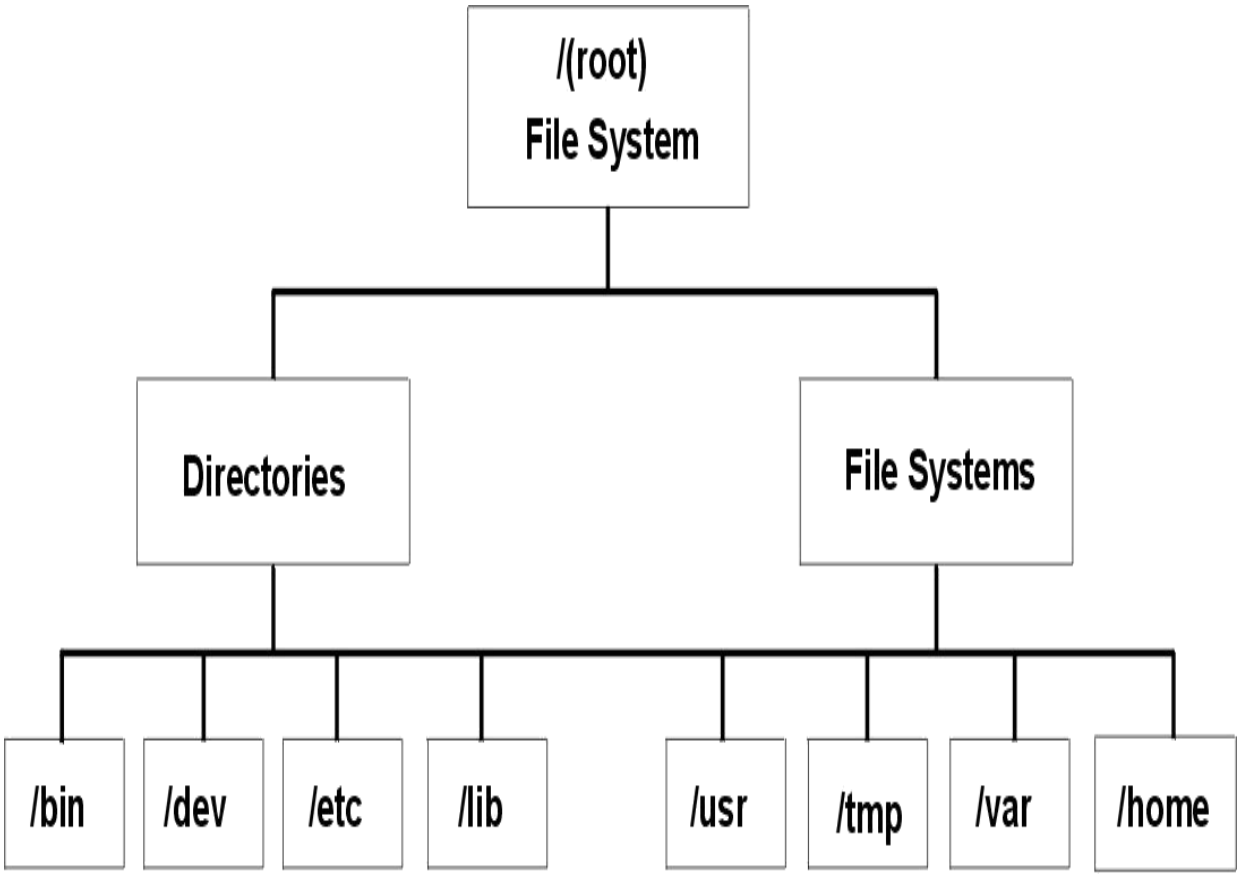
Remote Desktop Protocol (RDP)...

The Remote Desktop Protocol (RDP) has been very much enhanced in Windows Server 2003. The display and device redirection as well as the security have been enhanced. Terminal Services does not need a VPN tunnel anymore when connecting to it over a public network.

Session Directory component...

The terminal server can be configured with so called Session Directory component. This add-in allows to scale Terminal Services upwards. This is used by large enterprises that need a load-balanced terminal server network.

Windows File System



What is file system?

In computing, file system controls how data is stored and retrieved. In other words, it is the method and data structure that an operating system uses to keep track of files on a disk or partition.

It separates the data we put in computer into pieces and gives each piece a name, so the data is easily isolated and identified.

Without file system, information saved in a storage media would be one large body of data with no way to tell where the information begins and ends.

Types of Windows File System

There are five types of Windows file system, such as FAT12, FAT16, FAT32, NTFS and exFAT. Most of us like to choose the latter three, and I would like to introduce them respectively for you.

FAT32 in Windows

In order to overcome the limited volume size of FAT16 (its supported maximum volume size is 2GB) Microsoft designed a new version of the file system FAT32, which then becomes the most frequently used version of the **FAT (File Allocation Table)** file system.

NTFS in Windows

NTFS is the newer drive format. Its full name is New Technology File System. Starting with Windows NT 3.1, it is the default file system of the **Windows NT** family.

Microsoft has released five versions of NTFS, namely v1.0, v1.1, v1.2, v3.0, and v3.1.

exFAT in Windows

exFAT (Extended File Allocation Table) was designed by Microsoft back in 2006 and was a part of the company's Windows CE 6.0 operating system.

This file system was created to be used on flash drives like USB memory sticks and SD cards, which gives a hint for its precursors: FAT32 and FAT16.

Comparisons among the Three Types of Windows File System

Everything comes in advantages and shortcomings. Comparisons among the three types of Windows File System will be showed in following content to help you make a choice about selecting one type of file system.

Compatibility

The three types can work in all versions of Windows.

For FAT32, it also works in game consoles and particularly anything with a USB port; for exFAT, it requires additional software on Linux; for NTFS, it is read only by default with Mac, and may be read only by default with some Linux distributions.

With respect to the ideal use, FAT32 is used on removable drives like USB and Storage Card; exFAT is used for USB flash drives and other external drivers, especially if you need files of more than 4 GB in size; NTFS can be used for servers.

Security

The files belonging to FAT32 and NTFS can be encrypted, but the flies belong to the latter can be compressed.

The encryption and compression in Windows are very useful. If other users do not use your user name to login Windows system, they will fail to open the encrypted and compressed files that created with your user name.

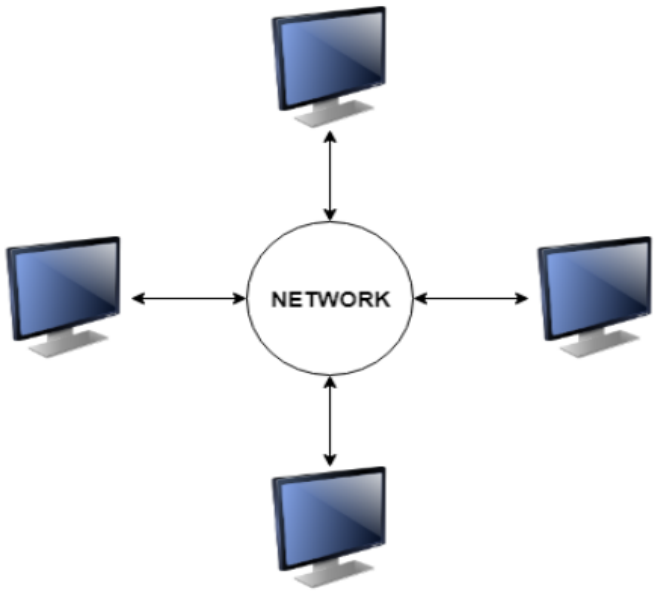
In other word, after some files are encrypted, such files only can be opened when people use our account to login Windows system.

Distributed Systems

A distributed system contains multiple nodes that are physically separate but linked together using the network. All the nodes in this system communicate with

each other and handle processes in tandem. Each of these nodes contains a small part of the distributed operating system software.

A diagram to better explain the distributed system is −



DISTRIBUTED OPERATING SYSTEM

Types of Distributed Systems

The nodes in the distributed systems can be arranged in the form of client/server systems or peer to peer systems. Details about these are as follows −

Client/Server Systems

In client server systems, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network and so they are a part of distributed systems.

Peer to Peer Systems

The peer to peer systems contains nodes that are equal participants in data sharing. All the tasks are equally divided between all the nodes. The nodes

interact with each other as required as share resources. This is done with the help of a network.

Advantages of Distributed Systems

Some advantages of Distributed Systems are as follows −

- All the nodes in the distributed system are connected to each other. So nodes can easily share data with other nodes.
- More nodes can easily be added to the distributed system i.e. it can be scaled as required.
- Failure of one node does not lead to the failure of the entire distributed system. Other nodes can still communicate with each other.
- Resources like printers can be shared with multiple nodes rather than being restricted to just one.

Disadvantages of Distributed Systems

Some disadvantages of Distributed Systems are as follows −

- It is difficult to provide adequate security in distributed systems because the nodes as well as the connections need to be secured.
- Some messages and data can be lost in the network while moving from one node to another.
- The database connected to the distributed systems is quite complicated and difficult to handle as compared to a single user system.
- Overloading may occur in the network if all the nodes of the distributed system try to send data at once.

NETWORK BASED OPERATING SYSTEM

Unlike operating systems, such as Windows, that are designed for single users to control one computer, network operating systems (NOS) coordinate the activities

of multiple computers across a network. The network operating system acts as a director to keep the network running smoothly

Network Based Operating System

The term network operating system is used to refer to two rather different concepts:

1. A specialized operating system for a network device such as a router , switch or firewall

2. An operating system oriented to computer networking, to allow shared file and printer access among multiple computers in a network, to enable the sharing of data, users, groups, security, applications, and other networking functions, typically over a local area network (LAN), or private network. This sense is now largely historical, as common operating systems generally now have such features included

Network operating systems can be embedded in a router or hardware firewall that operates the functions in the network layer(layer 3)

NETWORK DEVICE OPERATING SYSTEM

Examples:

- pfSense, a fork of M0n0wall uses Pf
- IPOS, used in routers from Ericsson
- FortiOS, used in Fortigates from Fortinet
- TiMOS used in routers from Alcatel-Lucent
- Versatile Routing Platform (VRP), used in routers from Huawei
- RouterOS, software which turns a PC or MikroTik hardware into a dedicated router
- Extensible Operating System used in switches from Arista

- ExtremeXOS(EXOS), used in network devices made by Extreme Networks

There are two types of network operating system

Historical Network Operating System

We can think a client as a computer in your network, where a network user is performing some network activity. For Example: Downloading a file from a File Server, Browsing Intranet/Internet etc. The network user normally uses a client computer to perform his day to day work

CLIENT SERVER (NOS)

The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web

Advantages

- Centralized servers are more stable.
- Security is provided through the server.
- New technology and hardware can be easily integrated into the system.
- Hardware and the operating system can be specialized, with a focus on performance.
- Servers are able to be accessed remotely from different locations and types of systems.

- Buying and running a server raises costs.
- Dependence on a central location for operation.
- Requires regular maintenance and updates.

Disadvantages

We can think a client as a computer in your network, where a network user is performing some network activity. For Example: Downloading a file from a File Server, Browsing Intranet/Internet etc. The network user normally uses a client computer to perform his day to day work

Peer to Peer (Nos)

In a peer-to-peer network operating system users are allowed to share resources and files located on their computers and access shared resources from others. This system is not based with having a file server or centralized management source. A peer-to-peer network sets all connected computers equal; they all share the same abilities to use resources available on the network.

- Ease of setup
- Less hardware needed, no server need be acquired

Advantages

- No central location for storage
- Less security than the client–server model

Disadvantages

Why build a distributed system?

- Microprocessors are getting more and more powerful.
- A distributed system combines (and increases) the computing power of individual computer.

- Some advantages include:
    - Resource sharing
      (but not as easily as if on the same machine)
    - Enhanced performance
      (but 2 machines are not as good as a single machine that is 2 times as fast)
    - Improved reliability & availability
      (but probability of single failure increases, as does difficulty of recovery)
    - Modular expandability
- Distributed OS's have not been economically successful!!!

System models:

- the minicomputer model (several minicomputers with each computer supporting multiple users and providing access to remote resources).
- the workstation model (each user has a workstation, the system provides some common services, such as a distributed file system).
- the processor pool model (the model allocates processor to a user according to the user's needs).

Where is the knowledge of distributed operating systems likely to be useful?

- custom OS's for high performance computer systems
- OS subsystems, like NFS, NIS
- distributed ``middleware'' for large computations
- distributed applications

Lack of Global Knowledge

- Communication delays are at the core of the problem
- Information may become false before it can be acted upon
- these create some fundamental problems:
    - no global clock -- scheduling based on fifo queue?
    - no global state -- what is the state of a task? What is a correct program?

Naming

- named objects: computers, users, files, printers, services
- namespace must be large
- unique (or at least unambiguous) names are needed
- logical to physical mapping needed
- mapping must be changeable, expandable, reliable, fast

Scalability

- How large is the system designed for?
- How does increasing number of hosts affect overhead?
- broadcasting primitives, directories stored at every computer -- these design options will not work for large systems.

Compatibility

- Binary level: same architecture (object code)
- Execution level: same source code can be compiled and executed (source code).
- Protocol level: only requires all system components to support a common set of protocols.

Process synchronization

- test-and-set instruction won't work.
- Need all new synchronization mechanisms for distributed systems.

Distributed Resource Management

- Data migration: data are brought to the location that needs them.
    - distributed filesystem (file migration)
    - distributed shared memory (page migration)
- Computation migration: the computation migrates to another location.
    - remote procedure call: computation is done at the remote machine.
    - processes migration: processes are transferred to other processors.

Security

- Authetication: guaranteeing that an entity is what it claims to be.
- Authorization: deciding what privileges an entity has and making only those privileges available.

Structuring

- the monolithic kernel: one piece
- the collective kernel structure: a collection of processes
- object oriented: the services provided by the OS are implemented as a set of objects.
- client-server: servers provide the services and clients use the services.

Communication Networks

- WAN and LAN

- traditional operating systems implement the TCP/IP protocol stack: host to network layer, IP layer, transport layer, application layer.
- Most distributed operating systems are not concerned with the lower layer communication primitives.

Communication Models

- message passing
- remote procedure call (RPC)

Message Passing Primitives

- Send (message, destination), Receive (source, buffer)
- buffered vs. unbuffered
- blocking vs. nonblocking
- reliable vs. unreliable
- synchronous vs. asynchronous

Example: Unix socket I/O primitives

```
#include <sys/socket.h>

ssize_t sendto(int socket, const void *message,

  size_t length, int flags,

  const struct sockaddr *dest_addr, size_t dest_len);

ssize_t recvfrom(int socket, void *buffer,

  size_t length, int flags, struct sockaddr *address,

  size_t *address_len);

int poll(struct pollfd fds[], nfds_t nfds,
```

```
 int timeout);

int select(int nfds, fd_set *readfds, fd_set *writefds,

 fd_set *errorfds, struct timeval *timeout);
```

RPC

With message passing, the application programmer must worry about many details:

- parsing messages
- pairing responses with request messages
- converting between data representations
- knowing the address of the remote machine/server
- handling communication and system failures

RPC is introduced to help hide and automate these details.

