Unit-1

Operating System Overview:-

⊛Introduction and Definition:

Computer software can be divided into two main categories: application software and system software. Application software consists of the programs for performing tasks particular to the machines utilization. This software is designed to solve a particular problem for users. Spreadsheets, database systems, powerpoint etc. are its examples.

System software acts as an interface between the hardware of the computer and the application software that users need to run on the computer. Different versions of Windows, Linux etc are its example. The most important type of system software is Operating System (OS).

Defn → An Operating System (OS) is a collection of programs that acts as an interface between a user of a computer and the computer hardware.

⊛. Two views/aspects of Operating System:

1) Operating system as an extended machine:-

The architecture of most computers at machine-language level is primitive and awkward to program, especially for input/output. So one of the major task of operating system is to hide the hardware and present programs with nice, clean, consistent and user friendly. We are not interested on how hardware components work together to perform our tasks, all we need is task to be completed with no complexity and overhead.

Thus all we need is the program should hide the truth about hardware and present simple and nice view. To achieve it all the operating system provides a variety of services that programs can obtain using special instructions called system calls.

9) Operating System as resource manager:-

The major function of any machine is how resources are managed. For example we constructed system which provides all necessary banking operations, Now one of the user needs to deposit amount while another user requires to print statement i.e, first user requires access to database while the second one too requires to access database as well as printer. Users are interested in getting their job done quicker anyhow. But the problem occurs with the system how to manage these requests. The process of managing all those requests/resources is called resource management and this is done by resource manager.

Hardware provides basic computing resources CPU, Memory, I/O devices etc. whereas operating system controls and co-ordinate the use of the hardware among various users.

⊛ Evolution/History of Operating System:-

The first true digital computer was designed by the English mathematician: Charles Babbage. But this computer was purely mechanical and did not have an Operating system. Later Babbage realized that he would need a software for his analytical engine, so he hired a young woman named Ada Lovelace as the world's first programmer. Operating systems are closely tied to the architecture of the computers on which they run. So successive generations of computers describes what their operating systems were like.

1) The first generation of Computer (1945-55):-

→ Vaccum-Tubes were used for building machine.
→ A single group of people designed, built, programmed, operated and maintained each machine.
→ All programming was done in absolute machine code.
→ Programming languages and operating system was unknown.

ii) The second generation of computer (1955-65):-

→ Transistors were used for building machine.
→ Batch systems were used.
→ Programs were written in FORTAN and assembly language.
→ Typical operating systems were used FMS(FORTON Monitor System) and IBSYS (IBM's Operating System).

iii) The third generation of computer (1965-1980):-

→ IC's were used for building machines.
→ Multiprogramming came into existence so that processors can be kept busy 100 percent of the time.
→ Operating systems: OS/360, MULTICS, CTSS were used.

iv) The fourth generation of computer (1980- Present):-

→ LSI's and VLSI's were used for building machines.
→ Microcomputers came into existence.
IBM PC: Operating System was MS-DOS.
IBM PC/AT: Operating System was MS-DOS.
Apple Macintosh: Operating System was Macintosh.
→ Introduction to Windows: Because of the influence of the success of Macintosh, Microsoft decided to build a GUI-based system which originally ran on top of MS-DOS and named it Windows. Later on different versions of Windows like Windows-XP, Windows-7, Windows-8, Windows-10 came.

⊛. Types of Operating Systems:-

1) Mainframe Operating Systems:

→ The operating systems used in mainframe computers are called mainframe operating systems.
→ OS/390 is an example of Mainframe OS.
→ Mainframe OS have very high I/O capacity.
→ They provide three types of services batch, transaction processing and timesharing.

Server Operating Systems:-

→ The operating systems which run on servers, which are either very large personal computers, workstations or even mainframes, are called server operating systems.

→ Linux, Windows 2000 etc. are its examples.

→ They serve multiple users at once over a network and then allow the users to share software and hardware resources.

→ They provide services like print service, file service or web service.

Multiprocessor Operating Systems:-

→ If more than one processor is used to handle multiple tasks at a time and reduce systems work load then, this is called a multiprocessor operating system.

→ By connecting multiple CPUs into a single system we can get heavy computing power out of the machine.

→ Depending upon precisely how they are connected and what is shared, these systems are called parallel computers, multicomputers or multiprocessors.

PC Operating System:

→ The operating system used in Personal Computers are called PC operating systems.

→ They have comparatively less processing power and memory than mainframes.

→ Windows-7, Macintosh, Linux etc. are its examples.

→ They are widely used for word processing, spreadsheets and internet access.

v) Real-time Operating Systems:

→ The operating systems that have time as the key parameter is called Real-time OS.
→ QNX, RTLinux, HART etc are its examples.
→ Real time OS has a well defined, fixed time constraints and processing must be done within the defined constraints.
→ They provide quick event response and thus meet the scheduling deadlines.

vi) Embedded Operating Systems:

→ The operating systems such as Palm OS and Windows CE (Consumer electronics) that run on embedded devices such as a palmtop computer or PDAs (Personal Digital Assistants) and mobile phones are called embedded operating systems.
→ These devices have generally low processing power, memory, size and battery life.

vii) Smart Card Operating Systems:

→ Smart cards are credit card-sized devices containing a CPU chip.
→ They have very low processing power and memory capabilities.
→ Resource management and protection are two of the main tasks of smart card operating systems.

viii) Multiprogramming Operating Systems:-

→ Multiprogramming is a technique to execute number of programs simultaneously in a single processor.
→ Multitasking and multiprocessing are two different forms of multiprogramming.
→ It manages computer related resources like CPU, memory, and I/O devices.
→ It provides high CPU utilization, efficient memory utilization and supports multiple simultaneous interactive users.

Operating System Structures:-

1) Monolithic Systems:- In monolithic systems the operating system is written as a collection of procedures, each of which can call any of the other ones whenever it needs to. In this approach one first compiles all the individual procedures or files containing the procedures and then binds them all together into a single object file using the system linker. This organization suggests a basic structure for the operating system:

→ A main program that invokes the requested service procedure.
→ A set of service procedures that carry out the system calls.
→ A set of utility procedures that help the service procedures.

In this model, for each system call there is one service procedure that takes care of it. The utility procedures do things that are needed by several service procedures, such as fetching data from user programs.

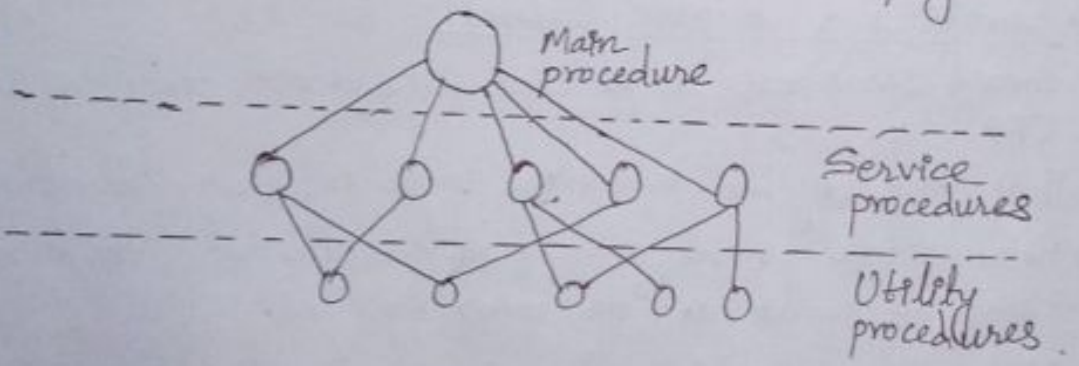


fig. A simple structuring model for monolithic systems.

2) Layered Systems:-

Layer	Function
5.	The operator.
4.	User programs.
3	Input/Output management.
2	Operator-process communication.
1	Memory and drum management.
0	Processor allocation and multiprogramming.

The table above was the structure of the layered systems.

This system had 6 layers as shown in table above. Layer 0 deals with the allocation of processors and multiple processes run on a single processor. Layer 1 did the memory management. It allocated space for processes in main memory and 512K word drum used for holding parts of processes. Layer 2 handeled communication between each process and the operator console. Layer 3 took care of managing the I/O devices and buffering the information streams to and from them. Layer 4 was where the user programs were found and the system operator process was located in layer 5.

iii) Virtual machines:- This system originally called CP/CMS and later renamed VM/370 had multiprogramming and more convenient interface. The heart of this system, known as the virtual machine monitor, runs on the bare hardware and does the multiprogramming, providing several virtual machines to the next layer up. They are exact copies of the bare hardware, including kernel/user mode, I/O, interrupts and everything else the real machine has.

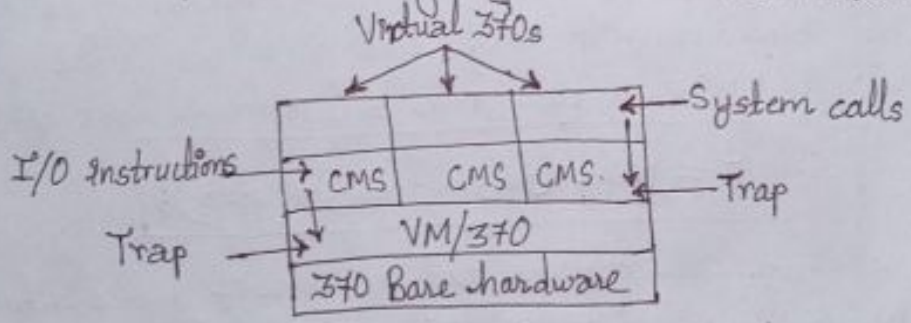


Fig. The structure of VM/370 with CMS.
CMS → stands for Conversational Monitor System.

iv) Client Server Model:-

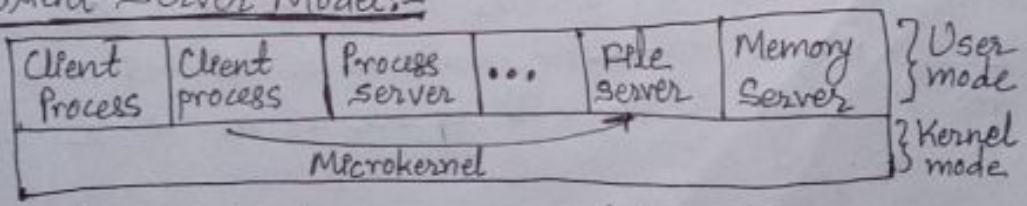


Fig. The client-server model.