RPC is based on a ``virtual'' procedure call model

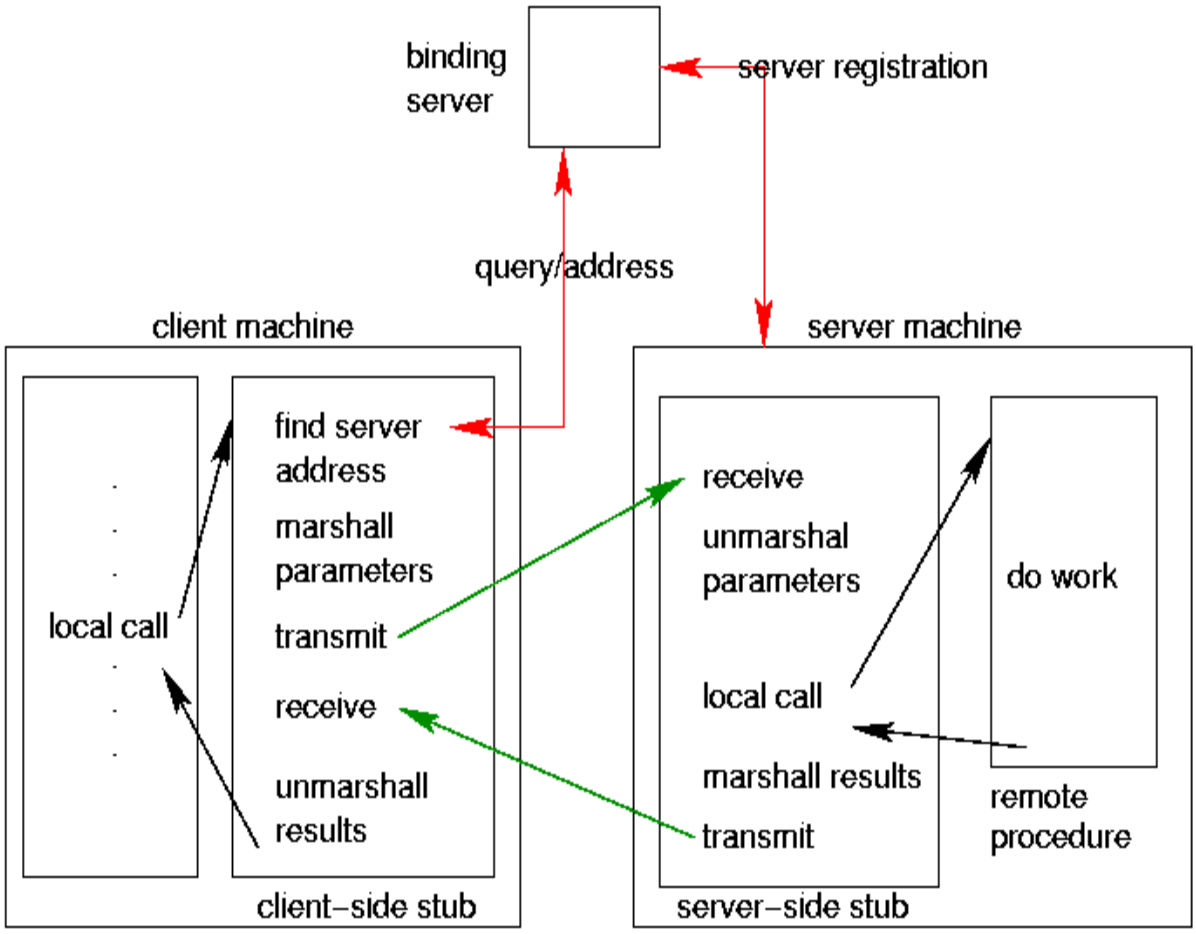
- client calls server, specifying operation and arguments
- server executes operation, returning results

RPC Issues

- Stubs (See Unix rpcgen tool, for example.)
    - are automatically generated, e.g. by compiler
    - do the ``dirty work'' of communication
- Binding method
    - server address may be looked up by service-name
    - or port number may be looked up

- Parameter and result passing
- Error handling semantics

RPC Diagram



Communication Protocols

When we are designing a communication network, we must deal with the inherent complexity of coordinating asynchronous operations communicating in a potentially slow and error-prone environment. In addition, the systems on the network must agree on a protocol or a set of protocols for determining host names, locating hosts on the network, establishing connections, and so on.

We can simplify the design problem (and related implementation) by partitioning the problem into multiple layers. Each layer on one system communicates with

the equivalent layer on other systems. Typically, each layer has its own protocols, and communication takes place between peer layers using a specific protocol. The protocols may be implemented in hardware or software.

For instance, Figure 16.6 shows the logical communications between two computers, with the three lowest-level layers implemented in hardware. Following the International Standards Organization (ISO), we refer to the layers as follows:

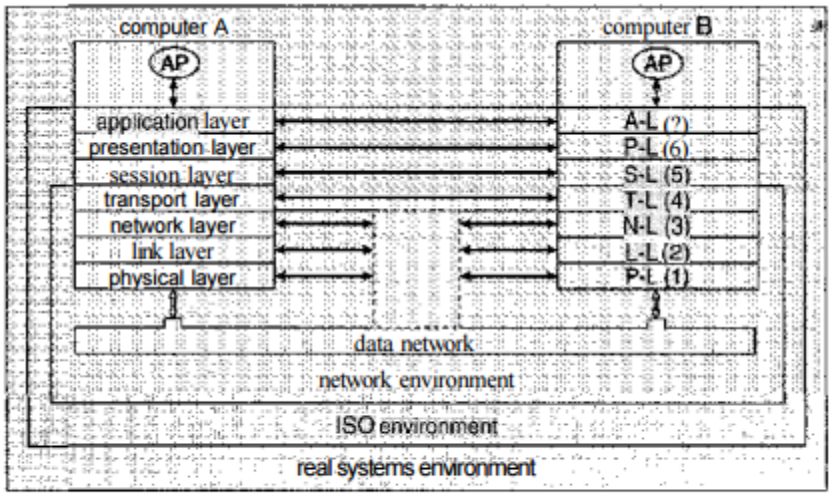


Figure 16.6 Two computers communicating via the ISO network model.

1. **Physical layer.** The physical layer is responsible for handling both the mechanical and the electrical details of the physical transmission of a bit stream. At the physical layer, the communicating systems must agree on the electrical representation of a binary 0 and 1, so that when data are

sent as a stream of electrical signals, the receiver is able to interpret the data properly as binary data. This layer is implemented in the hardware of the networking device.

2. **Data-link layer.** The data-link layer is responsible for handling *frames*, or fixed-length parts of packets, including any error detection and recovery that occurred in the physical layer.
3. **Network layer.** The network layer is responsible for providing connections and for routing packets in the communication network, including handling the addresses of outgoing packets, decoding the addresses of incoming packets, and maintaining routing information for proper response to changing load levels. Routers work at this layer.
4. **Transport layer.** The transport layer is responsible for low-level access to the network and for transfer of messages between clients, including partitioning messages into packets, maintaining packet order, controlling flow, and generating physical addresses.
5. **Session layer.** The session layer is responsible for implementing sessions, or process-to-process communication protocols. Typically, these protocols are the actual communications for remote logins and for file and mail transfers.
6. **Presentation layer.** The presentation layer is responsible for resolving the differences in formats among the various sites in the network, including character conversions and half duplex–full duplex modes (character echoing).
7. **Application layer.** The application layer is responsible for interacting directly with users. This layer deals with file transfer, remote-login protocols, and electronic mail, as well as with schemas for distributed databases.

2.

Figure 16.7 summarizes the ISO protocol stack—a set of cooperating protocols—showing the physical flow of data. As mentioned, logically each layer of a protocol stack communicates with the equivalent layer on other systems. But physically, a message starts at or above the application layer and is passed through each lower level in turn. Each layer may modify the message and include message-header data for the equivalent layer on the receiving side. Ultimately, the message reaches the data-network layer and is transferred as one or more packets (Figure 16.8).

The data-link layer of the target system receives these data, and the message is moved up through the protocol stack; it is analyzed, modified, and stripped of headers as it progresses. It finally reaches the application layer for use by the receiving process.

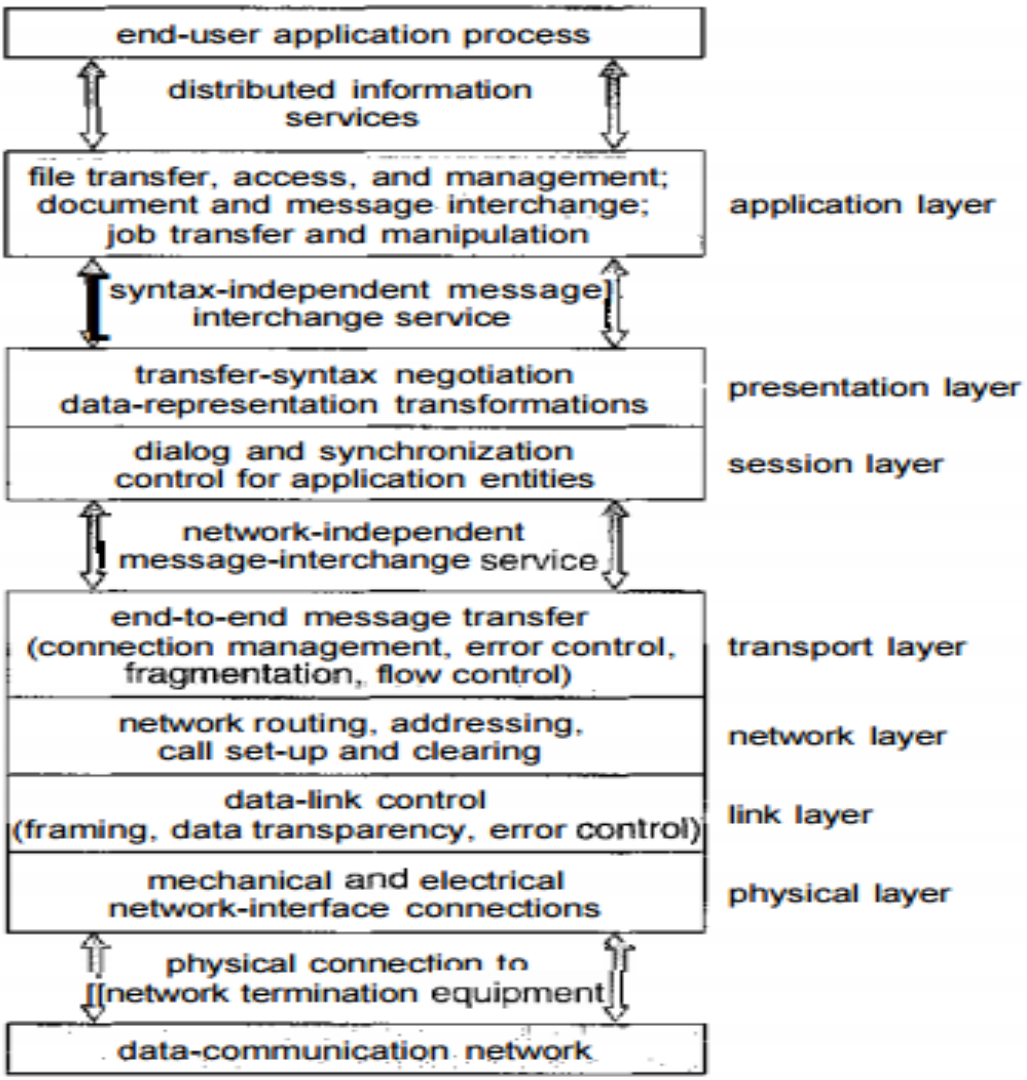


Figure 16.7 The ISO protocol stack.

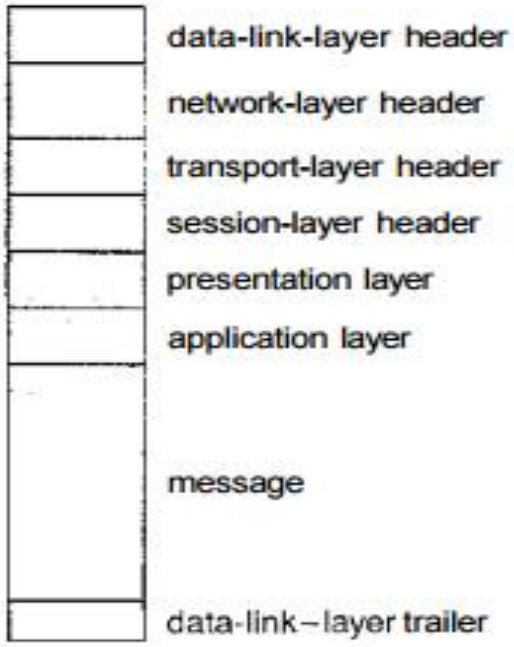


Figure 16.8 An ISO network message.

The ISO model formalizes some of the earlier work done in network protocols but was developed in the late 1970s and is currently not in widespread use. Perhaps the most widely adopted protocol stack is the TCP/IP model, which has been adopted by virtually all Internet sites. The TCP/IP protocol stack has fewer layers than does the ISO model. Theoretically, because it combines several functions in each layer, it is more difficult to implement but more efficient than ISO networking. The relationship between the ISO and TCP/IP models is shown in Figure 16.9.

The TCP/IP application layer identifies several protocols in widespread use in the Internet, including HTTP, FTP, Telnet, DNS, and SMTP. The transport layer identifies the unreliable, connectionless user datagram protocol (UDP) and the reliable, connection-oriented transmission control protocol (TCP). The Internet protocol (IP) is responsible for routing IP datagrams through the Internet. The TCP/IP model does not formally identify a link or physical layer, allowing TCP/IP traffic to run across any physical network. In Section 16.9, we consider the TCP/IP model running over an Ethernet network.

Design Issues of Distributed System

The distributed information system is defined as “a number of interdependent computers linked by a network for sharing information among them”. A distributed information system consists of multiple autonomous computers that communicate or exchange information through a computer network.

Design issues of distributed system –

1. **Heterogeneity** : Heterogeneity is applied to the network, computer hardware, operating system and implementation of different developers. A key component of the heterogeneous distributed system client-server environment is middleware. Middleware is a set of service that enables application and end-user to interacts with each other across a heterogeneous distributed system.

2. **Openness**: The openness of the distributed system is determined primarily by the degree to which new resource sharing services can be made available to the users. Open systems are characterized by the fact that their key interfaces are published. It is based on a uniform communication mechanism and published interface for access to shared resources. It can be constructed from heterogeneous hardware and software.