In this model to request a service, such as reading a block of a files, a user process (now known as client process) sends the request to a server process, which then does the work and sends back the answer. All the kernel handles the communication between clients and servers. By splitting the operating system up into parts each of which only handles one facet of the system.

⊛. System Calls:-

System calls are the interface between a process (i.e, user program) and the operating system. They are generally available as assembly language instructions, however languages like C that are defined to replace assembly language for systems programming allow system calls to be made directly via procedure calls.

To make a system call mechanism we need three parameters: the first parameter specifies file, the second parameter points to the buffer and the third parameter gives the number of bytes to read. Like all system calls, it is invoked from a C program by calling a library procedure with the same name as the system call; read. A call from a C program looks like:

```
count = read (fd, buffer, nbytes);
```

The system call returns the number of bytes actually read in count. This value is normally the same as nbytes, but maybe smaller if end of file is encountered while reading. If the system call cannot be carried out, either due to an invalid parameter or a disk error, count is set to -1. Programs should always check the results of a system call to see if an error occured.

⊛. Handling System Calls / System call flow with block diagram

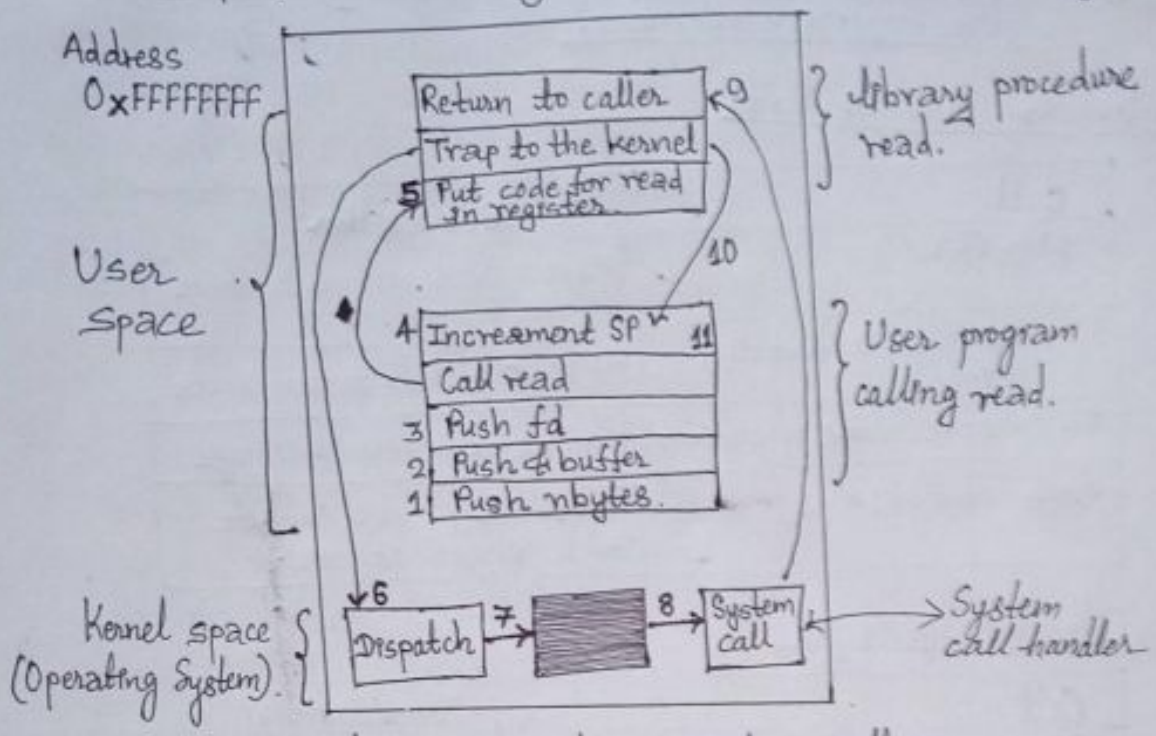


Fig. Steps in making system call

→ For calling the read library procedure the calling program first pushes the parameters onto the stack (steps 1-3 in fig). C compilers push the parameters onto the stack in reverse order. The first and third parameters are passed by value, but the second parameter is passed by refrence.
→ Then the library procedure is called. (step 4).
→ The library procedure puts the system call number in a place where the OS expects it, such as a register. (step 5)
→ The library procedure then executes a TRAP instruction to switch from user mode to kernel mode. (step 6).
→ Started kernel mode examines the system call number and then dispatches to the correct system call handler (step 7).
→ The system call handler runs (step 8).
→ Once system call handler completes its work control is returned to the user space library procedure. (step 9).
→ Then kernel is trapped. (step 10).
→ Finally stack pointer (SP) is incremented to remove the parameters pushed. (step 11).

⑧. Types of System calls / System calls for process, file and directory management.

1) Process management:

Call	Description
pid = fork()	Create a child process identical to the parent.
pid = waitpid(pid, &statloc, options).	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process core image
exit (status)	Terminate process execute and return status.

2) File management:

Call	Description
fd = open (file, how, ...)	Open a file for reading, writing or both.
s = close(fb)	Close an open file
n = read (fd, buffer, nbytes)	Read data from a file into buffer.
n = write (fd, buffer, nbytes)	Write data from a buffer into file.
position = seek (fd, offset, whence)	Move a file pointer
s = stat (name, &buf)	Get a file's status information.

3) Directory and file system management:

Call	Description
s = mkdir (name, mode)	Create a new directory
s = rmdir (name)	Remove an empty directory
s = link (name1, name2)	Create a new entry name2 pointing to name1
s = unlink (name)	Remove a directory entry.
s = mount (special, name, flag)	Mount a file system.
s = umount (special)	Unmount a file system.

⊛. The Shell:

The shell is the UNIX command interpreter. Although it is not a part of operating system, it makes heavy use of many operating system features and thus serves as a good example of how the system calls can be used. It is also the primary interface between a user sitting at his terminal and the operating system, unless the user is using a graphical user interface. Many shells exist, including sh, csh, ksh and bash.

When any user logs in, a shell is started up. The shell has the terminal as standard input and standard output. It starts out by typing the prompt, a character such as a dollar sign, which ~~the~~ tells the user that the shell is waiting to accept a command. If the user now types date then the shell creates a child process and run the data program as the child. While the child process is running, the shell waits for it to terminate. When the child finishes, the shell types the prompt again and tries to read the next input line.

→ The user can specify that standard output be redirected to a file. For example:

date > file

→ Similarly standard ~~or~~ input can be redirected to file as:

sort < file1 > file 2.

This invokes the sort program with input token from file 1 and send the sorted output to file 2.

⊛. Open Source Operating Systems:-

The operating systems that are open to all, anyone can get the source code and make desirable changes are called open-source operating systems.

For example:- Android is open source, so phones of different manufacturer have different look and feel to it.

The code that google provides is called stock code. Manufactures change it according to their needs and add functionalities to their phones to make them different from others. Like Samsung's newly released note 4 has support for stylus, lg g3 has its own awesome multitasking or the camera features. Following are the different open source operating systems for computer available in the market:-

i) Cosmos → This is an open source operating system written mostly in C# programming language. Its full form is open source managed operating system. Till 2016, Cosmos did not intend to be a fully fitted with features but this system allowed other developers to easily build their own OS. It also hid the inner workings of the hardware from the developers thus providing data abstraction.

ii) Free DOS → This was a free operating system developed for systems compatible with IBM PC computers. Free DOS provides a complete environment to run legacy software and other embedded systems. It can be booted from a floppy disk or USB flash drive. Free DOS is licensed under GNU (General Public License) and contains free and open source software.

iii) Genode → Genode is also free as well as open source. It contains a microkernel layer and different user components. Genode can be used as an operating system for computers, tablets etc. as required. As it has a small code system it is also used as a base for virtualisation, interprocess communication, software development etc.

iv) Ghost OS → This is a free, open source operating system developed for personal computers. It started as a research project and development to contain various advanced features like graphical user interface, C library etc. It has features like multiprocessing and multitasking and is based on Ghost kernel. Most of the programming in Ghost OS is done by C++.

⊛ Functions of OS:-

this is out of micro-syllabus but it is imp and maybe asked in exam

1) Security:- The operating system uses password protection to protect user data and similar other techniques. It also prevents unauthorized access to programs and user data.

2) Control over system performance:- It monitors overall system health to help improve performance. It records the response time between service requests and system response to have a complete view of the system health. It provides information needed to troubleshoot problems and help to improve performance.

3) Error detecting aids:- Operating system constantly monitors the system to detect errors and avoid the malfunctioning of computer system.

4) Coordination between other software and users:- Operating systems also coordinate and assign interpreters, compilers, assemblers and other software to the various users of the computer systems.

5) Memory Management:- The operating system manages the Primary memory or main memory. It keeps tracks of primary memory, i.e, which bytes of memory are used by which user program. In multiprogramming, the OS decides the order in which ~~OS~~ process are granted access to memory, and for how long. It allocates the memory to a process when the process requests it and ~~del~~ deallocates the memory when the process has terminated.

6) Process management:- In a multiprogramming environment, the OS decides the order in which processes have access to the processor, and how much processing time each process has. This function of OS is called process scheduling.

vii) Device Management:- An OS manages device communication via their respective drivers. It keeps tracks of all devices connected to system. It decides which process gets access to a certain device and for how long. It allocates devices when needed and deallocates devices when they are no longer required.

viii) File Management:- A file system is organized into directories for efficient or easy navigation and usage. OS keeps track of where information is stored, user access settings and status of every file and more...

Unit-2

Process Management:-

1) Introduction:- A process is an executing program. The CPU executes a large number of programs, while its main concern is the execution of user programs, the CPU is also needed for other system activities. These activities are called processes. A time-shared user program is a process. Process management is one of the function of operating system.

Process vs Program:-

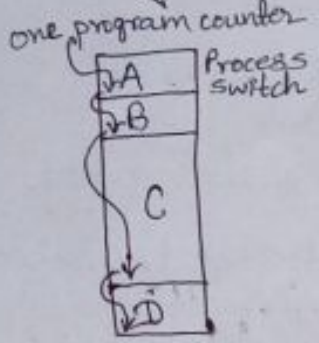
Process	Program.
i) Process is the activity.	i) Program is a group of instructions.
ii) Process is a dynamic entity.	ii) Process is a static entity.
iii) Process has a high resource requirement, it needs resources like CPU, memory, address, I/O during its lifetime.	iii) Program does not have any resource requirement, it only requires memory space for storing the instructions.
iv) Process has its own control block called process control block.	iv) Program does not have any control block.
~~v) Process has its own control~~ v) Process exists for a limited span of time as it gets terminated after the completition of task.	v) Program exists at a single place and continues to exist until it is deleted.

Multiprogramming:- In any multiprogramming system, the CPU switches from process to process quickly, running each for tens or hundreds of milliseconds. This rapid switching back and forth of the CPU is called multiprogramming. In multiprogramming systems, process are performed in a pseudo-parallelism as if each process has its own processor. In fact there is only one processor but it switches back and forth from process to process.

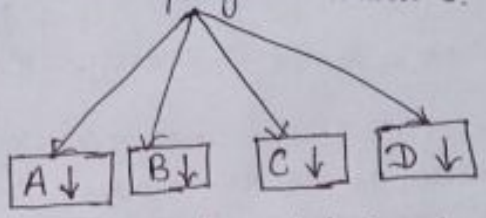
Keeping track of multiple parallel activities is hard for people to do. Therefore, operating system designers over the years have evolved a sequential process that makes parallelism easier to deal with.

Process Model:

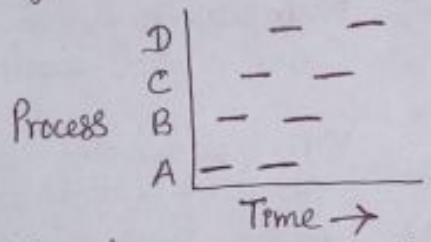
In this model all the runnable software on the computer, sometimes including the OS, is organized into a number of sequential processes. A process is just an instance of executing program, including the current values of program counter, registers and variables.



a) Multiprogramming four programs



b) Conceptual model for four independent, sequential processes.



c) Only one program is active at once.

As we see in the figure (a), a computer multiprogramming four programs in memory. In figure (b) we see four processes each with its own flow of control (i.e, its own logical program counter) and each one is running independently of the other ones. Of course there is only one physical program counter so when each process runs, its logical program counter is loaded into the real program counter. When it is finished, the physical program counter is saved in the process stored logical program counter in memory. But in figure (c) we see that, viewed over a long enough time interval, all the processes have made progress, but at any given instant only one process is actually running.

Process states:

A process is an independent entity with its own input values, output values and internal state. One process may generate some outputs that other process uses as input. For example in the shell command.

cat file1 file2 file3 | grep tree

The first process running cat, concatenates and outputs three files. Depending on the relative speed of the two process, it may happen that grep is ready to run, but there is no input waiting for it. It must then block until some input is available. The process state may be in one of the following:

- New → The process is being created.
- Ready → The process is waiting to be assigned to a processor.
- Running → Instructions are being executed.
- Waiting/Suspended/Blocked → The process is waiting for some event to occur.
- Terminated → The process has finished execution.

The transition of the process states are shown in figure and their corresponding transition is described below:-

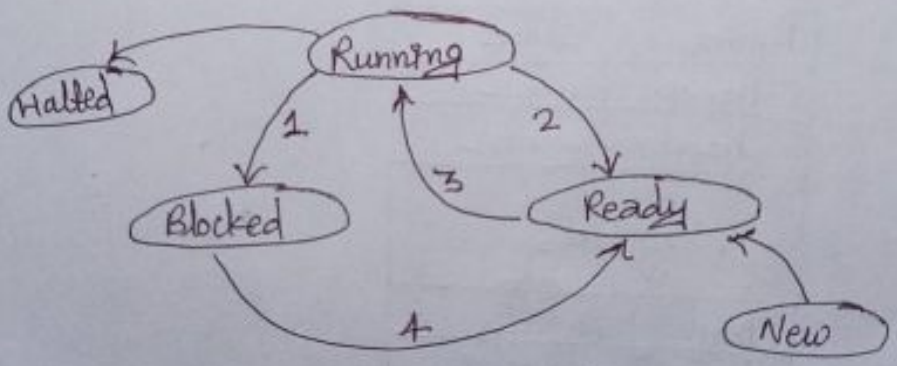


Fig. Typical process states.

Process Control Block / Process Table (Implementation of Processes.)

To implement the process model, the operating systems ~~maintains~~ maintains a table, an array of structures, called the process table or process control block (PCB) or switch frame. Each entry identifies a process with information such as

process state, its program counter, stack-pointer, memory allocation etc. The following is the information stored in a process table:

→ ~~Process state, each process~~.

→ Process state, which may be new, ready, running, waiting or halted.

→ Process number, each process is identified by its process number called process ID.

→ Program counter, which indicates the address of the next instruction to be executed for this process.

→ CPU registers, which vary in number and type, depending on the concrete microprocessor architecture.

→ Memory management information, which include base and bounds registers or page table.

→ I/O status information, composed I/O requests, I/O devices allocated to this process, a list of open files and so on.

→ Processor scheduling information, which includes process priority, pointers to scheduling queues and any other scheduling parameters.

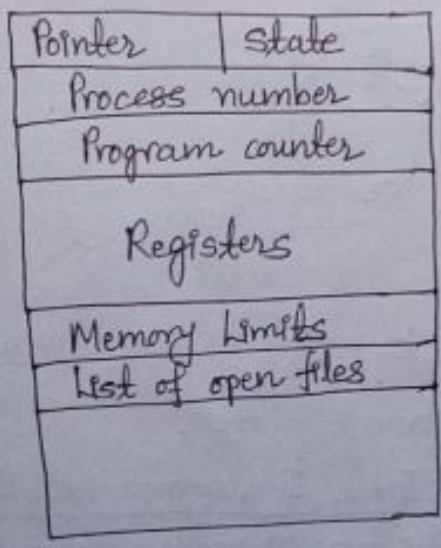


Fig. Process table structure.

2) Threads:

A thread is a path of execution within a process. A process can contain multiple threads. A thread is also called a lightweight processes (LWPs) which are independently scheduled parts of a single program. Each thread represents a seperate flow of control. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improve performance of OS by reducing the overhead thread. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.

Thread vs Process:-

Process	Thread.
i) Process is heavy weight or resource intensive.	i) Thread is light weight, taking lesser resources than a process.
ii) Process switching needs interaction with operating system.	ii) Thread switching does not need.
iii) In multiple processing environments each process executes the same code but has its own memory and file resources.	iii) All threads can share same set of open files, child processes.
iv) If one process is blocked, then no other processes can execute until the first process is unblocked.	iv) While one thread is blocked and waiting, second thread in the same task can run.
v) Multiple processes without using threads use more resources.	v) Multiple threaded processes use fewer resources.
vi) In multiple processes each process operates independently of the others.	vi) One thread can read write or change another thread's data.

Types of Threads:-

Threads are implemented in following two ways:-

1) **User Level threads** → In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing data and message between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

Advantages

→ Thread switching does not require kernel mode privileges.
→ User level thread can run on any operating system.
→ Scheduling can be application specific in the user level thread.
→ User level threads are fast to create and manage.

Disadvantages

→ In a typical operating system, most system calls are blocking.
→ Multithreaded application cannot take advantage of multiprocessing.

2) **Kernel level threads** → In this case, thread management is done by the kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

Advantages

→ Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
→ If one thread in a process is blocked, the kernel can schedule another thread of the same process.
→ Kernel routines themselves can be multethreaded.

Disadvantages

→ Kernel threads are generally slower to create and manage than the user threads.
→ Transfer of control from one thread to another within the same process requires a mode switch to the kernel.

User-Level Threads vs. Kernel-Level Threads:-

User-level threads	Kernel-level threads.
1) User-level threads are faster to create and manage.	1) Kernel-level threads are slower to create and manage.
2) Implementation is by a thread library at the user level.	2) Operating system supports creation of kernel threads.
3) User-level thread is generic and can run on any operating system.	3) Kernel-level thread is specific to the operating system.
4) Multi-threaded applications cannot take advantage of multiprocessing.	4) Kernel routines themselves can be multithreaded.

3). Inter Process Communication (IPC):-

Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. This communication could involve a process letting another process know that some event has occured or the transfering of data from one process to another. This allows a specific program to handle many user requests at the same time.

Inter process communication describes three issues. The first one is how one process can pass information to another. ~~This allows a specific program to~~ The second one is making sure two or more processes do not get in each others way. The third concerns proper sequencing when dependencies are present.

Race Conditions:

A race condition is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in critical section differs according to the order in which the threads execute.

In some operating system, processes that are working together may share some common storage that each one can read and write. The shared storage may be in main memory or it may be a shared file. The location of the shared memory does not change the nature of the communication or the problems that arise. Situation like this where two or more processes are reading or writing some shared data and the final result depends on who runs precisely when race conditions are called.

Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

Critical Section:

Sometimes processes have to access shared memory or files, or doing other critical things that can lead to races. That part of the program where the shared memory is accessed is called the critical region or critical section.

If we could arrange matters such that no two processes were ever in their critical regions at the same time we could avoid races. We need four conditions to hold to have a good solution:-

i) No two processes may be simultaneously inside their critical regions.

ii) No assumptions may be made about speeds or the number of CPUs.

iii) No process running outside its critical region may block other processes.

iv) No process should have to wait forever to enter its critical region.

4>. Implementing Mutual Exclusion:

@ Mutual Exclusion with Busy Waiting:

In this section we will examine various proposals for achieving mutual exclusion, so that while one process is busy updating shared memory in its critical region, no other will enter its critical region and cause trouble.

1> Disabling Interrupts → On a single-processor system, the simplest solution to have each process is disabling all interrupts just after entering its critical region and re-enable them just before leaving it. Once a process has disabled interrupts, it can examine and update the shared memory without fear that any process will interrupt.