3. **Scalability**: Scalability of the system should remain efficient even with a significant increase in the number of users and resources connected.

4. **Security** : Security of information system has three components Confidentially, integrity and availability. Encryption protects shared resources, keeps sensitive information secrets when transmitted.

5. **Failure Handling** : When some faults occur in hardware and the software program, it may produce incorrect results or they may stop before they have completed the intended computation so corrective measures should to implemented to handle this case.

Failure handling is difficult in distributed systems because the failure is partial i, e, some components fail while others continue to function.

6. **Concurrency**: There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on the same resources, i.e read, write, and update. Each resource must be safe in a concurrent environment. Any object that represents a shared resource a distributed system must ensure that it operates correctly in a concurrent environment.

7. **Transparency** : Transparency ensures that the distributes system should be perceived as the single entity by the users or the application programmers rather than the collection of autonomous systems, which is cooperating. The user should be unaware of where the services are located and the transferring from a local machine to a remote one should be transparent.

Distributed File System (DFS)

A distributed file system (DFS) is a file system with data stored on a server. The data is accessed and processed as if it was stored on the local client machine. The DFS makes it convenient to share information and files among users on a network in a controlled and authorized way. The server allows the client users to share files and store data just like they are storing the information locally. However, the servers have full control over the data and give access control to the clients.

Distributed file system (DFS) is a method of storing and accessing files based in a client/server architecture. In a distributed file system, one or more central servers store files that can be accessed, with proper authorization rights, by any number of remote clients in the network.

Much like an operating system organizes files in a hierarchical file management system, the distributed system uses a uniform naming convention and a mapping scheme to keep track of where files are located. When the client device retrieves a file from the server, the file appears as a normal file on the client machine, and the user is able to work with the file in the same ways as if it were

stored locally on the workstation. When the user finishes working with the file, it is returned over the network to the server, which stores the now-altered file for retrieval at a later time.

Distributed file systems can be advantageous because they make it easier to distribute documents to multiple clients and they provide a centralized storage system so that client machines are not using their resources to store files.

Explained Distributed File System (DFS)

There has been exceptional growth in network-based computing recently and client/server-based applications have brought revolutions in this area. Sharing storage resources and information on the network is one of the key elements in both local area networks (LANs) and wide area networks (WANs). Different technologies have been developed to bring convenience to sharing resources and files on a network; a distributed file system is one of the processes used regularly.

One process involved in implementing the DFS is giving access control and storage management controls to the client system in a centralized way, managed by the servers. Transparency is one of the core processes in DFS, so files are accessed, stored, and managed on the local client machines while the process itself is actually held on the servers. This transparency brings convenience to the end user on a client machine because the network file system efficiently manages all the processes. Generally, a DFS is used in a LAN, but it can be used in a WAN or over the Internet.

A DFS allows efficient and well-managed data and storage sharing options on a network compared to other options. Another option for users in network-based computing is a shared disk file system. A shared disk file system puts the access control on the client's systems so the data is inaccessible when the client system goes offline. DFS is fault-tolerant and the data is accessible even if some of the network nodes are offline.

A DFS makes it possible to restrict access to the file system depending on access

lists or capabilities on both the servers and the clients, depending on how the protocol is designed.

DIWAKAR EDUCATION HUB

System Software and Operating System Unit – 5 MCQs

As per updated syllabus

DIWAKAR EDUCATION HUB

2020

THE LEARN WITH EXPERTIES

1. The physical devices of a computer :
a) Software
b) Package
c) Hardware
d) System Software

Answer: c
Explanation: Hardware refers to the physical devices of a computer system. Software refers to a collection of programs. A program is a sequence of instructions.

2. Software Package is a group of programs that solve multiple problems.
a) True
b) False

Answer: b
Explanation: The statement is false. The software package is a group of programs that solve a specific problem or perform a specific type of job.

3. _____ refer to renewing or changing components like increasing the main memory, or hard disk capacities, or adding speakers, or modems, etc.
a) Grades
b) Prosody
c) Synthesis
d) Upgrades

Answer: d
Explanation: Upgrades is the right term to be used. Upgrades are installed to renew or implement a new feature. Except for upgrades, hardware is normally one-time expense.

4. The memory unit is made up of _____ bytes.
a) 256
b) 124
c) 4096
d) 3096

Answer: c
Explanation: The memory unit is made up of 4,096 bytes. Memory unit is responsible for the storage of data. It is an important entity in the computer system.

5. Which of the following is not an example of system software?
a) Language Translator
b) Utility Software
c) Communication Software
d) Word Processors

Answer: d
Explanation: A system software is responsible for controlling the operations of a computer system. Word Processor is an application software since it is specific to its purpose.

6. A person who designs the programs in a software package is called :
a) User
b) Software Manager
c) System Developer
d) System Programmer

Answer: d
Explanation: The programs included in a system software package are called system programs. The programmers who design them and prepare them are called system programmers.

7. _____ is designed to solve a specific problem or to do a specific

DIWAKAR EDUCATION HUB

task.
a) Application Software
b) System Software
c) Utility Software
d) User

Answer: a
Explanation: An application software is specific to solving a specific problem. System software is designed for controlling the operations of a computer system.

8. Assembler is used as a translator for?
a) Low level language
b) High Level Language
c) COBOL
d) C

Answer: a
Explanation: Assembler is used in case of low level languages. It is generally used to make the binary code into an understandable format. Interpreter is used with the high level languages similarly.

9. What do you call a program in execution?
a) Command
b) Process
c) Task
d) Instruction

Answer: b
Explanation: Option Process is correct. A program is a set of instructions. A program in execution is called a process.

10. Which of the following is not a process state?
a) Terminated
b) Running
c) Blocked
d) Execution

Answer: c
Explanation: There is no blocked state in a process model. The different states are ready, running, executing, waiting and terminated.

11. The language made up of binary coded instructions.
a) Machine
b) C
c) BASIC
d) High level

Answer: a
Explanation: The language made up of binary coded instructions built into the hardware of a particular computer and used directly by the computer is machine language.

12. Binary code comprises of digits from 0 to 9.
a) True
b) False

Answer: b
Explanation: The statement is false. Binary as the word suggests contains only 2 digits : 0 and 1.
0 denotes false and 1 denotes a truth value.

13. The _____ contains the address of the next instruction to be executed.
a) IR
b) PC
c) Accumulator
d) System counter

DIWAKAR EDUCATION HUB

Answer: b
Explanation: PC stands for program counter (It contains the address of the next instruction to be executed).

14. A document that specifies how many times and with what data the program must be run in order to thoroughly test it.
a) addressing plan
b) test plan
c) validation plan
d) verification plan

Answer: b
Explanation: Test plan is the A document that specifies how many times and with what data the program must be run in order to thoroughly test it. It comes under testing.

15. Each personal computer has a _____ that manages the computer's arithmetical, logical and control activities.
a) Microprocessor
b) Assembler
c) Microcontroller
d) Interpreter

Answer: a
Explanation: Microprocessor handles all these activities. Each family of processors has its own set of instructions for handling various operations like getting input from keyboard, displaying information on a screen and performing various other jobs.

16. Assembly Language requires less memory and execution time.
a) True
b) False

Answer: a
Explanation: The statement is true.
Advantages of using assembly language are:
• It requires less memory and execution time.
• It allows hardware-specific complex jobs in an easier way.
• It is suitable for time-critical jobs.

17. The data size of a word is _____
a) 2-byte
b) 4-byte
c) 8-byte
d)16-byte

Answer: a
Explanation: The processor supports the following data sizes:
• Word: a 2-byte data item
• Double word: a 4-byte (32 bit) data item, etc.

18. A direct reference of specific location.
a) Segment Address
b) Absolute Address
c) Offset
d) Memory Address

Answer: b
Explanation: There are two kinds of memory addresses:
• An absolute address – a direct reference of specific location.
• The segment address (or offset) – starting address of a memory segment with the offset value.

19. A Borland Turbo Assembler.
a) nasm
b) tasm

DIWAKAR EDUCATION HUB

c) gas
d) asm

Answer: b
Explanation: Tasm is the borland turbo assembler. Nasm is used with linux generally. Gas is the GNU assembler.

20. Prolog comes under _____
a) Logic Programming
b) Procedural Programming
c) OOP
d) Functional

Answer: a
Explanation: Prolog stands for Programming in Logic. The options mentioned are the four categories of programming. Prolog is a type of logic programming.

21. Java is procedural programming.
a) True
b) False

Answer: b
Explanation: The statement is false. Java is a type of object oriented programming language. It involves solving real-life problems as well.

22. A program that can execute high-level language programs.
a) Compiler
b) Interpreter
c) Sensor
d) Circuitry

Answer: b
Explanation: Interpreter is a program that can execute high-level language programs "directly," without first being translated into machine language.

23. Executables might be called _____
a) native code
b) executable code
c) complex code
d) machine code

Answer: a
Explanation: The executables are sometimes called native code. HLL are translated to Machine language called the native code.

24. Source program is compiled to an intermediate form called _____
a) Byte Code
b) Smart code
c) Executable code
d) Machine code

Answer: a
Explanation: The Source program is compiled to an intermediate form called byte code. For each supported platform, write a "virtual machine" emulator that reads byte code and emulates its execution.

25. What is operating system?
a) collection of programs that manages hardware resources
b) system service provider to the application programs
c) link to interface the hardware and application programs
d) all of the mentioned

Answer: d
Explanation: None.

DIWAKAR EDUCATION HUB

26. To access the services of operating system, the interface is provided by the _____
a) System calls
b) API
c) Library
d) Assembly instructions

Answer: a
Explanation: None.

27. Which one of the following is not true?
a) kernel is the program that constitutes the central core of the operating system
b) kernel is the first part of operating system to load into memory during booting
c) kernel is made of various modules which can not be loaded in running operating system
d) kernel remains in the memory during the entire computer session

Answer: c
Explanation: None.

28. Which one of the following error will be handle by the operating system?
a) power failure
b) lack of paper in printer
c) connection failure in the network
d) all of the mentioned

Answer: d
Explanation: None.

29. What is the main function of the command interpreter?
a) to get and execute the next user-specified command
b) to provide the interface between the API and application program
c) to handle the files in operating system
d) none of the mentioned

Answer: a
Explanation: None.

30. By operating system, the resource management can be done via _____
a) time division multiplexing
b) space division multiplexing
c) time and space division multiplexing
d) none of the mentioned

Answer: c
Explanation: None.

31. If a process fails, most operating system write the error information to a _____
a) log file
b) another running process
c) new file
d) none of the mentioned

Answer: a
Explanation: None.

32. Which facility dynamically adds probes to a running system, both in user processes and in the kernel?
a) DTrace
b) DLocate
c) DMap
d) DAdd

Answer: a
Explanation: None.

33. Which one of the following is not a real time operating system?
a) VxWorks
b) Windows CE

DIWAKAR EDUCATION HUB

c) RTLinux
d) Palm OS

Answer: d
Explanation: None.

34. The OS X has _____
a) monolithic kernel
b) hybrid kernel
c) microkernel
d) monolithic kernel with modules

Answer: b
Explanation: None.

35. The initial program that is run when the computer is powered up is called _____
a) boot program
b) bootloader
c) initializer
d) bootstrap program

Answer: d
Explanation: None.

36. How does the software trigger an interrupt?
a) Sending signals to CPU through bus
b) Executing a special operation called system call
c) Executing a special program called system program
d) Executing a special program called interrupt trigger program

Answer: b
Explanation: None.

37. What is a trap/exception?
a) hardware generated interrupt caused by an error
b) software generated interrupt caused by an error
c) user generated interrupt caused by an error
d) none of the mentioned

Answer: b
Explanation: None.

38. What is an ISR?
a) Information Service Request
b) Interrupt Service Request
c) Interrupt Service Routine
d) Information Service Routine

Answer: c
Explanation: None.

39. What is an interrupt vector?
a) It is an address that is indexed to an interrupt handler
b) It is a unique device number that is indexed by an address
c) It is a unique identity given to an interrupt
d) None of the mentioned

Answer: a
Explanation: None.

40. The systems which allow only one process execution at a time, are called _____
a) uniprogramming systems
b) uniprocessing systems
c) unitasking systems
d) none of the mentioned

Answer: b
Explanation: Those systems which allows

DIWAKAR EDUCATION HUB

more than one process execution at a time, are called multiprogramming systems. Uniprocessing means only one processor.

41. In operating system, each process has its own _____
a) address space and global variables
b) open files
c) pending alarms, signals and signal handlers
d) all of the mentioned

Answer: d
Explanation: None.

42. In Unix, Which system call creates the new process?
a) fork
b) create
c) new
d) none of the mentioned

Answer: a
Explanation: None.

43. A process can be terminated due to _____
a) normal exit
b) fatal error
c) killed by another process
d) all of the mentioned

Answer: d
Explanation: None.

44. What is the ready state of a process?
a) when process is scheduled to run after some execution
b) when process is unable to run until some task has been completed
c) when process is using the CPU
d) none of the mentioned

Answer: a
Explanation: When process is unable to run until some task has been completed, the process is in blocked state and if process is using the CPU, it is in running state.

45. What is interprocess communication?
a) communication within the process
b) communication between two process
c) communication between two threads of same process
d) none of the mentioned

Answer: b
Explanation: None.

46. A set of processes is deadlock if _____
a) each process is blocked and will remain so forever
b) each process is terminated
c) all processes are trying to kill each other
d) none of the mentioned

Answer: a
Explanation: None.

47. A process stack does not contain _____
a) Function parameters
b) Local variables
c) Return addresses
d) PID of child process

Answer: d
Explanation: None.

DIWAKAR EDUCATION HUB

48. Which system call returns the process identifier of a terminated child?
a) wait
b) exit
c) fork
d) get

Answer: a
Explanation: None.

49. The address of the next instruction to be executed by the current process is provided by the _____
a) CPU registers
b) Program counter
c) Process stack
d) Pipe

Answer: b
Explanation: None.

50. Which of the following do not belong to queues for processes?
a) Job Queue
b) PCB queue
c) Device Queue
d) Ready Queue

Answer: b
Explanation: None.

51. When the process issues an I/O request _____
a) It is placed in an I/O queue
b) It is placed in a waiting queue
c) It is placed in the ready queue
d) It is placed in the Job queue

Answer: a
Explanation: None.

52. What will happen when a process terminates?
a) It is removed from all queues
b) It is removed from all, but the job queue
c) Its process control block is de-allocated
d) Its process control block is never de-allocated

Answer: a
Explanation: None.

53. Which process can be affected by other processes executing in the system?
a) cooperating process
b) child process
c) parent process
d) init process

Answer: a
Explanation: None.

54. When several processes access the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called?
a) dynamic condition
b) race condition
c) essential condition
d) critical condition

Answer: b
Explanation: None.