This approach is generally unattractive because it is unwise to give user processes the power to turn off interrupts. What if one of them did it, and never turned them on again? that could be the end of system. Thus, disabling interrupts is often a useful technique within the operating system itself but is not appropriate for user processes.

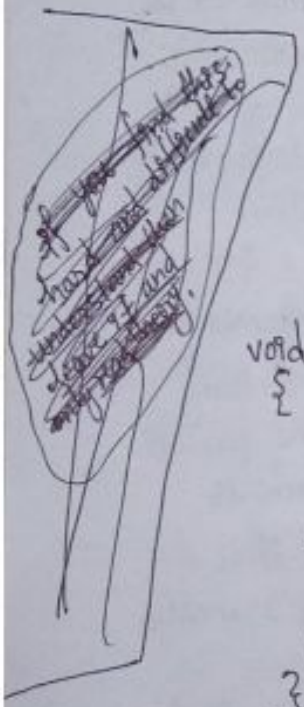
11> Lock Variables: → Consider having a single, shared (lock) variable, initially set to 0. When a process wants to enter its critical region, it first tests the lock. If the lock is 0, the process sets it to 1 and enters the critical region. If the lock is already 1, the process just waits until it becomes 0. Thus 0 means that no processes in its critical region and 1 means that some process is in its critical region.

This idea also contains fatal flaw. Suppose that one process reads the lock and sees that it is 0. Before it can set the lock to 1, another process may get scheduled, runs and sets the lock to 1. When first process runs again it also sets the lock to 1. Now, two processes will be on their critical regions at the same time.

iii) Strict Alternation → The integer variable (turn) initially 0, keeps track of whose turn to enter critical region and updates shared memory. Initially first process finds value of turn 0 and enters it's critical region. Second process also finds it to be 0 and therefore sits in a tight loop continually testing turn ~~until~~ to see when it becomes 1.

Continuously testing a variable until some value appears is called busy waiting. It should usually be avoided since it wastes CPU time. A lock that uses by busy waiting is called a spin lock.

iv) Peterson's Solution → This algorithm consists of two procedures written in ANSI C, which means that function prototypes should be supplied for all the functions defined and used.



```
#define FALSE 0
#define TRUE 1
#define N 2  /*no of processes*/
int turn; /* whose turn is it */
int interested[N]; /*all values initially 0. (i.e, false) */

void enter_region(int process)  /*process is 0 or 1*/
{
  int other; /*number of the other process */
  other = 1 - process; /* the opposite of process */
  interested[process] = TRUE; /*show that you are interested*/
  turn = process;  /*set flag */
  while (turn = process && interested[other] = TRUE)
                              /*null statement */
}

void leave_region (int process) /*process, who is leaving */
{
  interested[process] = FALSE; /*indicates departure from
                                 critical region */
}
```

Before entering its critical region, each process calls enter_region method (i.e, function) with its own process number 0 or 1 as parameter. This call will cause it to wait until it is safe to enter. After it has finished with the shared variables the process calls leave_region method to indicate that it is done and to allow other process to enter.

v) Test and Set Lock (The TSL Instruction) →

This is a proposal that requires a little help from hardware. It reads the contents of the memory word lock into register RX and then stores a non-zero value at the memory address lock.

To use the TSL instruction, we will use a shared variable, (lock) to coordinate access to shared memory. When lock is 0, any process may set it to 1 using the TSL instruction and then read or write the shared memory. When it is done, the process sets lock back to 0 using an ordinary move instruction.

⑥ Sleep and Wake up:

Both Petersons solution and the solution using TSL are correct, but both have the defect of requiring busy waiting. These approaches have unexpected effects and waste CPU time. Sleep and Wakeup is one of the simplest inter process communication primitives that block instead of wasting CPU time.

Sleep is a system call that causes the caller to block, that is be suspended until another process wake it up. The wakeup call has one parameter, the process to be awakened. Alternatively, both sleep and wakeup each have one parameter, a memory address used to match up sleeps with wakeups.

⑦ Semaphore:

This was the situation in 1965, when E.W. Dijkstra (1965) suggested using an inter variable to count the number of wakeups saved for future use. In his proposal, a new ~~vari~~ variable type, called a semaphore was introduced.

A semaphore could have the value 0, indicating that no wakeups were saved, or some positive value if one or more wakeups were pending. Dijkstra proposed having two operations, down and up. The down operation on semaphore checks to see if the value is greater than 0. If so, it decrements the value and just continues.

If the value is 0, the process is put to sleep without completing the down for the moment. Checking the value, changing it and possibly going to sleep, is all done as a single indivisible atomic action. It is guaranteed that once a semaphore operation has started, no other process can access the semaphore until the operation has completed or blocked. This atomicity is absolutely essential to solve synchronization problems and avoiding race conditions.

② Monitors: A monitor is a set of multiple routines which are protected by a mutual exclusion lock. None of the routines in the monitor can be executed by a thread until that thread acquires the lock. This means that only one thread can execute within the monitor at a time. Any other threads must wait for the thread that is currently executing to give up control of the lock.

However a thread can actually suspend itself inside a monitor and then wait for an event to occur. If this happens, then another thread is given the opportunity to enter the monitor. The thread that was suspended will eventually be notified that the event it was waiting for has now occured which means it can wake up and reacquire the lock.

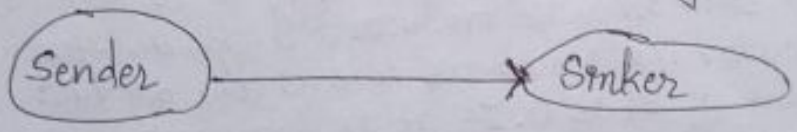
② Message Passing:

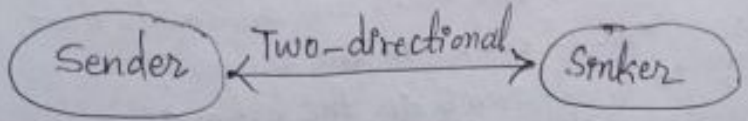
This method of inter process communication uses two primitives, send and receive, which like semaphores and unlike monitors, are system calls rather than language constructs. As such they can easily be put into library procedures, such as:

```
send(destination, &message);
and
receive(source, &message);
```

The former call sends a message to a given destination and the latter one receives a message from a given source. If no message is available, the receiver can block until one arrives. Alternatively, it can return immediately with an error code.

Message passing is commonly used in parallel programming systems like MPI (Message-Passing Interface) which is one of the well-known message-passing system. It is widely used for scientific computing.





The communication link in this scheme has following properties:

- A link is established automatically between every pair of processes that want to communicate.
- A link is associated with exactly two processes.
- Exactly one link exists between each pair of processes.

5) Classical IPC Problems:-

- These problems are used for process synchronization.

@Producer-Consumer Problem: (Bounded-Buffer problem)

In producer-consumer problem two processes share a common, fixed-size buffer. The producer puts information into the buffer and the consumer takes it out. Trouble arises when the producer wants to put a new item in the buffer, but it is already full. The solution is the producer should go to sleep, and to be awakened when the consumer has removed one or more items. Similarly, if the consumer wants to remove an item from the buffer and sees that the buffer is empty, it goes to sleep until the producer puts something in the buffer and wakes it up.

To keep the track of the number of items in the buffer we will need a variable count. If the maximum number of items the buffer can hold is N, then producers code will first test to see if count is N. If it is, then the producer will go to sleep otherwise producer will add an item and increments count. The consumer's code is also similar: first it test count if it is 0. If it is, then goes to sleep. If it is nonzero, it removes an item and decrements count.

```
#define N 100 /*no. of slots in the buffer*/
int count=0; /*no of items in the buffer*/
void producer(void){
    int item;
    while(TRUE){
        item=produce_item(); /*generate next item*/
        if(count==N) sleep(); /*if buffer is full go to sleep*/
        insert_item(item); /*put item in buffer*/
        count=count+1;
        if(count==1) wakeup(consumer); /*was buffer empty?*/
    }
}
```

```
void consumer(void){
    int item;
    while(TRUE){
        if(count==0) sleep();
        item=remove_item();
        count=count-1;
        if(count==N-1) wakeup(producer);
        consume_item(item);
    }
}
```

(b) Sleeping Barber problem:

In this problem there is a barber shop with one barber, one barber chair, and n chairs waiting for customers if there are any to sit on the chair.

→ If there is no customer, then barber sleeps in his own chair.
→ When a customer arrives, he has to wake up the barber.
→ If there are many customers and the barber is cutting a customer's hair, then the remaining customers either wait if there are empty chairs in waiting room or they leave if no chairs are empty.

The solution to this problem includes three semaphores. First is for the customer which counts no. of customers present in the waiting room. Second the barber, 0 or 1 is used to tell whether the barber is idle or working and third mutex is used to provide the mutual exclusion required for process to execute.

When the customer arrives, he acquires the mutex for entering critical region, if another customer enters thereafter, the second one will not be able to anything until the first one has released the mutex.

If the chair is available then the customer sits in the waiting room and increments the variable and also increases the customer's semaphore this wakes up the barber if he is sleeping. At this point, customer and barber both

are awake and the barber is ready to give that person a haircut. When the haircut is over, the customer exits the procedure and if there are no customers in waiting room barber sleeps.

Algorithm:

```
Semaphore customers=0;
Semaphore barber=0;
Mutex seats=1;
int freeSeats=N;

Barber { while (true){ down(customers); /*waits for customer*/
                       down(seats); /*mutex to protect no. of seats*/
                       freeSeats++; /*a chair gets free*/
                       up(barber); /*bring barber for haircut*/
                       up(seats); /*release the mutex on the chair*/
                       /*barber is now cutting hair*/
                     }
}

Customer { while(true){
             if(freeSeats>0){
                freeSeats--; /*sitting down*/
                up(customers); /*notify the barber*/
                up(seats); /*release the lock*/
                down(barber); /*wait in waiting room if
                                barber is busy*/
                /*Customer is now having haircut*/
             }
             else{ up(seats); /*release the lock*/
                   /*customer leaves*/
             }
}
```

© Dinning Philosopher Problem:

refer to book theory image of this is difficult to understand

In this problem five philosophers are seated around a circle table for their lunch. Each philosopher has a plate of spaghetti (a kind of food). The spaghetti is so slippery that a philosopher needs two forks to eat it. Between each pair of plates is one fork. The philosophers alternate between thinking and eating.