55. If a process is executing in its critical section, then no other processes can be executing in their critical section. This condition is called?
a) mutual exclusion
b) critical exclusion

DIWAKAR EDUCATION HUB

c) synchronous exclusion
d) asynchronous exclusion

Answer: a
Explanation: None.

56. Which one of the following is a synchronization tool?
a) thread
b) pipe
c) semaphore
d) socket

Answer: c
Explanation: None.

57. A semaphore is a shared integer variable _____
a) that can not drop below zero
b) that can not be more than zero
c) that can not drop below one
d) that can not be more than one

Answer: a
Explanation: None.

58. Mutual exclusion can be provided by the _____
a) mutex locks
b) binary semaphores
c) both mutex locks and binary semaphores
d) none of the mentioned

Answer: c
Explanation: Binary Semaphores are known as mutex locks.

59. When high priority task is indirectly preempted by medium priority task effectively inverting the relative priority of the two tasks, the scenario is called _____
a) priority inversion
b) priority removal
c) priority exchange
d) priority modification

Answer: a
Explanation: None.

60. Process synchronization can be done on _____
a) hardware level
b) software level
c) both hardware and software level
d) none of the mentioned

Answer: c
Explanation: None.

61. What is Inter process communication?
a) allows processes to communicate and synchronize their actions when using the same address space
b) allows processes to communicate and synchronize their actions without using the same address space
c) allows the processes to only synchronize their actions without communication
d) none of the mentioned

Answer: b
Explanation: None.

62. Message passing system allows processes to _____
a) communicate with one another without resorting to shared data
b) communicate with one another by resorting to shared data
c) share data

DIWAKAR EDUCATION HUB

d) name the recipient or sender of the message

Answer: a
Explanation: None.

63. Which of the following two operations are provided by the IPC facility?
a) write & delete message
b) delete & receive message
c) send & delete message
d) receive & send message

Answer: d
Explanation: None.

64. Messages sent by a process _____
a) have to be of a fixed size
b) have to be a variable size
c) can be fixed or variable sized
d) None of the mentioned

Answer: c
Explanation: None.

65. The link between two processes P and Q to send and receive messages is called _____
a) communication link
b) message-passing link
c) synchronization link
d) all of the mentioned

Answer: a
Explanation: None.

66. Which of the following are TRUE for direct communication?
a) A communication link can be associated with N number of process(N = max. number of processes supported by system)
b) A communication link can be associated with exactly two processes
c) Exactly N/2 links exist between each pair of processes(N = max. number of processes supported by system)
d) Exactly two link exists between each pair of processes

Answer: b
Explanation: None.

67. In indirect communication between processes P and Q _____
a) there is another process R to handle and pass on the messages between P and Q
b) there is another machine between the two processes to help communication
c) there is a mailbox to help communication between P and Q
d) none of the mentioned

Answer: c
Explanation: None.

68. In the non blocking send _____
a) the sending process keeps sending until the message is received
b) the sending process sends the message and resumes operation
c) the sending process keeps sending until it receives a message
d) none of the mentioned

Answer: b
Explanation: None.

69. In the Zero capacity queue _____
a) the queue can store at least one message
b) the sender blocks until the receiver receives the message
c) the sender keeps sending and the

DIWAKAR EDUCATION HUB

messages don't wait in the queue
d) none of the mentioned

Answer: b
Explanation: None.

70. The Zero Capacity queue _____
a) is referred to as a message system with buffering
b) is referred to as a message system with no buffering
c) is referred to as a link
d) none of the mentioned

Answer: b
Explanation: None.

71. Which module gives control of the CPU to the process selected by the short-term scheduler?
a) dispatcher
b) interrupt
c) scheduler
d) none of the mentioned

Answer: a
Explanation: None.

72. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called _____
a) job queue
b) ready queue
c) execution queue
d) process queue

Answer: b
Explanation: None.

73. The interval from the time of submission of a process to the time of completion is termed as _____
a) waiting time
b) turnaround time
c) response time
d) throughput

Answer: b
Explanation: None.

74. Which scheduling algorithm allocates the CPU first to the process that requests the CPU first?
a) first-come, first-served scheduling
b) shortest job scheduling
c) priority scheduling
d) none of the mentioned

Answer: a
Explanation: None.

75. In priority scheduling algorithm _____
a) CPU is allocated to the process with highest priority
b) CPU is allocated to the process with lowest priority
c) Equal priority processes can not be scheduled
d) None of the mentioned

Answer: a
Explanation: None.

76. In priority scheduling algorithm, when a process arrives at the ready queue, its priority is compared with the priority of _____
a) all process
b) currently running process

DIWAKAR EDUCATION HUB

c) parent process
d) init process

Answer: b
Explanation: None.

77. Which algorithm is defined in Time quantum?
a) shortest job scheduling algorithm
b) round robin scheduling algorithm
c) priority scheduling algorithm
d) multilevel queue scheduling algorithm

Answer: b
Explanation: None.

78. Process are classified into different groups in _____
a) shortest job scheduling algorithm
b) round robin scheduling algorithm
c) priority scheduling algorithm
d) multilevel queue scheduling algorithm

Answer: d
Explanation: None.

79. CPU scheduling is the basis of _____
a) multiprocessor systems
b) multiprogramming operating systems
c) larger memory sized systems
d) none of the mentioned

Answer: b
Explanation: None.

80. With multiprogramming _____ is used productively.
a) time
b) space
c) money
d) all of the mentioned

Answer: a
Explanation: None.

81. What are the two steps of a process execution?
a) I/O & OS Burst
b) CPU & I/O Burst
c) Memory & I/O Burst
d) OS & Memory Burst

Answer: b
Explanation: None.

82. An I/O bound program will typically have _____
a) a few very short CPU bursts
b) many very short I/O bursts
c) many very short CPU bursts
d) a few very short I/O bursts

Answer: c
Explanation: None.

83. A process is selected from the _____ queue by the _____ scheduler, to be executed.
a) blocked, short term
b) wait, long term
c) ready, short term
d) ready, long term

Answer: c
Explanation: None.

84. Round robin scheduling falls under the category of _____
a) Non-preemptive scheduling
b) Preemptive scheduling

DIWAKAR EDUCATION HUB

c) All of the mentioned
d) None of the mentioned

Answer: b
Explanation: None.

85. With round robin scheduling algorithm in a time shared system _____
a) using very large time slices converts it into First come First served scheduling algorithm
b) using very small time slices converts it into First come First served scheduling algorithm
c) using extremely small time slices increases performance
d) using very small time slices converts it into Shortest Job First algorithm

Answer: a
Explanation: All the processes will be able to get completed.

86. The portion of the process scheduler in an operating system that dispatches processes is concerned with _____
a) assigning ready processes to CPU
b) assigning ready processes to waiting queue
c) assigning running processes to blocked queue
d) all of the mentioned

Answer: a
Explanation: None.

87. Complex scheduling algorithms _____
a) are very appropriate for very large computers
b) use minimal resources
c) use many resources
d) all of the mentioned

Answer: a
Explanation: Large computers are overloaded with a greater number of processes.

88. What is FIFO algorithm?
a) first executes the job that came in last in the queue
b) first executes the job that came in first in the queue
c) first executes the job that needs minimal processor
d) first executes the job that has maximum processor needs

Answer: b
Explanation: None.

89. The strategy of making processes that are logically runnable to be temporarily suspended is called _____
a) Non preemptive scheduling
b) Preemptive scheduling
c) Shortest job first
d) First come First served

Answer: b
Explanation: None.

90. What is Scheduling?
a) allowing a job to use the processor
b) making proper use of processor
c) all of the mentioned
d) none of the mentioned

Answer: a
Explanation: None.

DIWAKAR EDUCATION HUB

91. Which is the most optimal scheduling algorithm?
a) FCFS – First come First served
b) SJF – Shortest Job First
c) RR – Round Robin
d) None of the mentioned

Answer: b
Explanation: None.

92. The real difficulty with SJF in short term scheduling is _____
a) it is too good an algorithm
b) knowing the length of the next CPU request
c) it is too complex to understand
d) none of the mentioned

Answer: b
Explanation: None.

93. The FCFS algorithm is particularly troublesome for _____
a) time sharing systems
b) multiprogramming systems
c) multiprocessor systems
d) operating systems

Answer: b
Explanation: In a time sharing system, each user needs to get a share of the CPU at regular intervals.

94. Consider the following set of processes, the length of the CPU burst time given in milliseconds.

Process	Burst time
P1	6
P2	8
P3	7
P4	3

Assuming the above process being scheduled with the SJF scheduling algorithm.
a) The waiting time for process P1 is 3ms
b) The waiting time for process P1 is 0ms
c) The waiting time for process P1 is 16ms
d) The waiting time for process P1 is 9ms

Answer: a
Explanation: None.

95. Preemptive Shortest Job First scheduling is sometimes called _____
a) Fast SJF scheduling
b) EDF scheduling – Earliest Deadline First
c) HRRN scheduling – Highest Response Ratio Next
d) SRTN scheduling – Shortest Remaining Time Next

Answer: d
Explanation: None.

96. An SJF algorithm is simply a priority algorithm where the priority is _____
a) the predicted next CPU burst
b) the inverse of the predicted next CPU burst
c) the current CPU burst
d) anything the user wants

Answer: a
Explanation: The larger the CPU burst, the lower the priority.

DIWAKAR EDUCATION HUB

97. Choose one of the disadvantages of the priority scheduling algorithm?
a) it schedules in a very complex manner
b) its scheduling takes up a lot of time
c) it can lead to some low priority process waiting indefinitely for the CPU
d) none of the mentioned

Answer: c
Explanation: None.

98. Concurrent access to shared data may result in _____
a) data consistency
b) data insecurity
c) data inconsistency
d) none of the mentioned

Answer: c
Explanation: None.

99. A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which access takes place is called _____
a) data consistency
b) race condition
c) aging
d) starvation

Answer: b
Explanation: None.

100. The segment of code in which the process may change common variables, update tables, write into files is known as _____
a) program
b) critical section
c) non – critical section
d) synchronizing

Answer: b
Explanation: None.

101. Which of the following conditions must be satisfied to solve the critical section problem?
a) Mutual Exclusion
b) Progress
c) Bounded Waiting
d) All of the mentioned

Answer: d
Explanation: None.

102. Mutual exclusion implies that _____
a) if a process is executing in its critical section, then no other process must be executing in their critical sections
b) if a process is executing in its critical section, then other processes must be executing in their critical sections
c) if a process is executing in its critical section, then all the resources of the system must be blocked until it finishes execution
d) none of the mentioned
Answer: a
Explanation: None.

103. Bounded waiting implies that there exists a bound on the number of times a process is allowed to enter its critical section _____
a) after a process has made a request to enter its critical section and before the request is granted
b) when another process is in its critical

DIWAKAR EDUCATION HUB

section
c) before a process has made a request to enter its critical section
d) none of the mentioned

Answer: a
Explanation: None.

105. A minimum of _____ variable(s) is/are required to be shared between processes to solve the critical section problem.
a) one
b) two
c) three
d) four

Answer: b
Explanation: None.

106. An un-interruptible unit is known as _____
a) single
b) atomic
c) static
d) none of the mentioned

Answer: b
Explanation: None.

107. TestAndSet instruction is executed _____
a) after a particular process
b) periodically
c) atomically
d) none of the mentioned

Answer: c
Explanation: None.

108. Semaphore is a/an _____ to solve the critical section problem.
a) hardware for a system
b) special program for a system
c) integer variable
d) none of the mentioned

Answer: c
Explanation: None.

109. What are the two atomic operations permissible on semaphores?
a) wait
b) stop
c) hold
d) none of the mentioned

Answer: a
Explanation: None.

110. What are Spinlocks?
a) CPU cycles wasting locks over critical sections of programs
b) Locks that avoid time wastage in context switches
c) Locks that work better on multiprocessor systems
d) All of the mentioned

Answer: d
Explanation: None.

111. What is the main disadvantage of spinlocks?
a) they are not sufficient for many process
b) they require busy waiting
c) they are unreliable sometimes
d) they are too complex for programmers

Answer: b
Explanation: None.

DIWAKAR EDUCATION HUB

112. The wait operation of the semaphore basically works on the basic _____ system call.
a) stop()
b) block()
c) hold()
d) wait()

Answer: b
Explanation: None.

113. What will happen if a non-recursive mutex is locked more than once?
a) Starvation
b) Deadlock
c) Aging
d) Signaling

Answer: b
Explanation: If a thread which had already locked a mutex, tries to lock the mutex again, it will enter into the waiting list of that mutex, which results in a deadlock. It is because no other thread can unlock the mutex.

114. What is a semaphore?
a) is a binary mutex
b) must be accessed from only one process
c) can be accessed from multiple processes
d) none of the mentioned

Answer: c
Explanation: None.

15. What are the two kinds of semaphores?
a) mutex & counting
b) binary & counting
c) counting & decimal
d) decimal & binary

Answer: b
Explanation: None.

116. What is a mutex?
a) is a binary mutex
b) must be accessed from only one process
c) can be accessed from multiple processes
d) none of the mentioned

Answer: b
Explanation: None.

117. At a particular time of computation the value of a counting semaphore is 7.Then 20 P operations and 15 V operations were completed on this semaphore. The resulting value of the semaphore is?
a) 42
b) 2
c) 7
d) 12

Answer: b
Explanation: P represents Wait and V represents Signal. P operation will decrease the value by 1 every time and V operation will increase the value by 1 every time.

118. The bounded buffer problem is also known as _____
a) Readers – Writers problem
b) Dining – Philosophers problem
c) Producer – Consumer problem
d) None of the mentioned

Answer: c
Explanation: None.

119. In the bounded buffer problem, there are the empty and full semaphores that _____

DIWAKAR EDUCATION HUB

a) count the number of empty and full buffers
b) count the number of empty and full memory spaces
c) count the number of empty and full queues
d) none of the mentioned

Answer: a
Explanation: None.

120. In the bounded buffer problem _____
a) there is only one buffer
b) there are n buffers (n being greater than one but finite)
c) there are infinite buffers
d) the buffer size is bounded

Answer: b
Explanation: None.

121. To ensure difficulties do not arise in the readers – writers problem _____ are given exclusive access to the shared object.
a) readers
b) writers
c) readers and writers
d) none of the mentioned

Answer: b
Explanation: None.

122. The dining – philosophers problem will occur in case of _____
a) 5 philosophers and 5 chopsticks
b) 4 philosophers and 5 chopsticks
c) 3 philosophers and 5 chopsticks
d) 6 philosophers and 5 chopsticks

Answer: a
Explanation: None.

123. A deadlock free solution to the dining philosophers problem _____
a) necessarily eliminates the possibility of starvation
b) does not necessarily eliminate the possibility of starvation
c) eliminates any possibility of any kind of problem further
d) none of the mentioned

Answer: b
Explanation: None.