When philosopher gets hungry, she tries to acquire her left and right fork one at a time, in either other. If successful is acquiring two forks, she eats for a while then puts down the forks and continues to think.

Solution to Dinning Philosopher problem

Attempt 1: When a philosopher is hungry, she picks up her left fork and waits for right fork. When she gets it, she eats for a while and then puts both forks back to the table.

The problem with this attempt is that, if all five philosophers take their left forks simultaneously then deadlock will happen.

Attempt 2: After taking the left fork, checks the right fork. If it is not available, the philosopher puts down the left one, waits for some time and then repeats whole problem.

Problem is that if all five philosophers take their left fork simultaneously then starvation will occur. A situation in which all the programs continue to run indefinitely but fail to make any progress is called starvation.

Final Attempt:- Among the many attempts using semaphore for each philosopher moves only in eating state if neither neighbour is eating is the perfect solution to the dinning philosopher problem.

Algorithm:

```
#define N 5 /*no. of philosophers*/
#define LEFT (i+N-1)%N /*no. of i's left neighbor*/
#define RIGHT (i+1)%N /*no. of i's right neighbor*/
#define THINKING 0 /*philosopher is thinking*/
#define HUNGRY 1 /*philosopher is trying to get forks*/
#define EATING 2 /*philosopher is eating*/
typedef int semaphore; /*semaphores are a special kind of int*/
int state[N]; /*array to keep track of everyone's state*/
semaphore mutex=1; /*mutual exclusion for critical regions*/
semaphore s[N]; /*one semaphore per philosopher*/
void philosopher(int i) : /*i: philosopher number from 0 to N-1*/
{ while(TRUE){ /*repeat forever*/
    think(); /*philosopher is thinking*/
    take_forks(i) /*acquire two forks or block*/
    eat(); /*eating*/
    put_forks(i); /*put both forks back on table*/
  }
}
void take_forks(int i){
    down(&mutex); /*enter critical region*/
    state[i]=HUNGRY;
    test(i); /*try to acquire 2 forks*/
    up(&mutex); /*exit critical region*/
    down(&s[i]); /*block if forks were not acquired*/
}
void put_forks(i){
    down(&mutex);
    state[i]=THINKING
    test(LEFT);
    test(RIGHT);
    up(&mutex);
}
void test(i){ if(state[i]==HUNGRY&& state[LEFT]!=EATING&& state[RIGHT]
    !=EATING)
  { state[i]=EATING;  up(&s[i]); }
}
```

6.) Process Scheduling:-

The part of operating system that makes the choice which process to run next whenever two or more processes are simultaneously in ready state is called the schedular and the algorithm it uses is called the scheduling algorithm.

Process Scheduling → It is the activity of 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 Behaviour →

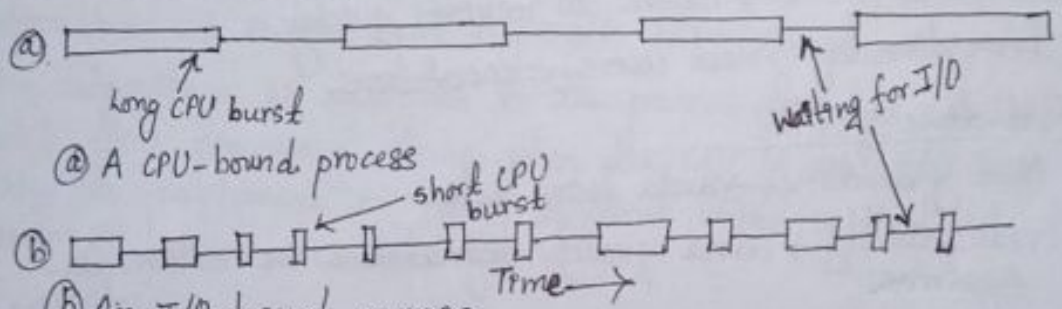


Fig. Bursts of CPU usage alternate with periods of waiting for I/O.

Typically the CPU runs for a while without stopping, then a system call is made to read from a file or write to a file. When the system call completes, the CPU computes again until it needs more data or has to write more data and so on.

Some processes such as fig(a) spend most of their time on computing, while others such as fig(b) spend most of their time waiting for I/O. The former are called compute-bound; the latter are called I/O-bound.

Scheduling Algorithm Goals:-

In order to design a scheduling algorithm, it is necessary to have some idea of what a good algorithm should do. Some goals depend on the environment (batch, interactive, or real time), but there are also some that are desirable in all cases. Some goals are listed below:

i) All systems

Fairness → giving each process a fair share of the CPU.
Policy enforcement → seeing that stated policy is carried out.
Balance → keeping all parts of system busy.

ii) Batch systems

Throughput → maximize jobs per hour.
Turnaround time → minimize time between submission and termination.
CPU utilization → keep the CPU busy all the time.

iii) Interactive systems

Response time → respond to requests quickly.
Proportionality → meet users expectations.

iv) Real-time systems

Meeting deadlines → avoid losing data.
Predictability → avoid quality degradation in multimedia systems.

Scheduling Algorithms:

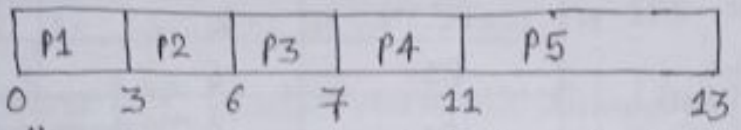
@ Batch System Scheduling:

i) First Come First Serve (FCFS):→ This is non-preemptive scheduling algorithm. This is the simplest scheduling algorithm. Jobs are scheduled in the order they are received. Implementation of this technique is easy and is based on FIFO queue. It has poor performance as average wait time is high.

Example:

Process	Arrival Time	Processing Time (In milliseconds)
P1	0	3
P2	2	3
P3	3	1
P4	5	4
P5	8	2

If the process arrive as per the arrival time, the Gantt chart will be:



Note:- If all the processes arrive at the time 0, then the order of scheduling will be P3, P5, P1, P2 and P4.

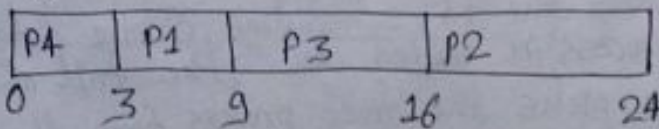
⇒ Average waiting time = (0+3+6+7+11)/5 = 5.4 ms

⇒ Shortest-Job First (SJF) → This is also non-preemptive scheduling algorithm. It is the best approach to minimize waiting time. This algorithm is assigned to the process that has smallest next CPU processing time when the CPU is available. It is easy to implement in Batch systems where required CPU time is known in advance.

Example:

Process	Processing Time (ms)
P1	6
P2	8
P3	7
P4	3

Using SJF, the Gantt chart will be:



Average waiting time = (0+3+9+16)/4 = 7 ms.

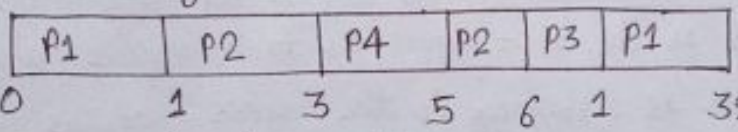
iii) Shortest Remaining Time Next → This is a preemptive scheduling algorithm. In this algorithm, process with the smallest amount of time remaining until completion is selected to execute. This method is advantageous because short processes are handeled very quickly. When a new process is added the algorithm only compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.

Example:

~~The Gantt~~

Process	Arrival time	Processing Time (Burst Time)
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The Gantt chart for preemptive shortest remaining time next scheduling will be



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

(b) Interactive System Scheduling:

i) Round-Robin (RR) Scheduling:- It is one of the oldest, simplest and most widely used algorithm. Each process is assigned a time-interval called its quantum, during which it is allowed to run. If the process is still running at the end of the quantum, the CPU is preempted (i.e, stopped) and given to another process. If the process has blocked or finished before the quantum has elasped, the CPU switching is done when the process blocks. Round-Robin is easy to implement.

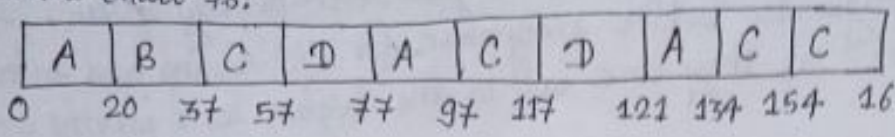
To implement the RR scheduling Queue data structure is used. A new process is added to the tail of the Queue. The CPU schedular picks the first process from the Queue,

allocate processor for a specified time Quantum. After that the CPU schedular will select the next process in the ready Queue.

Example:

Process	Burst Time (Processing Time)
A	53
B	17
C	68
D	24

The Gantt chart is:



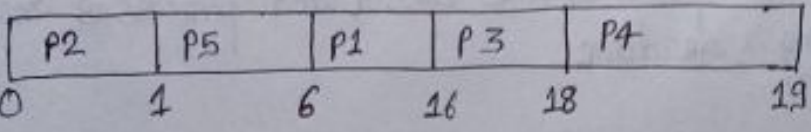
9) Priority Scheduling: [OR Event-Driven (ED) Scheduling]

A priority is associated with each process and the schedular always picks up the highest priority process for execution from the ready queue. Equal priority processes are scheduled FCFS. The level of priority may be determined on the basis of resource requirements, process characteristics and its runtime behaviour. A major problem with a priority based scheduling is indefinite blocking of a lost priority process by a high priority process.

Example:

Process	Processing Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Using priority scheduling the Gantt chart is:



(iii) Multiple Queues:- In this system there are multiple queues with a different priority level set for each queue. Processes in the highest priority level use less CPU time than processes in the next-highest priority level. Processes may move between the queues. If a process uses its entire quantum, it will be moved to the tail of the next-lower-priority-level queue while if the process blocks before using its entire quantum it is moved to the next-higher-level-priority queue. Its advantage is better response for I/O-bound and interactive processes as they are set in the higher level priority queue.