124. What is a reusable resource?
a) that can be used by one process at a time and is not depleted by that use
b) that can be used by more than one process at a time
c) that can be shared between various threads
d) none of the mentioned

Answer: a
Explanation: None.

125. Which of the following condition is required for a deadlock to be possible?
a) mutual exclusion
b) a process may hold allocated resources while awaiting assignment of other resources
c) no resource can be forcibly removed from a process holding it
d) all of the mentioned

Answer: d
Explanation: None.

DIWAKAR EDUCATION HUB

126. A system is in the safe state if _____
a) the system can allocate resources to each process in some order and still avoid a deadlock
b) there exist a safe sequence
c) all of the mentioned
d) none of the mentioned

Answer: a
Explanation: None.

127. The circular wait condition can be prevented by _____
a) defining a linear ordering of resource types
b) using thread
c) using pipes
d) all of the mentioned

Answer: a
Explanation: None.

128. Which one of the following is the deadlock avoidance algorithm?
a) banker's algorithm
b) round-robin algorithm
c) elevator algorithm
d) karn's algorithm

Answer: a
Explanation: None.

129. What is the drawback of banker's algorithm?
a) in advance processes rarely know how much resource they will need
b) the number of processes changes as time progresses
c) resource once available can disappear
d) all of the mentioned

Answer: d
Explanation: None.

130. For an effective operating system, when to check for deadlock?
a) every time a resource request is made
b) at fixed time intervals
c) every time a resource request is made at fixed time intervals
d) none of the mentioned

Answer: c
Explanation: None.

131. A problem encountered in multitasking when a process is perpetually denied necessary resources is called _____
a) deadlock
b) starvation
c) inversion
d) aging

Answer: b
Explanation: None.

132. Which one of the following is a visual (mathematical) way to determine the deadlock occurrence?
a) resource allocation graph
b) starvation graph
c) inversion graph
d) none of the mentioned

Answer: a
Explanation: None.

133. To avoid deadlock _____
a) there must be a fixed number of resources to allocate
b) resource allocation must be done only once

DIWAKAR EDUCATION HUB

c) all deadlocked processes must be aborted
d) inversion technique can be used

Answer: a
Explanation: None.

134. The number of resources requested by a process _____
a) must always be less than the total number of resources available in the system
b) must always be equal to the total number of resources available in the system
c) must not exceed the total number of resources available in the system
d) must exceed the total number of resources available in the system

Answer: c
Explanation: None.

135. The request and release of resources are _____
a) command line statements
b) interrupts
c) system calls
d) special programs

Answer: c
Explanation: None.

136. What are Multithreaded programs?
a) lesser prone to deadlocks
b) more prone to deadlocks
c) not at all prone to deadlocks
d) none of the mentioned

Answer: b
Explanation: Multiple threads can compete for shared resources.

137. For a deadlock to arise, which of the following conditions must hold simultaneously?
a) Mutual exclusion
b) No preemption
c) Hold and wait
d) All of the mentioned

Answer: d
Explanation: None.

138. For Mutual exclusion to prevail in the system _____
a) at least one resource must be held in a non sharable mode
b) the processor must be a uniprocessor rather than a multiprocessor
c) there must be at least one resource in a sharable mode
d) all of the mentioned

Answer: a
Explanation: If another process requests that resource (non – shareable resource), the requesting process must be delayed until the resource has been released.

139. For a Hold and wait condition to prevail _____
a) A process must be not be holding a resource, but waiting for one to be freed, and then request to acquire it
b) A process must be holding at least one resource and waiting to acquire additional resources that are being held by other processes
c) A process must hold at least one resource and not be waiting to acquire additional resources
d) None of the mentioned

DIWAKAR EDUCATION HUB

Answer: b
Explanation: None.

140. Each request requires that the system consider the _____ to decide whether the current request can be satisfied or must wait to avoid a future possible deadlock.
a) resources currently available
b) processes that have previously been in the system
c) resources currently allocated to each process
d) future requests and releases of each process

Answer: a
Explanation: None.

141. Given a priori information about the _____ number of resources of each type that maybe requested for each process, it is possible to construct an algorithm that ensures that the system will never enter a deadlock state.
a) minimum
b) average
c) maximum
d) approximate

Answer: c
Explanation: None.

142. A deadlock avoidance algorithm dynamically examines the _____ to ensure that a circular wait condition can never exist.
a) resource allocation state
b) system storage state
c) operating system
d) resources

Answer: a
Explanation: Resource allocation states are used to maintain the availability of the already and current available resources.

143. A state is safe, if _____
a) the system does not crash due to deadlock occurrence
b) the system can allocate resources to each process in some order and still avoid a deadlock
c) the state keeps the system protected and safe
d) all of the mentioned

Answer: b
Explanation: None.

144. A system is in a safe state only if there exists a _____
a) safe allocation
b) safe resource
c) safe sequence
d) all of the mentioned

Answer: c
Explanation: None.

145. All unsafe states are _____
a) deadlocks
b) not deadlocks
c) fatal
d) none of the mentioned

Answer: b
Explanation: None.

DIWAKAR EDUCATION HUB

146. The wait-for graph is a deadlock detection algorithm that is applicable when _____
a) all resources have a single instance
b) all resources have multiple instances
c) all resources have a single 7 multiple instances
d) all of the mentioned

Answer: a
Explanation: None.

147. An edge from process Pi to Pj in a wait for graph indicates that _____
a) Pi is waiting for Pj to release a resource that Pi needs
b) Pj is waiting for Pi to release a resource that Pj needs
c) Pi is waiting for Pj to leave the system
d) Pj is waiting for Pi to leave the system
View Answer

Answer: a
Explanation: None.

148. If the wait for graph contains a cycle _____
a) then a deadlock does not exist
b) then a deadlock exists
c) then the system is in a safe state
d) either deadlock exists or system is in a safe state

Answer: b
Explanation: None.

149. If deadlocks occur frequently, the detection algorithm must be invoked _____
a) rarely
b) frequently
c) rarely & frequently
d) none of the mentioned

Answer: b
Explanation: None.

150. What is the disadvantage of invoking the detection algorithm for every request?
a) overhead of the detection algorithm due to consumption of memory
b) excessive time consumed in the request to be allocated memory
c) considerable overhead in computation time
d) all of the mentioned

Answer: c
Explanation: None.

151. A deadlock can be broken by _____
a) abort one or more processes to break the circular wait
b) abort all the process in the system
c) preempt all resources from all processes
d) none of the mentioned

Answer: a
Explanation: None.

152. The two ways of aborting processes and eliminating deadlocks are _____
a) Abort all deadlocked processes
b) Abort all processes
c) Abort one process at a time until the deadlock cycle is eliminated
d) All of the mentioned

Answer: c
Explanation: None.

DIWAKAR EDUCATION HUB

153. Those processes should be aborted on occurrence of a deadlock, the termination of which?
a) is more time consuming
b) incurs minimum cost
c) safety is not hampered
d) all of the mentioned

Answer: b
Explanation: None.

154. The process to be aborted is chosen on the basis of the following factors?
a) priority of the process
b) process is interactive or batch
c) how long the process has computed
d) all of the mentioned

Answer: d
Explanation: None.

155. Cost factors for process termination include _____
a) Number of resources the deadlock process is not holding
b) CPU utilization at the time of deadlock
c) Amount of time a deadlocked process has thus far consumed during its execution
d) All of the mentioned

Answer: c
Explanation: None.

156. If we preempt a resource from a process, the process cannot continue with its normal execution and it must be _____
a) aborted
b) rolled back
c) terminated
d) queued

Answer: b
Explanation: None.

157. To _____ to a safe state, the system needs to keep more information about the states of processes.
a) abort the process
b) roll back the process
c) queue the process
d) none of the mentioned

Answer: b
Explanation: None.

158. If the resources are always preempted from the same process _____ can occur.
a) deadlock
b) system crash
c) aging
d) starvation

Answer: d
Explanation: None.

159. What is the solution to starvation?
a) the number of rollbacks must be included in the cost factor
b) the number of resources must be included in resource preemption
c) resource preemption be done instead
d) all of the mentioned

Answer: a
Explanation: None.

160. CPU fetches the instruction from memory according to the value of _____
a) program counter
b) status register

DIWAKAR EDUCATION HUB

c) instruction register
d) program status word

Answer: a
Explanation: None.

161. A memory buffer used to accommodate a speed differential is called _____
a) stack pointer
b) cache
c) accumulator
d) disk buffer

Answer: b
Explanation: None.

162. Which one of the following is the address generated by CPU?
a) physical address
b) absolute address
c) logical address
d) none of the mentioned

Answer: c
Explanation: None.

163. Run time mapping from virtual to physical address is done by _____
a) Memory management unit
b) CPU
c) PCI
d) None of the mentioned

Answer: a
Explanation: None.

164. Memory management technique in which system stores and retrieves data from secondary storage for use in main memory is called?
a) fragmentation
b) paging
c) mapping
d) none of the mentioned

Answer: b
Explanation: None.

165. The address of a page table in memory is pointed by _____
a) stack pointer
b) page table base register
c) page register
d) program counter

Answer: b
Explanation: None.

166. Program always deals with _____
a) logical address
b) absolute address
c) physical address
d) relative address

Answer: a
Explanation: None.

167. The page table contains _____
a) base address of each page in physical memory
b) page offset
c) page size
d) none of the mentioned

Answer: a
Explanation: None.

168. What is compaction?
a) a technique for overcoming internal fragmentation

DIWAKAR EDUCATION HUB

b) a paging technique
c) a technique for overcoming external fragmentation
d) a technique for overcoming fatal error

Answer: c
Explanation: None.

169. Operating System maintains the page table for _____
a) each process
b) each thread
c) each instruction
d) each address

Answer: a
Explanation: None.

170. The main memory accommodates _____
a) operating system
b) cpu
c) user processes
d) all of the mentioned

Answer: a
Explanation: None.

171. What is the operating system?
a) in the low memory
b) in the high memory
c) either low or high memory (depending on the location of interrupt vector)
d) none of the mentioned

Answer: c
Explanation: None.

172. In contiguous memory allocation _____
a) each process is contained in a single contiguous section of memory
b) all processes are contained in a single contiguous section of memory
c) the memory space is contiguous
d) none of the mentioned

Answer: a
Explanation: None.

173. The relocation register helps in _____
a) providing more address space to processes
b) a different address space to processes
c) to protect the address spaces of processes
d) none of the mentioned

Answer: c
Explanation: None.

174. With relocation and limit registers, each logical address must be _____ the limit register.
a) less than
b) equal to
c) greater than
d) none of the mentioned

Answer: a
Explanation: None.

175. The operating system and the other processes are protected from being modified by an already running process because _____
a) they are in different memory spaces
b) they are in different logical addresses
c) they have a protection algorithm
d) every address generated by the CPU is

DIWAKAR EDUCATION HUB

being checked against the relocation and limit registers

Answer: d
Explanation: None.

176. In internal fragmentation, memory is internal to a partition and _____
a) is being used
b) is not being used
c) is always used
d) none of the mentioned

Answer: b
Explanation: None.

177. A solution to the problem of external fragmentation is _____
a) compaction
b) larger memory space
c) smaller memory space
d) none of the mentioned

Answer: a
Explanation: None.

178. Another solution to the problem of external fragmentation problem is to _____
a) permit the logical address space of a process to be noncontiguous
b) permit smaller processes to be allocated memory at last
c) permit larger processes to be allocated memory at last
d) all of the mentioned

Answer: a
Explanation: None.

179. If relocation is static and is done at assembly or load time, compaction _____
a) cannot be done
b) must be done
c) must not be done
d) can be done

Answer: a
Explanation: None.

180. The disadvantage of moving all process to one end of memory and all holes to the other direction, producing one large hole of available memory is _____
a) the cost incurred
b) the memory used
c) the CPU used
d) all of the mentioned

Answer: a
Explanation: None.

181. _____ is generally faster than _____ and _____
a) first fit, best fit, worst fit
b) best fit, first fit, worst fit
c) worst fit, best fit, first fit
d) none of the mentioned

Answer: a
Explanation: None.

182. Physical memory is broken into fixed-sized blocks called _____
a) frames
b) pages
c) backing store
d) none of the mentioned

DIWAKAR EDUCATION HUB

Answer: a
Explanation: None.

183. Logical memory is broken into blocks of the same size called _____
a) frames
b) pages
c) backing store
d) none of the mentioned

Answer: b
Explanation: None.

184. Every address generated by the CPU is divided into two parts. They are _____
a) frame bit & page number
b) page number & page offset
c) page offset & frame bit
d) frame offset & page offset

Answer: b
Explanation: None.

185. The _____ is used as an index into the page table.
a) frame bit
b) page number
c) page offset
d) frame offset

Answer: b
Explanation: None.

186. The _____ table contains the base address of each page in physical memory.
a) process
b) memory
c) page
d) frame

Answer: c
Explanation: None.

187. The size of a page is typically _____
a) varied
b) power of 2
c) power of 4
d) none of the mentioned

Answer: b
Explanation: None.

188. Each entry in a translation lookaside buffer (TLB) consists of _____
a) key
b) value
c) bit value
d) constant

Answer: a
Explanation: None.

189. If a page number is not found in the TLB, then it is known as a _____
a) TLB miss
b) Buffer miss
c) TLB hit
d) All of the mentioned

Answer: a
Explanation: None.

190. An _____ uniquely identifies processes and is used to provide address space protection for that process.
a) address space locator
b) address space identifier
c) address process identifier
d) none of the mentioned

DIWAKAR EDUCATION HUB

Answer: b
Explanation: None.

191. The percentage of times a page number is found in the TLB is known as _____
a) miss ratio
b) hit ratio
c) miss percent
d) none of the mentioned

Answer: b
Explanation: None.

192. Memory protection in a paged environment is accomplished by _____
a) protection algorithm with each page
b) restricted access rights to users
c) restriction on page visibility
d) protection bit with each page

Answer: d
Explanation: None.

193. When the valid – invalid bit is set to valid, it means that the associated page _____
a) is in the TLB
b) has data in it
c) is in the process's logical address space
d) is the system's physical address space

Answer: c
Explanation: None.

194. In segmentation, each address is specified by _____
a) a segment number & offset
b) an offset & value
c) a value & segment number
d) a key & value

Answer: a
Explanation: None.

195. In paging the user provides only _____ which is partitioned by the hardware into _____ and _____
a) one address, page number, offset
b) one offset, page number, address
c) page number, offset, address
d) none of the mentioned

Answer: a
Explanation: None.

196. Each entry in a segment table has a _____
a) segment base
b) segment peak
c) segment value
d) none of the mentioned

Answer: a
Explanation: None.

197. The segment base contains the _____
a) starting logical address of the process
b) starting physical address of the segment in memory
c) segment length
d) none of the mentioned

Answer: b
Explanation: None.

198. The segment limit contains the _____
a) starting logical address of the process

DIWAKAR EDUCATION HUB

b) starting physical address of the segment in memory
c) segment length
d) none of the mentioned

Answer: c
Explanation: None.

199. The offset ‘d’ of the logical address must be _____
a) greater than segment limit
b) between 0 and segment limit
c) between 0 and the segment number
d) greater than the segment number

Answer: b
Explanation: None.

200. If the offset is legal _____
a) it is used as a physical memory address itself
b) it is subtracted from the segment base to produce the physical memory address
c) it is added to the segment base to produce the physical memory address
d) none of the mentioned

Answer: a
Explanation: None.

201. When the entries in the segment tables of two different processes point to the same physical location _____
a) the segments are invalid
b) the processes get blocked
c) segments are shared
d) all of the mentioned

Answer: c
Explanation: None.

202. The protection bit is 0/1 based on _____
a) write only
b) read only
c) read – write
d) none of the mentioned

Answer: c
Explanation: None.

203. If there are 32 segments, each of size 1Kb, then the logical address should have _____
a) 13 bits
b) 14 bits
c) 15 bits
d) 16 bits

Answer: a
Explanation: To specify a particular segment, 5 bits are required. To select a particular byte after selecting a page, 10 more bits are required. Hence 15 bits are required.

204. If one or more devices use a common set of wires to communicate with the computer system, the connection is called _____
a) CPU
b) Monitor
c) Wirefull
d) Bus

Answer: d
Explanation: None.

205. A ____ a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.
a) port

DIWAKAR EDUCATION HUB

b) node
c) bus
d) none of the mentioned

Answer: c
Explanation: None.

206. When device A has a cable that plugs into device B, and device B has a cable that plugs into device C and device C plugs into a port on the computer, this arrangement is called a _____
a) port
b) daisy chain
c) bus
d) cable

Answer: b
Explanation: None.

207. The _____ present a uniform device-access interface to the I/O subsystem, much as system calls provide a standard interface between the application and the operating system.
a) Devices
b) Buses
c) Device drivers
d) I/O systems

Answer: c
Explanation: None.

208. A _____ is a collection of electronics that can operate a port, a bus, or a device.
a) controller
b) driver
c) host
d) bus

Answer: a
Explanation: None.

209. An I/O port typically consists of four registers status, control, _____ and _____ registers.
a) system in, system out
b) data in, data out
c) flow in, flow out
d) input, output

Answer: b
Explanation: None.

210. The _____ register is read by the host to get input.
a) flow in
b) flow out
c) data in
d) data out

Answer: c
Explanation: None.

211. The _____ register is written by the host to send output.
a) status
b) control
c) data in
d) data out

Answer: d
Explanation: None.

212. The hardware mechanism that allows a device to notify the CPU is called _____
a) polling
b) interrupt
c) driver
d) controlling

DIWAKAR EDUCATION HUB

Answer: b
Explanation: None.

213. The CPU hardware has a wire called _____ that the CPU senses after executing every instruction.
a) interrupt request line
b) interrupt bus
c) interrupt receive line
d) interrupt sense line

Answer: a
Explanation: None.

214. In real time operating system _____
a) all processes have the same priority
b) a task must be serviced by its deadline period
c) process scheduling can be done only once
d) kernel is not required

Answer: b
Explanation: None.

215. For real time operating systems, interrupt latency should be _____
a) minimal
b) maximum
c) zero
d) dependent on the scheduling

Answer: a
Explanation: Interrupt latency is the time duration between the generation of interrupt and execution of its service.

216. In rate monotonic scheduling _____
a) shorter duration job has higher priority
b) longer duration job has higher priority
c) priority does not depend on the duration of the job
d) none of the mentioned

Answer: a
Explanation: None.

217. In which scheduling certain amount of CPU time is allocated to each process?
a) earliest deadline first scheduling
b) proportional share scheduling
c) equal share scheduling
d) none of the mentioned

Answer: b
Explanation: None.

218. The problem of priority inversion can be solved by _____
a) priority inheritance protocol
b) priority inversion protocol
c) both priority inheritance and inversion protocol
d) none of the mentioned

Answer: a
Explanation: None.

219. Earliest deadline first algorithm assigns priorities according to _____
a) periods
b) deadlines
c) burst times
d) none of the mentioned

Answer: b
Explanation: None.

220. A process P1 has a period of 50 and a CPU burst of t1 = 25, P2 has a period of 80 and a CPU burst of 35. The total CPU

DIWAKAR EDUCATION HUB

utilization is _____
a) 0.90
b) 0.74
c) 0.94
d) 0.80

Answer: c
Explanation: None.

221. A process P1 has a period of 50 and a CPU burst of t1 = 25, P2 has a period of 80 and a CPU burst of 35., the priorities of P1 and P2 are?
a) remain the same throughout
b) keep varying from time to time
c) may or may not be change
d) none of the mentioned

Answer: b
Explanation: None.

222. A process P1 has a period of 50 and a CPU burst of t1 = 25, P2 has a period of 80 and a CPU burst of 35., can the two processes be scheduled using the EDF algorithm without missing their respective deadlines?
a) Yes
b) No
c) Maybe
d) None of the mentioned

Answer: a
Explanation: None.

223. Using EDF algorithm practically, it is impossible to achieve 100 percent utilization due to _____
a) the cost of context switching
b) interrupt handling
c) power consumption
d) all of the mentioned

Answer: a
Explanation: None.

224. T shares of time are allocated among all processes out of N shares in _____ scheduling algorithm.
a) rate monotonic
b) proportional share
c) earliest deadline first
d) none of the mentioned

Answer: b
Explanation: None.

225. If there are a total of T = 100 shares to be divided among three processes, A, B and C. A is assigned 50 shares, B is assigned 15 shares and C is assigned 20 shares.
A will have _____ percent of the total processor time.
a) 20
b) 15
c) 50
d) none of the mentioned

Answer: c
Explanation: None.

226. If there are a total of T = 100 shares to be divided among three processes, A, B and C. A is assigned 50 shares, B is assigned 15 shares and C is assigned 20 shares.
B will have _____ percent of the total processor time.
a) 20
b) 15
c) 50
d) none of the mentioned

DIWAKAR EDUCATION HUB

Answer: b
Explanation: None.

227. If there are a total of T = 100 shares to be divided among three processes, A, B and C. A is assigned 50 shares, B is assigned 15 shares and C is assigned 20 shares.
C will have _____ percent of the total processor time.
a) 20
b) 15
c) 50
d) none of the mentioned

Answer: a
Explanation: None.

228. If there are a total of T = 100 shares to be divided among three processes, A, B and C. A is assigned 50 shares, B is assigned 15 shares and C is assigned 20 shares.
If a new process D requested 30 shares, the admission controller would _____
a) allocate 30 shares to it
b) deny entry to D in the system
c) all of the mentioned
d) none of the mentioned

Answer: b
Explanation: None.

229. To schedule the processes, they are considered _____
a) infinitely long
b) periodic
c) heavy weight
d) light weight

Answer: b
Explanation: None.

230. If the period of a process is 'p', then what is the rate of the task?
a) p^2
b) 2*p
c) 1/p
d) p

Answer: c
Explanation: None.

231. The scheduler admits a process using _____
a) two phase locking protocol
b) admission control algorithm
c) busy wait polling
d) none of the mentioned

Answer: c
Explanation: None.

232. The _____ scheduling algorithm schedules periodic tasks using a static priority policy with preemption.
a) earliest deadline first
b) rate monotonic
c) first cum first served
d) priority

Answer: b
Explanation: None.

233. Rate monotonic scheduling assumes that the _____
a) processing time of a periodic process is same for each CPU burst
b) processing time of a periodic process is different for each CPU burst
c) periods of all processes is the same
d) none of the mentioned

DIWAKAR EDUCATION HUB

Answer: a
Explanation: None.

234. The _____ can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted.
a) nonmaskable interrupt
b) blocked interrupt
c) maskable interrupt
d) none of the mentioned

Answer: c
Explanation: None.

235. The _____ is used by device controllers to request service.
a) nonmaskable interrupt
b) blocked interrupt
c) maskable interrupt
d) none of the mentioned

Answer: c
Explanation: None.

236. The interrupt vector contains _____
a) the interrupts
b) the memory addresses of specialized interrupt handlers
c) the identifiers of interrupts
d) the device addresses

Answer: b
Explanation: None.

237. Division by zero, accessing a protected or non existent memory address, or attempting to execute a privileged instruction from user mode are all categorized as _____
a) errors
b) exceptions
c) interrupt handlers
d) all of the mentioned

Answer: b
Explanation: None.

238. For large data transfers, _____ is used.
a) dma
b) programmed I/O
c) controller register
d) none of the mentioned

Answer: a
Explanation: None.

239. Buffering is done to _____
a) cope with device speed mismatch
b) cope with device transfer size mismatch
c) maintain copy semantics
d) all of the mentioned

Answer: d
Explanation: None.

240. Caching is _____ spooling.
a) same as
b) not the same as
c) all of the mentioned
d) none of the mentioned

Answer: b
Explanation: None.

241. Caching _____
a) holds a copy of the data
b) is fast memory
c) holds the only copy of the data
d) holds output for a device

DIWAKAR EDUCATION HUB

Answer: a
Explanation: None.

242. Spooling _____
a) holds a copy of the data
b) is fast memory
c) holds the only copy of the data
d) holds output for a device

Answer: c
Explanation: None.

243. The _____ keeps state information about the use of I/O components.
a) CPU
b) OS
c) kernel
d) shell

Answer: c
Explanation: None.

244. The kernel data structures include _____
a) process table
b) open file table
c) close file table
d) all of the mentioned

Answer: b
Explanation: None.

245. Windows NT uses a _____ implementation for I/O.
a) message – passing
b) draft – passing
c) secondary memory
d) cache

Answer: a
Explanation: None.

246. A _____ is a full duplex connection between a device driver and a user level process.
a) Bus
b) I/O operation
c) Stream
d) Flow

Answer: c
Explanation: None.

247. The process of dividing a disk into sectors that the disk controller can read and write, before a disk can store data is known as _____
a) partitioning
b) swap space creation
c) low-level formatting
d) none of the mentioned

Answer: c
Explanation: None.

249. The header and trailer of a sector contain information used by the disk controller such as _____ and _____
a) main section & disk identifier
b) error correcting codes (ECC) & sector number
c) sector number & main section
d) disk identifier & sector number

Answer: b
Explanation: None.

250. The two steps the operating system takes to use a disk to hold its files are _____ and _____
a) partitioning & logical formatting
b) swap space creation & caching

DIWAKAR EDUCATION HUB

c) caching & logical formatting
d) logical formatting & swap space creation

Answer: a
Explanation: None.

251. The _____ program initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system.
a) main
b) bootloader
c) bootstrap
d) rom

Answer: c
Explanation: None.

252. For most computers, the bootstrap is stored in _____
a) RAM
b) ROM
c) Cache
d) Tertiary storage

Answer: b
Explanation: None.

253. A disk that has a boot partition is called a _____
a) start disk
b) end disk
c) boot disk
d) all of the mentioned

Answer: c
Explanation: None.

254. In _____ information is recorded magnetically on platters.
a) magnetic disks
b) electrical disks
c) assemblies
d) cylinders

Answer: a
Explanation: None.

255. The heads of the magnetic disk are attached to a _____ that moves all the heads as a unit.
a) spindle
b) disk arm
c) track
d) none of the mentioned

Answer: b
Explanation: None.

256. The set of tracks that are at one arm position make up a _____
a) magnetic disks
b) electrical disks
c) assemblies
d) cylinders

Answer: d
Explanation: None.

257. The time taken to move the disk arm to the desired cylinder is called the _____
a) positioning time
b) random access time
c) seek time
d) rotational latency

Answer: c
Explanation: None.

DIWAKAR EDUCATION HUB

258. Whenever a process needs I/O to or from a disk it issues a _____
a) system call to the CPU
b) system call to the operating system
c) a special procedure
d) all of the mentioned

Answer: b
Explanation: None.

259. If a process needs I/O to or from a disk, and if the drive or controller is busy then _____
a) the request will be placed in the queue of pending requests for that drive
b) the request will not be processed and will be ignored completely
c) the request will be not be placed
d) none of the mentioned

Answer: a
Explanation: None.

260. Consider a disk queue with requests for I/O to blocks on cylinders.
98 183 37 122 14 124 65 67
Considering FCFS (first cum first served) scheduling, the total number of head movements is, if the disk head is initially at 53 is?
a) 600
b) 620
c) 630
d) 640

Answer: d
Explanation: None.

261. Consider a disk queue with requests for I/O to blocks on cylinders.
98 183 37 122 14 124 65 67
Considering SSTF (shortest seek time first) scheduling, the total number of head movements is, if the disk head is initially at 53 is?
a) 224
b) 236
c) 245
d) 240

Answer: b
Explanation: None.

262. Random access in magnetic tapes is _____ compared to magnetic disks.
a) fast
b) very fast
c) slow
d) very slow

Answer: d
Explanation: None.

263. I/O hardware contains _____
a) Bus
b) Controller
c) I/O port and its registers
d) All of the mentioned

Answer: d
Explanation: None.

264. The data-in register of I/O port is _____
a) Read by host to get input
b) Read by controller to get input
c) Written by host to send output
d) Written by host to start a command

Answer: a
Explanation: None.

DIWAKAR EDUCATION HUB

265. The host sets _____ bit when a command is available for the controller to execute.
a) write
b) status
c) command-ready
d) control

Answer: c
Explanation: None.

266. When hardware is accessed by reading and writing to the specific memory locations, then it is called _____
a) port-mapped I/O
b) controller-mapped I/O
c) bus-mapped I/O
d) none of the mentioned

Answer: d
Explanation: It is called memory-mapped I/O.

267. Device drivers are implemented to interface _____
a) character devices
b) block devices
c) network devices
d) all of the mentioned

Answer: d
Explanation: None.

268. Which hardware triggers some operation after certain programmed count?
a) programmable interval timer
b) interrupt timer
c) programmable timer
d) none of the mentioned

Answer: a
Explanation: None.

269. The device-status table contains _____
a) each I/O device type
b) each I/O device address
c) each I/O device state
d) all of the mentioned

Answer: d
Explanation: None.

270. The model in which one kernel thread is mapped to many user-level threads is called _____
a) Many to One model
b) One to Many model
c) Many to Many model
d) One to One model

Answer: a
Explanation: None.

271. The model in which one user-level thread is mapped to many kernel level threads is called _____
a) Many to One model
b) One to Many model
c) Many to Many model
d) One to One model

Answer: b
Explanation: None.

272. In the Many to One model, if a thread makes a blocking system call _____
a) the entire process will be blocked
b) a part of the process will stay blocked, with the rest running

DIWAKAR EDUCATION HUB

c) the entire process will run
d) none of the mentioned

Answer: a
Explanation: None.