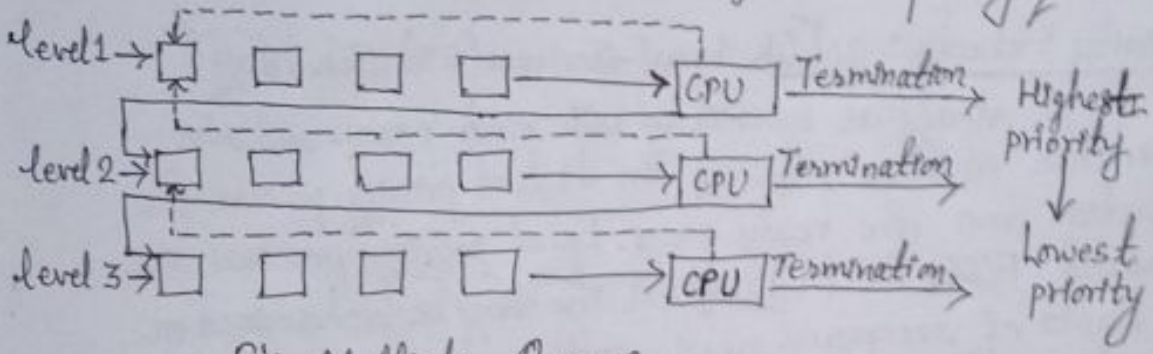


Fig. Multiple Queues

⊛ Overview of Real-Time System Scheduling:

Systems whose correctness depends on their temporal aspects as well as their functional aspects are called real-time systems. Predictability on timing constraints is its key property. These are the systems that carry real-time tasks. These tasks need to be performed immediately with a certain degree of urgency. In real-time systems, the schedular is considered as the most important component which is typically a short term task schedular. The main focus of this schedular is to reduce the response time associated with each of the associated processes instead of handling deadline.

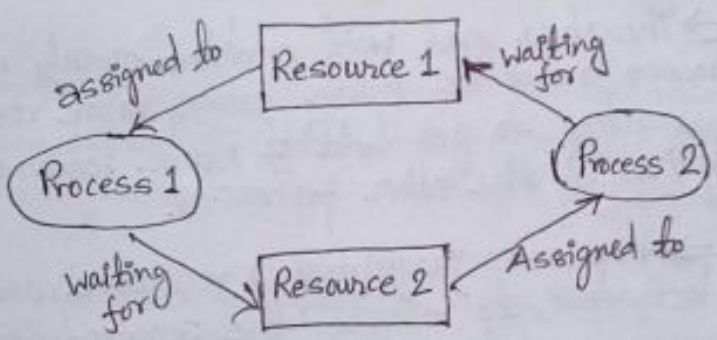
Real-time scheduling is playing an important role in cyber-physical systems (CPSs). Examples of CPSs range from small systems, such as medical equipment and automobiles, to large systems like national power grid.

Unit-3

Process Deadlocks:

resource → Can be a hardware device or a piece of information. e.g. record in database.

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. In operating system when there are two or more processes that hold some resources and wait for resources held by other(s), then deadlock occurs. For example, in below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by Process 2, and Process 2 is waiting for resource 1.



⊛ Resources: To make the discussion of deadlocks as general as possible, we will refer to the objects granted as resources. A resource can be a hardware device (e.g. a Blu-ray drive) or a piece of information (e.g. a record in database). A computer will have many different resources that a process can acquire. A resource is anything that must be acquired, used and released over the course of time. Resources are of two types: preemptable and non preemptable resources.

Preemptable is a resource that can be taken away from its current owner/place and given back later & non preemptable is one that can not be given back.

For example:- Memory is an example of a preemptable resource. Blu-ray recorders are not preemptable at any arbitary moment. Whether a resource is preemptible depends on the context.

⊛ Deadlock Characterization:

Deadlock characterization describes the distinctive features that are the cause of deadlock occurance. It consists of deadlock prerequisites and System resource allocation graph.

1) Deadlock Prerequisites (i.e, Necessary conditions for Deadlock):

i) Mutual Exclusion → In a multiprogramming environment, there may be several processes requesting the same resource at a time. The mutual exclusion condition, allow only a single process to access the resource at a time. While the other processes requesting the same resource must wait and delay their execution until it has been released.

ii) Hold and wait → The hold and wait condition simply means that the process must be holding access to one resource and must be waiting to get hold of other resources that have been acquired by the other processes.

iii) No preemption → A process acquiring a resource, can not be preempted in between, to release the acquired resource. Instead, the process must release the resource it has acquired when the task of the process has been completed.

iv) Circular Wait → The processes must be waiting in a circular pattern to acquire the resource, This is similar to hold and wait. The difference is that the processes are waiting in a circular pattern.

2) Resource Allocation Graph:

It is a directed graph that briefs about the deadlock more precisely. Like every graph, it also has a set of vertices and a set of edges. Further, the set of vertices can be classified into two types of nodes P and R. Where P is the set of vertices indicating the set of active processes and R is the set of vertices indicating all types of resources in the system.

⇒ When a process requests for a resource it is denoted by the request edge in the resource-allocation graph. It is directed edge requesting process P_i to requested resource R_j. i.e, $P_i \rightarrow R_j$.

⇒When a resource is alloted to some process then it is denoted by the assignment edge. The assignment edge is the directed edge from the instance of resource R_j to process P_i i.e, $R_j \rightarrow P_i$.

⇒In the graph, resources are denoted by the rectangles and the processes are denoted by the circles. If a resource has multiple instances then it is denoted by dots. inside the rectangle.

⇒When a process request for an instance of the resource it directs a request edge to the resource. If the resource is able to allocate the resource instance to the requesting process then immediately the request edge is converted to assignment edge.

⇒The request edge always points to the resource rectangle in the graph, not to dots (instance) inside the rectangle. Although the assignment edge nominates the dot (instance) to a process.

Example: Consider we have following set of nodes and edges.

1. There are three active processes $P = \{P_1, P_2, P_3\}$
2. There are four resources $R = \{R_1, R_2, R_3, R_4\}$
3. The set of request edge and assignment edges we have

$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

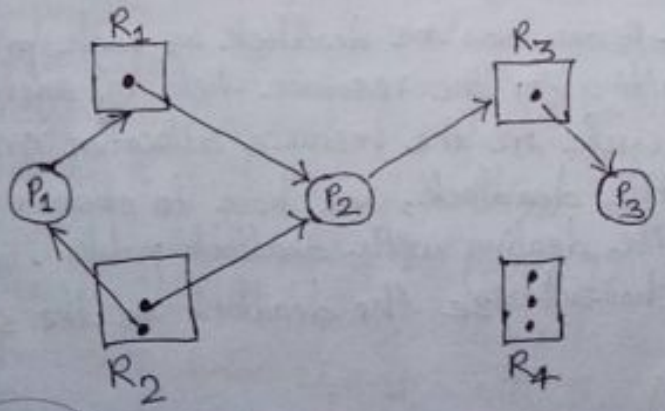


fig. Resource Allocation Graph

The figure shows that the process P_1 has requested for instance of resource R_1 is already holding the instance of resource R_2. The process P_2 has requested for the instance of resource R_3 and is already holding the instances of resource R_1 and R_3. The process P_3 has not requested for any resource

instance but is holding the instance for resource R_3.

If the resource allocation graph has a cycle and every resource has a single instance then it implies that a deadlock has occured. In case, the resources has multiple instances then a cycle in the graph need not be indicating the occurance of deadlock.

Consider that the process P_3 is requesting for the instance of resource R_2 which is already held by the process P_1 and P_2. In this case, we will observe that there are two cycles in the resource allocation graph:

- $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

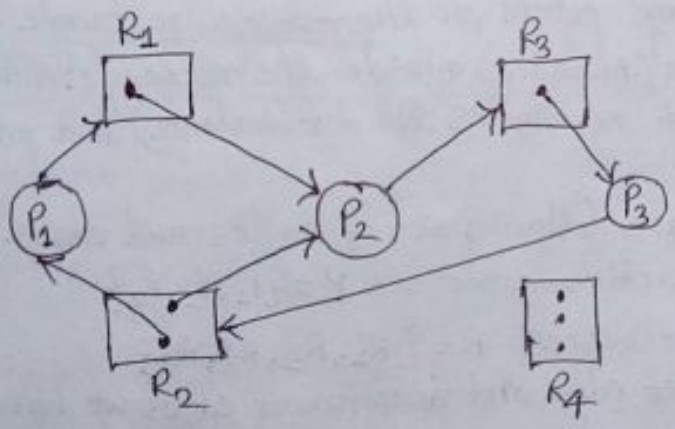


fig. Resource allocation graph with Deadlock

Process P_1, P_2 and P_3 are now in deadlock as each process in the cycle is waiting for the resource held by another process. But every cycle in the resource allocation graph does not indicate the deadlock, you have to observe the cycle carefully while dealing with deadlock problem. So, this is how you can characterize the deadlock in the system.

#Handling Deadlocks:

1) The Ostrich Algorithm: The ostrich algorithm is a strategy of ignoring potential problems on the basis that they may be exceedingly rare. It is named for the ostrich effect which is defined as "to stick one's head in the sand and pretend there is no problem". It is used when it is more cost-effective to allow the problem to occur than to attempt its prevention. This approach may be used in dealing with deadlocks in concurrent programming if they are believed to be very rare and the cost of detection or prevention is high. The UNIX and Windows operating system take this approach.

2) Deadlock Detection:

@. Deadlock Detection with one resource of Each Type:

For any system we construct a resource graph. If graph contains one or more cycles, a deadlock exists. Any process that is part of a cycle is deadlocked. If no cycles exist, the system is not deadlocked.