273. In the Many to One model, multiple threads are unable to run in parallel on multiprocessors because of _____
a) only one thread can access the kernel at a time
b) many user threads have access to just one kernel thread
c) there is only one kernel thread
d) none of the mentioned

Answer: a
Explanation: None.

273. The One to One model allows _____
a) increased concurrency
b) decreased concurrency
c) increased or decreased concurrency
d) concurrency equivalent to other models

Answer: a
Explanation: None.

274. In the One to One model when a thread makes a blocking system call _____
a) other threads are strictly prohibited from running
b) other threads are allowed to run
c) other threads only from other processes are allowed to run
d) none of the mentioned

Answer: b
Explanation: None.

275. When is the Many to One model at an advantage?
a) When the program does not need multithreading
b) When the program has to be multi-threaded
c) When there is a single processor
d) None of the mentioned

Answer: a
Explanation: None.

276. In the Many to Many model true concurrency cannot be gained because _____
a) the kernel can schedule only one thread at a time
b) there are too many threads to handle
c) it is hard to map threads with each other
d) none of the mentioned

Answer: a
Explanation: None.

277. In the Many to Many models when a thread performs a blocking system call _____
a) other threads are strictly prohibited from running
b) other threads are allowed to run
c) other threads only from other processes are allowed to run
d) none of the mentioned

Answer: b
Explanation: None.

278. 1. Thread pools are useful when _____
a) when we need to limit the number of threads running in the application at the same time

DIWAKAR EDUCATION HUB

b) when we need to limit the number of threads running in the application as a whole
c) when we need to arrange the ordering of threads
d) none of the mentioned

Answer: a
Explanation: None.

279. Instead of starting a new thread for every task to execute concurrently, the task can be passed to a _____
a) process
b) thread pool
c) thread queue
d) none of the mentioned

Answer: b
Explanation: None.

280. Each connection arriving at multi threaded servers via network is generally _____
a) is directly put into the blocking queue
b) is wrapped as a task and passed on to a thread pool
c) is kept in a normal queue and then sent to the blocking queue from where it is dequeued
d) none of the mentioned

Answer: b
Explanation: None.

281. What are the parts of network structure?
a) Workstation
b) Gateway
c) Laptop
d) All of the mentioned

Answer: d
Explanation: None.

282. What is a valid network topology?
a) Multiaccess bus
b) Ring
c) Star
d) All of the mentioned

Answer: d
Explanation: None.

283. What are sites in network topology compared?
a) Basic cost
b) Communication cost
c) Reliability
d) All of the mentioned

Answer: d
Explanation: None.

284. Which design features of a communication network are important?
a) Naming and name resolution
b) Routing strategies
c) Connection strategies
d) All of the mentioned

Answer: d
Explanation: None.

285. What are the characteristics of Naming and Name resolution?
a) name systems in the network
b) address messages with the process-id
c) virtual circuit
d) message switching

Answer: b
Explanation: None.

DIWAKAR EDUCATION HUB

286. What are routing strategies which is not used in distributed systems?
a) Fixed routing
b) Token routing
c) Virtual circuit
d) Dynamic routing

Answer: c
Explanation: None.

287. What are the connection strategies not used in distributed systems?
a) Circuit switching
b) Message switching
c) Token switching
d) Packet switching

Answer: c
Explanation: None.

288. How is are collisions avoided in network?
a) Carrier sense with multiple access (CSMA); collision detection (CD)
b) Carrier sense multiple access with collision avoidance
c) Message slots
d) All of the mentioned

Answer: d
Explanation: None.

289. In distributed system, each processor has its own _____
a) local memory
b) clock
c) both local memory and clock
d) none of the mentioned

Answer: c
Explanation: None.

290. If one site fails in distributed system then _____
a) the remaining sites can continue operating
b) all the sites will stop working
c) directly connected sites will stop working
d) none of the mentioned

Answer: a
Explanation: None.

291. Network operating system runs on _____
a) server
b) every system in the network
c) both server and every system in the network
d) none of the mentioned

Answer: a
Explanation: None.

292. Which technique is based on compile-time program transformation for accessing remote data in a distributed-memory parallel system?
a) cache coherence scheme
b) computation migration
c) remote procedure call
d) message passing

Answer: b
Explanation: None.

293. Logical extension of computation migration is _____
a) process migration
b) system migration
c) thread migration
d) data migration

DIWAKAR EDUCATION HUB

Answer: a
Explanation: None.

294. Processes on the remote systems are identified by _____
a) host ID
b) host name and identifier
c) identifier
d) process ID

Answer: b
Explanation: None.

295. Which routing technique is used in a distributed system?
a) fixed routing
b) virtual routing
c) dynamic routing
d) all of the mentioned

Answer: d
Explanation: None.

296. In distributed systems, link and site failure is detected by _____
a) polling
b) handshaking
c) token passing
d) none of the mentioned

Answer: b
Explanation: None.

297. What is not true about a distributed system?
a) It is a collection of processor
b) All processors are synchronized
c) They do not share memory
d) None of the mentioned

Answer: b
Explanation: None.

298. What are the characteristics of processor in distributed system?
a) They vary in size and function
b) They are same in size and function
c) They are manufactured with single purpose
d) They are real-time devices

Answer: a
Explanation: None.

299. What are the characteristics of a distributed file system?
a) Its users, servers and storage devices are dispersed
b) Service activity is not carried out across the network
c) They have single centralized data repository
d) There are multiple dependent storage devices

Answer: a
Explanation: None.

300. What is not a major reason for building distributed systems?
a) Resource sharing
b) Computation speedup
c) Reliability
d) Simplicity

Answer: d
Explanation: None.

301. What are the types of distributed operating system?
a) Network Operating system

DIWAKAR EDUCATION HUB

b) Zone based Operating system
c) Level based Operating system
d) All of the mentioned

Answer: a
Explanation: None.

302. What are characteristic of Network Operating Systems?
a) Users are aware of multiplicity of machines
b) They are transparent
c) They are simple to use
d) All of the mentioned

Answer: a
Explanation: None.

303. What are routing strategies which is not used in distributed systems?
a) Fixed routing
b) Token routing
c) Virtual circuit
d) Dynamic routing

Answer: c
Explanation: None.

304. What are the connection strategies not used in distributed systems?
a) Circuit switching
b) Message switching
c) Token switching
d) Packet switching

Answer: c
Explanation: None.

305. How is are collisions avoided in network?
a) Carrier sense with multiple access (CSMA); collision detection (CD)
b) Carrier sense multiple access with collision avoidance
c) Message slots
d) All of the mentioned

Answer: d
Explanation: None.

306. What is a common problem found in distributed system?
a) Process Synchronization
b) Communication synchronization
c) Deadlock problem
d) Power failure

Answer: c
Explanation: None.

307. How many layers does the Internet model ISO consist of?
a) Three
b) Five
c) Seven
d) Eight

Answer: c
Explanation: None.

308. Which layer is responsible for The process-to-process delivery?
a) Network
b) Transport
c) Application
d) Physical

Answer: b
Explanation: None.

309. Which layer is the layer closest to the transmission medium?

DIWAKAR EDUCATION HUB

a) Physical
b) Data link
c) Network
d) Transport

Answer: a
Explanation: None.

310. Header are _____ when data packet moves from upper to the lower layers?
a) Modified
b) Removed
c) Added
d) All of the mentioned

Answer: c
Explanation: None.

311. Which layer lies between the transport layer and data link layer?
a) Physical
b) Network
c) Application
d) Session

Answer: b
Explanation: None.

312. Which of the following is an application layer service?
a) Mail service
b) File transfer
c) Remote access
d) All of the mentioned

Answer: d
Explanation: None.

313. What are the different ways in which clients and servers are dispersed across machines?
a) Servers may not run on dedicated machines
b) Servers and clients can be on same machines
c) Distribution cannot be interposed between a OS and the file system
d) OS cannot be distributed with the file system a part of that distribution

Answer: b
Explanation: None.

314. What are not the characteristics of a DFS?
a) login transparency and access transparency
b) Files need not contain information about their physical location
c) No Multiplicity of users
d) No Multiplicity if files

Answer: c
Explanation: None.

315. What are characteristic of a DFS?
a) Fault tolerance
b) Scalability
c) Heterogeneity of the system
d) Upgradation

Answer: d
Explanation: None.

316. What are the different ways file accesses take place?
a) sequential access
b) direct access
c) indexed sequential access
d) all of the mentioned

DIWAKAR EDUCATION HUB

Answer: d
Explanation: None.

317. Which is not a major component of a file system?
a) Directory service
b) Authorization service
c) Shadow service
d) System service

Answer: c
Explanation: None.

318. What are the different ways mounting of the file system?
a) boot mounting
b) auto mounting
c) explicit mounting
d) all of the mentioned

Answer: d
Explanation: None.

319. Implementation of a stateless file server must not follow?
a) Idempotency requirement
b) Encryption of keys
c) File locking mechanism
d) Cache consistency

Answer: b
Explanation: None.

320. What are the advantages of file replication?
a) Improves availability & performance
b) Decreases performance
c) They are consistent
d) Improves speed

Answer: a
Explanation: None.

321. What are characteristic of NFS protocol?
a) Search for file within directory
b) Read a set of directory entries
c) Manipulate links and directories
d) All of the mentioned

Answer: d
Explanation: None.

322. The file once created can not be changed is called _____
a) immutable file
b) mutex file
c) mutable file
d) none of the mentioned

Answer: a
Explanation: None.

323. _____ of the distributed file system are dispersed among various machines of distributed system.
a) Clients
b) Servers
c) Storage devices
d) All of the mentioned

Answer: d
Explanation: None.

324. _____ is not possible in distributed file system.
a) File replication
b) Migration
c) Client interface
d) Remote access

DIWAKAR EDUCATION HUB

Answer: b
Explanation: None.

325. Which one of the following hides the location where in the network the file is stored?
a) transparent distributed file system
b) hidden distributed file system
c) escaped distribution file system
d) spy distributed file system

Answer: a
Explanation: None.

326. A process can be _____
a) single threaded
b) multithreaded
c) both single threaded and multithreaded
d) none of the mentioned

Answer: c
Explanation: None.

327. If one thread opens a file with read privileges then _____
a) other threads in the another process can also read from that file
b) other threads in the same process can also read from that file
c) any other thread can not read from that file
d) all of the mentioned

Answer: b
Explanation: None.

328. The time required to create a new thread in an existing process is _____
a) greater than the time required to create a new process
b) less than the time required to create a new process
c) equal to the time required to create a new process
d) none of the mentioned

Answer: b
Explanation: None.

329. When the event for which a thread is blocked occurs?
a) thread moves to the ready queue
b) thread remains blocked
c) thread completes
d) a new thread is provided

Answer: a
Explanation: None.

330. A thread is also called _____
a) Light Weight Process(LWP)
b) Heavy Weight Process(HWP)
c) Process
d) None of the mentioned

Answer: a
Explanation: None.

331. A thread shares its resources(like data section, code section, open files, signals) with _____
a) other process similar to the one that the thread belongs to
b) other threads that belong to similar processes
c) other threads that belong to the same process
d) all of the mentioned

Answer: c
Explanation: None.

DIWAKAR EDUCATION HUB

332. A heavy weight process _____
a) has multiple threads of execution
b) has a single thread of execution
c) can have multiple or a single thread for execution
d) none of the mentioned

Answer: b
Explanation: None.

333. A process having multiple threads of control implies _____
a) it can do more than one task at a time
b) it can do only one task at a time, but much faster
c) it has to use only one thread per process
d) none of the mentioned

Answer: a
Explanation: None.

334. Multithreading an interactive program will increase responsiveness to the user by _____
a) continuing to run even if a part of it is blocked
b) waiting for one part to finish before the other begins
c) asking the user to decide the order of multithreading
d) none of the mentioned

Answer: a
Explanation: None.

335. Resource sharing helps _____
a) share the memory and resources of the process to which the threads belong
b) an application have several different threads of activity all within the same address space
c) reduce the address space that a process could potentially use
d) all of the mentioned

Answer: d
Explanation: None.

336. Multithreading on a multi – CPU machine _____
a) decreases concurrency
b) increases concurrency
c) doesn't affect the concurrency
d) can increase or decrease the concurrency

Answer: b
Explanation: None.

337. The kernel is _____ of user threads.
a) a part of
b) the creator of
c) unaware of
d) aware of

Answer: c
Explanation: None.

338. _____ is a unique tag, usually a number identifies the file within the file system.
a) File identifier
b) File name
c) File type
d) None of the mentioned

Answer: a
Explanation: None.

339. To create a file _____
a) allocate the space in file system
b) make an entry for new file in directory
c) allocate the space in file system & make

DIWAKAR EDUCATION HUB

an entry for new file in directory
d) none of the mentioned

Answer: c
Explanation: None.

340. By using the specific system call, we can _____
a) open the file
b) read the file
c) write into the file
d) all of the mentioned

Answer: d
Explanation: None.

341. File type can be represented by _____
a) file name
b) file extension
c) file identifier
d) none of the mentioned

Answer: b
Explanation: None.

342. Which file is a sequence of bytes organized into blocks understandable by the system's linker?
a) object file
b) source file
c) executable file
d) text file

Answer: a
Explanation: None.

343. What is the mounting of file system?
a) crating of a filesystem
b) deleting a filesystem
c) attaching portion of the file system into a directory structure
d) removing the portion of the file system into a directory structure

Answer: c
Explanation: None.

344. Mapping of file is managed by _____
a) file metadata
b) page table
c) virtual memory
d) file system

Answer: a
Explanation: None.

345. Mapping of network file system protocol to local file system is done by _____
a) network file system
b) local file system
c) volume manager
d) remote mirror

Answer: a
Explanation: None.

346. Which one of the following explains the sequential file access method?
a) random access according to the given byte number
b) read bytes one at a time, in order
c) read/write sequentially by record
d) read/write randomly by record

Answer: b
Explanation: None.

347. When will file system fragmentation occur?

DIWAKAR EDUCATION HUB

a) unused space or single file are not contiguous
b) used space is not contiguous
c) unused space is non-contiguous
d) multiple files are non-contiguous

Answer: a
Explanation: None.

348. What is the mount point?
a) an empty directory at which the mounted file system will be attached
b) a location where every time file systems are mounted
c) is the time when the mounting is done
d) none of the mentioned

Answer: a
Explanation: None.

349. When a file system is mounted over a directory that is not empty then _____
a) the system may not allow the mount
b) the system must allow the mount
c) the system may allow the mount and the directory's existing files will then be made obscure
d) all of the mentioned

Answer: c
Explanation: None.

350. In UNIX, exactly which operations can be executed by group members and other users is definable by _____
a) the group's head
b) the file's owner
c) the file's permissions
d) all of the mentioned

Answer: b
Explanation: None.

351. A process _____ lower the priority of another process if both are owned by the same owner.
a) must
b) can
c) cannot
d) none of the mentioned

Answer: b
Explanation: None.

352. In distributed file system _____ directories are visible from the local machine.
a) protected
b) local
c) private
d) remote

Answer: d
Explanation: None.

353. In the world wide web, a ____ is needed to gain access to the remote files, and separate operations are used to transfer files.
a) laptop
b) plugin
c) browser
d) player

Answer: c
Explanation: None.

354. Anonymous access allows a user to transfer files _____
a) without having an account on the remote system

DIWAKAR EDUCATION HUB

b) only if he accesses the system with a guest account
c) only if he has an account on the remote system
d) none of the mentioned

Answer: a
Explanation: The world wide web uses anonymous file exchange almost exclusively.

355. The machine containing the files is the _____ and the machine wanting to access the files is the _____
a) master, slave
b) memory, user
c) server, client
d) none of the mentioned

Answer: c
Explanation: None.

356. Distributed naming services/Distributed information systems have been devised to _____
a) provide information about all the systems
b) provide unified access to the information needed for remote computing
c) provide unique names to all systems in a network
d) all of the mentioned

Answer: b
Explanation: None.

357. Domain name system provides _____
a) host-name-to-network-address translations for the entire internet
b) network-address-to-host-name translations for the entire internet
c) binary to hex translations for the entire internet
d) all of the mentioned

Answer: a
Explanation: None.

358. Reliability of files can be increased by _____
a) keeping the files safely in the memory
b) making a different partition for the files
c) by keeping them in external storage
d) by keeping duplicate copies of the file

Answer: d
Explanation: None.

359. Protection is only provided at the _____ level.
a) lower
b) central
c) higher
d) none of the mentioned

Answer: a
Explanation: None.

360. What is the main problem with access control lists?
a) their maintenance
b) their length
c) their permissions
d) all of the mentioned

Answer: b
Explanation: None.

361. Many systems recognize three classifications of users in connection with each file (to condense the access control

DIWAKAR EDUCATION HUB

list).
a) Owner
b) Group
c) Universe
d) All of the mentioned

Answer: d
Explanation: None.

362. The three major methods of allocating disk space that are in wide use are _____
a) contiguous
b) linked
c) indexed
d) all of the mentioned

Answer: d
Explanation: None.

363. In contiguous allocation _____
a) each file must occupy a set of contiguous blocks on the disk
b) each file is a linked list of disk blocks
c) all the pointers to scattered blocks are placed together in one location
d) none of the mentioned

Answer: a
Explanation: None.

364. In linked allocation _____
a) each file must occupy a set of contiguous blocks on the disk
b) each file is a linked list of disk blocks
c) all the pointers to scattered blocks are placed together in one location
d) none of the mentioned

Answer: b
Explanation: None.

365. In indexed allocation _____
a) each file must occupy a set of contiguous blocks on the disk
b) each file is a linked list of disk blocks
c) all the pointers to scattered blocks are placed together in one location
d) none of the mentioned

Answer: c
Explanation: None.

366. On systems where there are multiple operating system, the decision to load a particular one is done by _____
a) boot loader
b) bootstrap
c) process control block
d) file control block

Answer: a
Explanation: None.

367. The VFS (virtual file system) activates file system specific operations to handle local requests according to their _____
a) size
b) commands
c) timings
d) file system types

Answer: d
Explanation: None.

368. A device driver can be thought of like a translator. Its input consists of _____ commands and output consists of _____ instructions.
a) high level, low level

DIWAKAR EDUCATION HUB

b) low level, high level
c) complex, simple
d) low level, complex

Answer: a
Explanation: None.

369. The file organization module knows about _____
a) files
b) logical blocks of files
c) physical blocks of files
d) all of the mentioned

Answer: d
Explanation: None.

370. Metadata includes _____
a) all of the file system structure
b) contents of files
c) both file system structure and contents of files
d) none of the mentioned

Answer: c
Explanation: None.

371. For each file there exists a _____ that contains information about the file, including ownership, permissions and location of the file contents.
a) metadata
b) file control block
c) process control block
d) all of the mentioned

Answer: b
Explanation: None.

372. For processes to request access to file contents, they need _____
a) to run a seperate program
b) special interrupts
c) to implement the open and close system calls
d) none of the mentioned

Answer: c
Explanation: None.

373. A better way of contiguous allocation to extend the file size is _____
a) adding an extent (another chunk of contiguous space)
b) adding an index table to the first contiguous block
c) adding pointers into the first contiguous block
d) none of the mentioned

Answer: a
Explanation: None.

374. If the extents are too large, then what is the problem that comes in?
a) internal fragmentation
b) external fragmentation
c) starvation
d) all of the mentioned

Answer: a
Explanation: None.

375. The FAT is used much as a _____
a) stack
b) linked list
c) data
d) pointer

DIWAKAR EDUCATION HUB

Answer: b
Explanation: None.

376. _____ tend to represent a major bottleneck in system performance.
a) CPUs
b) Disks
c) Programs
d) I/O

Answer: b
Explanation: None.

377. In UNIX, even an 'empty' disk has a percentage of its space lost to _____
a) programs
b) inodes
c) virtual memory
d) stacks

Answer: b
Explanation: None.

378. Some directory information is kept in main memory or cache to _____
a) fill up the cache
b) increase free space in secondary storage
c) decrease free space in secondary storage
d) speed up access

Answer: d
Explanation: None.

379. A systems program such as fsck in _____ is a consistency checker.
a) UNIX
b) Windows
c) Macintosh
d) Solaris

Answer: a
Explanation: None.

380. A consistency checker _____ and tries to fix any inconsistencies it finds.
a) compares the data in the secondary storage with the data in the cache
b) compares the data in the directory structure with the data blocks on disk
c) compares the system generated output and user required output
d) all of the mentioned

Answer: b
Explanation: None.

381. Each set of operations for performing a specific task is a _____
a) program
b) code
c) transaction
d) all of the mentioned

Answer: c
Explanation: None.

382. Once the changes are written to the log, they are considered to be _____
a) committed
b) aborted
c) completed
d) none of the mentioned

Answer: a
Explanation: None.

383. When an entire committed transaction is completed, _____
a) it is stored in the memory
b) it is removed from the log file

DIWAKAR EDUCATION HUB

c) it is redone
d) none of the mentioned

Answer: b
Explanation: None.

384. A machine in Network file system (NFS) can be _____
a) client
b) server
c) both client and server
d) neither client nor server

Answer: c
Explanation: None.

385. A _____ directory is mounted over a directory of a _____ file system.
a) local, remote
b) remote, local
c) local, local
d) none of the mentioned

Answer: d
Explanation: None.

386. What is Address Binding?
a) going to an address in memory
b) locating an address with the help of another address
c) binding two addresses together to form a new address in a different memory space
d) a mapping from one address space to another
View Answer

Answer: d
Explanation: None.

387. Binding of instructions and data to memory addresses can be done at _____
a) Compile time
b) Load time
c) Execution time
d) All of the mentioned

Answer: d
Explanation: None.

Explanation: None.

388. Which one of the following is a process that uses the spawn mechanism to revage the system performance?
a) worm
b) trojan
c) threat
d) virus

Answer: a
Explanation: None.

389. What is true regarding 'Fence'?
a) Its a method to confine users to one side of a boundary
b) It can protect Operating system from one user
c) It cannot protect users from each other
d) All of the mentioned

Answer: d
Explanation: None.

390. What is not true regarding 'Fence'?
a) It is implemented via hardware register
b) It doesn't protect users from each other
c) It good to protect OS from abusive users
d) Its implementation is unrestricted and can take any amount of space in Operating system.

DIWAKAR EDUCATION HUB

Answer: d
Explanation: None.

391. What is correct regarding 'relocation' w.r.t protecting memory?
a) It is a process of taking a program as if it began at address 0
b) It is a process of taking a program as if it began at address 0A
c) Fence cannot be used within relocation process
d) All of the mentioned

Answer: a
Explanation: None.

392. What are the incorrect methods of revocation of access rights?
a) Immediate/Delayed
b) Selective/General
c) Partial/total
d) Crucial

Answer: d
Explanation: None.

393. Why is it difficult to revoke capabilities?
a) They are too many
b) They are not defined precicely
c) They are distributed throughout the system
d) None of the mentioned

Answer: c
Explanation: None.

394. What is the reacquisition scheme to revoke capability?
a) When a process capability is revoked then it won't be able to reacquire it
b) Pointers are maintained for each object which can be used to revoke
c) Indirect pointing is done to revoke object's capabilities
d) Master key can be used compare and revoke.

Answer: a
Explanation: None.

395. What is false regarding Back-Pointers scheme to revoke capability?
a) List of pointers is maintained with each object
b) When revocation is required these pointers are followed
c) This scheme is not adopted in MULTICS system
d) These point to all capabilities associated with that object

Answer: c
Explanation: None.

396. From the following, which is not a common file permission?
a) Write
b) Execute
c) Stop
d) Read

Answer: c
Explanation: None.

397. Which of the following is a good practice?
a) Give full permission for remote transferring
b) Grant read only permission
c) Grant limited permission to specified account

DIWAKAR EDUCATION HUB

d) Give both read and write permission but not execute

Answer: c
Explanation: Limited access is a key method to circumvent unauthorized access and exploits.

398. What is breach of availability?
a) This type of violation involves unauthorized reading of data
b) This violation involves unauthorized modification of data
c) This violation involves unauthorized destruction of data
d) This violation involves unauthorized use of resources

Answer: c
Explanation: None.

399. What is Trojan horse?
a) It is a useful way to encrypt password
b) It is a user which steals valuable information
c) It is a rogue program which tricks users
d) It's a brute force attack algorithm

Answer: c
Explanation: None.

400. What is trap door?
a) IT is trap door in WarGames
b) It is a hole in software left by designer
c) It is a Trojan horse
d) It is a virus which traps and locks user terminal

Answer: b
Explanation: None.

401. Which mechanism is used by worm process?
a) Trap door
b) Fake process
c) Spawn Process
d) VAX process

Answer: c
Explanation: None.

402. Which of the following is not a characteristic of a virus?
a) Virus destroy and modify user data
b) Virus is a standalone program
c) Virus is a code embedded in a legitimate program
d) Virus cannot be detected

Answer: d
Explanation: Virus can be detected by having an antivirus program.

403. What is not an important part of security protection?
a) Large amount of RAM to support antivirus
b) Strong passwords
c) Audit log periodically
d) Scan for unauthorized programs in system directories

Answer: a
Explanation: RAM has no effect on security of a system. System's protection remains unchanged in increasing or decreasing amount of RAM.

404. What is used to protect network from outside internet access?
a) A trusted antivirus
b) 24 hours scanning for virus

DIWAKAR EDUCATION HUB

c) Firewall to separate trusted and untrusted network
d) Deny users access to websites which can potentially cause security leak

Answer: c
Explanation: Firewall create a protective barrier to secure internal network. An antivirus can only detect harmful viruses but cannot stop illegal access by remote attacker.

405. What is the best practice in the firewall domain environment?
a) Create two domain trusted and untrusted domain
b) Create strong policy in firewall to support different types of users
c) Create a Demilitarized zone
d) Create two DMZ zones with one untrusted domain

Answer: c
Explanation: All live servers or workstations are kept in a separate zone than inside and outside to enhance protection.

406. Which direction access cannot happen using DMZ zone by default?
a) Company computer to DMZ
b) Internet to DMZ
c) Internet to company computer
d) Company computer to internet

Answer: c
Explanation: Connection from internet is never allowed to directly access internal PCs but is routed through DMZ zone to prevent atta

407. A process is thrashing if _____
a) it spends a lot of time executing, rather than paging
b) it spends a lot of time paging than executing
c) it has no memory allocated to it
d) none of the mentioned

Answer: b
Explanation: None.

408. Thrashing _____ the CPU utilization.
a) increases
b) keeps constant
c) decreases
d) none of the mentioned
View Answer

Answer: c
Explanation: None.

409. RAID level 3 supports a lower number of I/Os per second, because _____
a) Every disk has to participate in every I/O request
b) Only one disk participates per I/O request
c) I/O cycle consumes a lot of CPU time
d) All of the mentioned

Answer: a
Explanation: None.

410. RAID level _____ is also known as block interleaved parity organisation and uses block level striping and keeps a parity block on a separate disk.
a) 1
b) 2

DIWAKAR EDUCATION HUB

c) 3
d) 4

Answer: d
Explanation: None.

411. A performance problem with _____ is the expense of computing and writing parity.
a) non-parity based RAID levels
b) parity based RAID levels
c) all RAID levels
d) none of the mentioned

Answer: b
Explanation: None.

412. In RAID level 4, one block read, accesses _____
a) only one disk
b) all disks simultaneously
c) all disks sequentially
d) none of the mentioned

Answer: a
Explanation: Other requests are allowed to be processed by other disks.

413. The overall I/O rate in RAID level 4 is _____
a) low
b) very low
c) high
d) none of the mentioned

Answer: c
Explanation: All disks can be read in parallel.

414. Linux uses a time-sharing algorithm _____
a) to pair preemptive scheduling between multiple processes
b) for tasks where absolute priorities are more important than fairness
c) all of the mentioned
d) none of the mentioned

Answer: a
Explanation: None.

415. The first linux kernel which supports the SMP hardware?
a) linux 0.1
b) linux 1.0
c) linux 1.2
d) linux 2.0

Answer: d
Explanation: None.

416. What is Linux?
a) single user, single tasking
b) single user, multitasking
c) multi user, single tasking
d) multi user, multitasking

Answer: d
Explanation: None.

417. Which one of the following is not a linux distribution?
a) debian
b) gentoo
c) open SUSE
d) multics

Answer: d
Explanation: None.

418. In distributed systems, a logical clock is associated with _____
a) each instruction

DIWAKAR EDUCATION HUB

b) each process
c) each register
d) none of the mentioned

Answer: b
Explanation: None.

419. If timestamps of two events are same, then the events are _____
a) concurrent
b) non-concurrent
c) monotonic
d) non-monotonic

Answer: a
Explanation: None.

420. If a process is executing in its critical section _____
a) any other process can also execute in its critical section
b) no other process can execute in its critical section
c) one more process can execute in its critical section
d) none of the mentioned

Answer: b
Explanation: None.

421. A process can enter into its critical section _____
a) anytime
b) when it receives a reply message from its parent process
c) when it receives a reply message from all other processes in the system
d) none of the mentioned

Answer: c
Explanation: None.

422. For proper synchronization in distributed systems _____
a) prevention from the deadlock is must
b) prevention from the starvation is must
c) prevention from the deadlock & starvation is must
d) none of the mentioned

Answer: c
Explanation: None.

DIWAKAR EDUCATION HUB