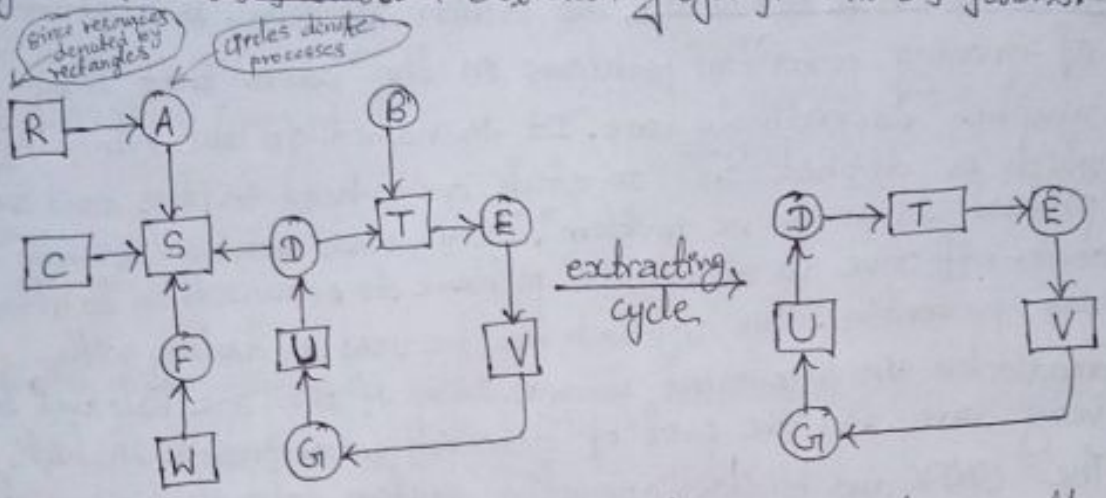
In this method only one resource of each type exists. Such a system might have one scanner, one CD recorder, one plotter etc. but no more than one of each class of resource.

Let us take an example: Consider a system with seven processes, A to G and six resources R to W. The state of which resources are currently owned and which ones are currently being requested as follows:

i) Process A holds R and wants S.
ii) Process B holds nothing but wants T.
iii) Process C holds nothing but wants S.
iv) Process D holds U and wants S and T.
v) Process E holds T and wants V.
vi) Process F holds W and wants S.
vii) Process G holds V and wants U.

The question is: "Is this system deadlocked, and if so, which processes are involved"?

To answer this question first we construct the resource graph of given data ~~as follows~~ and extracting cycle from it as follows:-



We can see that this graph contains one cycle. From this cycle we can see that processes D, E and G are all deadlocked.

(b) Deadlock Detection with multiple resources of each type:

When multiple copies of some of the resources exist, a different approach is needed to detect deadlocks. We will now present a matrix-based algorithm for detecting deadlock among n processes, P_1 to P_n. Let the number of resource classes be m, with E_1 resources of class 1, E_2 resources of class 2, and generally E_i resources of class i ($1 \le i \le m$). E is the existing resource vector. For example if class 1 is tape drivers, then $E_1 = 2$ means the system has two tape drives.

At any instant, some of the resources are assigned and are not available. Let A be the available resource vector, with A_i giving number of instances of resource i that are currently available (i.e, unassigned). For example if both of our two tape drivers are assigned then A_1 will be 0.

Now we need two arrays, C the current allocation matrix and R the request matrix.

where,

C_{ij} = no. of instances of resource j that are held by process i.

R_{ij} = no. of instances of resource j that P_i wants.

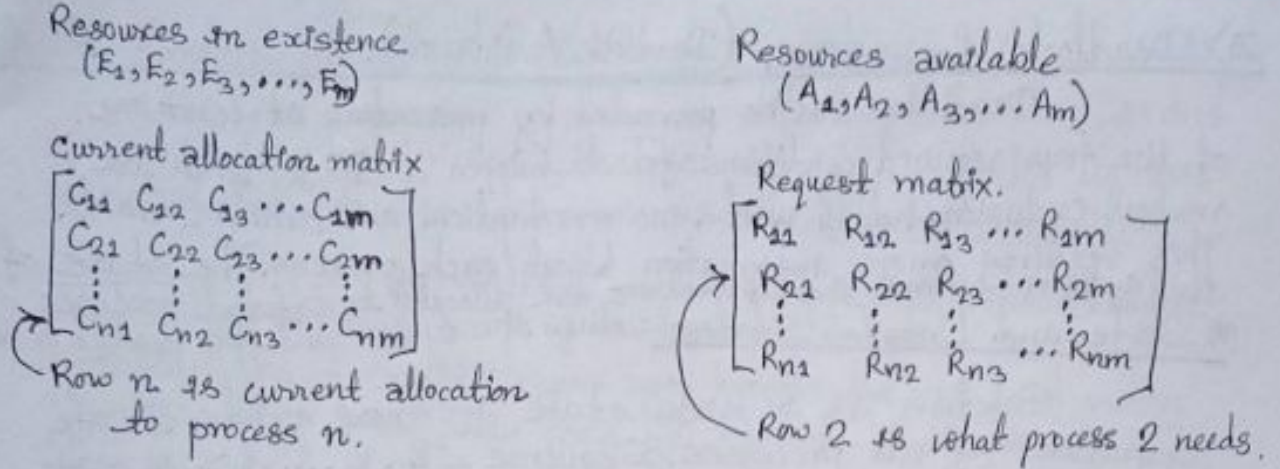


Fig. The four structures needed by the deadlock detection algorithm.

This algorithm is based on comparing vectors. Let us define the relation $A \leq B$ on two vectors A and B to mean that each element of A is less than or equal to the corresponding element of B. Each process is initially said to be unmarked. As the algorithm processes, processes will be marked, indicating they are able to complete and are thus not deadlocked. When the algorithm terminates, any unmarked processes are known to be deadlocked.

The deadlock detection algorithm can now be given, as follows:-

i). Look for an unmarked process P_i for which the ith row of R is less than or equal to A.

ii) If such a process is found, add the ith row of C to A, mark the process and go back to step 1. (i.e, step i).

iii). If no such process exists, the algorithm terminates.

The selected process is run until it finishes. If all the processes are ultimately able to run, then none of them are deadlocked. If some of them can never run, they are deadlocked.

Example:

$E = (4\ 2\ 3\ 1)$ (Tape drives, Plotters, Scanners, CD Roms)

$A = (2\ 1\ 0\ 0)$ (Tape drives, Plotters, Scanners, CD Roms)

current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Fig. An example of deadlock detection algorithm.

3) Deadlock Prevention: (Deadlock Avoidance):

Deadlocks can be prevented by preventing at least one of the four required conditions for to happen deadlock which are mutual exclusion, hold & wait, no preemption and circular wait. This requires more information about each process. The state of the system informs that if resources are allocated to different processes then the system undergoes deadlock or not.

⊛ Safe and Unsafe States:

System is in safe state if there exists a safe sequence of all processes. Sequence $\langle P_1, P_2, \ldots, P_n \rangle$ is safe if for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$.

→ If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.

→ When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.

→ When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Note:

i) If a system is in safe state ⇒ no deadlocks.

ii) If a system is in unsafe state ⇒ possibility of deadlock.

iii) Avoidance ⇒ Ensure that a system will never enter an unsafe state.

fig. safe, unsafe, deadlock state

@ The Banker's Algorithm for a Single Resources:-

The extension of deadlock detection algorithm, and a scheduling algorithm that can avoid deadlocks is due to Dijkstra (1965); is known as the banker's algorithm.

It is modeled on the way a small-town banker might deal with a group of customers to whom he has granted lies of credit. (Years ago, banks did not lend money unless they knew they could be repaid). What the algorithm does is check to see if granting the request leads to an unsafe state. If so, the request is denied. If granting the request leads to safe state, then it is carried out.

The banker's algorithm considers each request as it occurs, seeing whether granting it leads to a safe state. If it does, the request is granted; otherwise it is postponed until later. To see if a state is safe, the banker checks to see if he has enough resources to satisfy some customer. If so, those loans are assumed to be repaid, and the customer now closest to the limit is checked and so on. If all loans can eventually be repaid, the state is safe and the initial request can be granted.

For Example:- If we have situation as ~~bel~~ figure below then it is safe state because with 10 free units one by one all customers can be served.

Process	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7
Free: 10		

Note:- to understand how this is ~~so~~ in safe state and to understand each process once refer youtube video by darshan institute of engineering and technology.

(b) The Banker's Algorithm for Multiple Resources:-

The bankers algorithm can be generalized to handle multiple resources as in figure below:-

Process	Tape drives	Plotters	Printers	Blu-rays
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	Blu-rays
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still assigned.

Fig. The banker's algorithm with multiple resources.

The algorithm for checking to see if a state is safe can now be stated as;

i) Look for a row, R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock since no process can run to completion.

ii) Assume the process of the chosen row requests all the resources it needs (which is guaranteed to be possible) and finishes. Mark that process as terminated and add all of its resources to the A vector.

iii) Repeat steps (i) and (ii) until either all processes are marked terminated (in which case the initial state was safe) or no process is left whose resource needs can be met (in which case the system was not safe).

If several processes are eligible to be chosen in step (i), it does not matter which one is selected.

4) Recovery From Deadlock:

When deadlock is detected, then our system stops working, and after the recovery of the deadlock, our system start working again. Therefore, after the detection of deadlock, a method/way must require to recover that deadlock to run the system again. The method/way is called as deadlock recovery. Following are the two main ways of deadlock recovery:-

(a) Deadlock Recovery through Preemption:

It is the ability to take a resource away from a process, have another process use it, and then give it back without the process noticing. It is highly dependent on the nature of the resource. Deadlock recovery through preemption is too difficult or sometime impossible.

(b). Deadlock Recovery through RollBack:

In this case, whenever a deadlock is detected, it is easy to see which resources are needed. To do the recovery of deadlock, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource just by starting one of its earlier checkpoints.

If my notes really helped you,then you can support me on esewa for my hardwork.

Esewa ID: 9806470952

Unit-4

Memory Management

Memory management is a functionality of operating system which manages primary memory. It keeps track of each and every memory allocation either it is allocated to some process or not. It also decides which process will get memory at what time. It tracks whenever some memory get freed or unallocated and correspondingly updates the state. The part of operating system that manages the memory hierarchy is called the memory manager.

⊛. Monoprogramming vs Multiprogramming:

Monoprogramming	Multiprogramming
i) In monoprogramming memory contains only one program at any point of time.	i) In multiprogramming memory contains more than one user program.
ii) In monoprogramming CPU executes the program and I/O operation is encountered, during that time CPU sits ideal. So, CPU is not efficiently used.	ii) In multiprogramming when one user program contains I/O operation CPU switches to next user program. So, CPU is efficiently used.
iii) In monoprogramming main memory needs less space as only one program sits in main memory.	iii) In multiprogramming main memory needs more space as it contain more than one program in its main memory.
iv) Fixed size partition is used in monoprogramming.	iv) Both fixed and variable size partition can be used in multiprogramming.
Example: Old mobile Operating Systems	Example: Windows 7,8,10.
Advantages: → Allocation of memory is easy & cheap. → Eliminates external fragmentation. → More efficient swapping.	Advantages: → CPU never becomes ideal. → Supports multiple users. → Resources are wisely used.
Disadvantages: → Longer memory access time. → Internal fragmentation. → Inverted page tables.	Disadvantages: → Difficult to program a system. → Tracking of all tasks is sometimes difficult to handle.

⊛. Modeling Multiprogramming:-

Assume that a process spend p% of its time waiting for I/O. With n processes in memory the probability that all processes are waiting for I/O (meaning the CPU is idle) is p^n. The CPU utilization is then given by;

$$\text{CPU utilization} = 1 - p^n$$

Example:- If $p = 0.5$ & $n = 4$ then,

$$1 - p^n = 1 - 0.5^4$$
$$= 0.9375$$

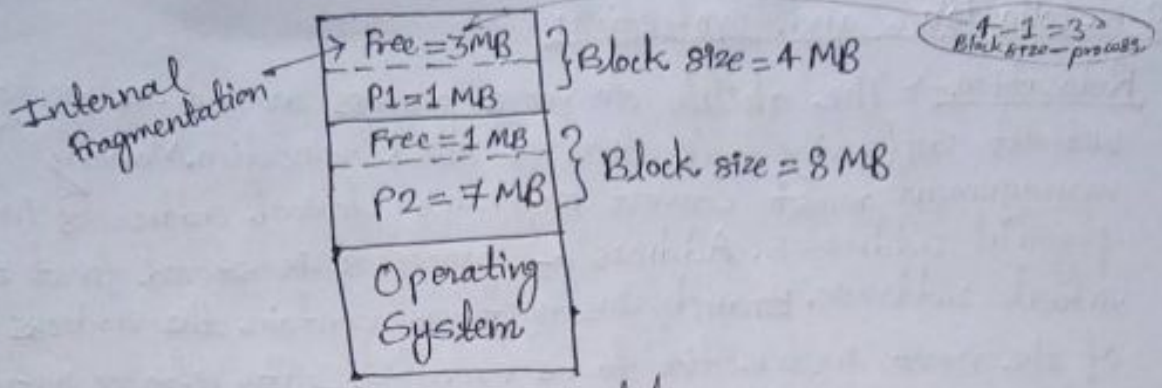
Note: In monoprogramming CPU utilization is 1-p. p is often >0.5 which shows CPU utilization is poor in monoprogramming. But above example of multiprogramming shows that CPU utilization is much better.

⊛. Multiprogramming with fixed and variable partitions:

In multiprogramming environment several programs reside in primary memory at a time and CPU passes its control rapidly between these programs. One way to support multiprogramming is to divide the main memory into several partitions each of which is allocated to single process. Depending on how and when partition are created there may be two types of partition: static & dynamic. (i.e, fixed & variable).

a) Fixed partition:

→ Memory is divided into several fixed size partition where each partition will accomodate only one program for execution.
→ The number of programs residing in the memory will be bounded by the number of partition.
→ When a program terminates the partition is freed for another program waiting in queue.
→ When job arrives it can be put into the input queue for the smallest partition large enough to hold it.
→ Since, the partitions are fixed in this schema, any space in a partition can not be used by job is wasted while that job runs.



Fixed size partition

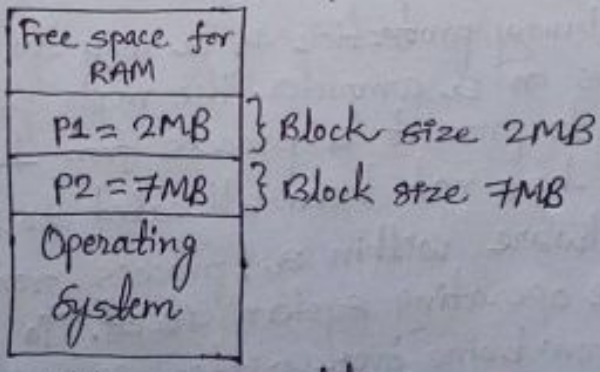
Advantages:
- → Easy to implement.
- → Little OS overhead.
- → Efficient utilization of process and I/O devices

Disadvantages:
- → Internal & External Fragmentation.
- → Limit process size.
- → Limitation on degree of multiprogramming.

b) Dynamic partition: (Variable partition):

→ Memory management approach is to create partition dynamically to meet the requirements of each requesting process, this technique is called variable partitioning.

→ When a process terminates or is swapped out then the memory manager can return the vacant space to pull off free memory area from which partition allocation are made.



variable size partition

Advantages:
- → No internal fragmentation.
- → No limitation on process size.
- → No limitation on degree of multiprogramming.

Disadvantages:
- → Difficult implementation.
- → External fragmentation.

⊛. Relocation and protection:

Relocation → The ability to move process around in memory without affecting execution is called relocation. Memory management must convert programs logical addresses into physical addresses. Address of processes is stored first as virtual address. Branch instructions contain the address of the next instruction to be executed. Data refrance instructions contain the address of byte or word of data referenced.

There are two types of relocations: static and dynamic. In static relocation program must be relocated before or during loading of process into memory. Program must always be loaded into same address space in memory, or relocator must be run again. In dynamic relocation process can be freely moved around in memory. Virtual-to-physical address space mapping is done at run-time.

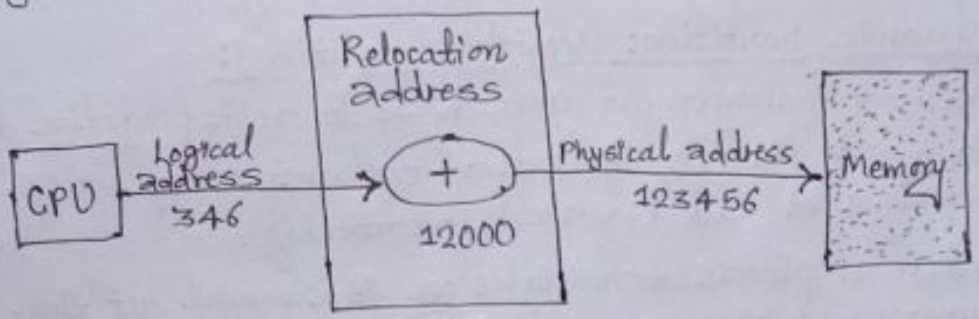


Fig: Memory relocation.

Protection → Memory protection is a way to control memory access rights on a computer. The main purpose of memory protection is to prevent a process from ~~being~~ accessing memory that has not been allocated to it. This prevents a bug or malware within a process from affecting other processes, or the operating system itself. To prevent data and instructions from being over-written is write protection and to ensure privacy of data and instructions is read protection. OS needs to be protected from user processes and user processes need to be protected from each other.

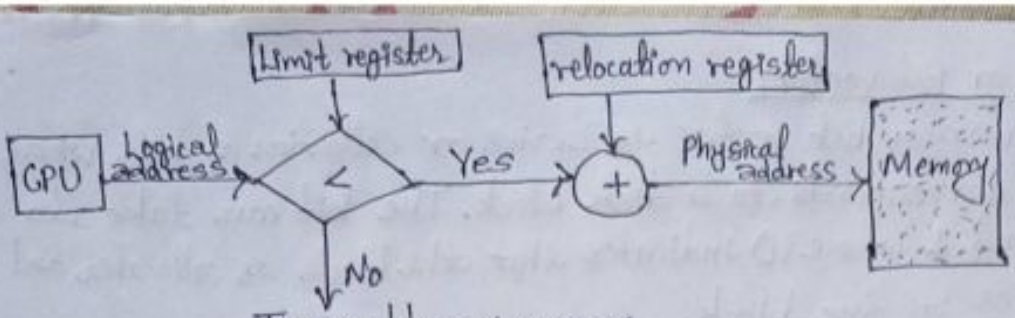


Fig. Memory protection and relocation.

#Space Management

⊛. fragmentation:

As a process are loaded and removed from the memory, the free memory space is broken into little pieces. It happens after sometime that process cannot be allocated to memory block considering their small size and memory blocks remain unused, this problem is known as fragmentation.

There are two types of fragmentation: Internal and External. In Internal fragmentation memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process. In External fragmentation total memory space is enough to satisfy a request or to reside a process in it, but it is not continious so it can't be used.

⊛. Compaction:

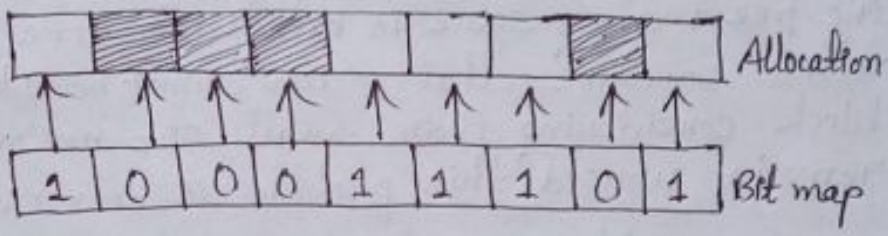
Compaction is a technique by which the programs are relocated in such a way that the small chunks of free memory space are made continious to eachother & clubbed together into a single free partition, that may be big enough to accomodate some more programs.

By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.

⊛. Bitmap or Bit vector:

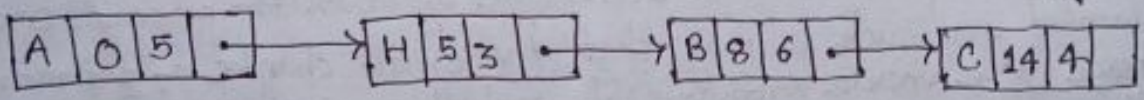
A Bitmap or Bit vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1 where, 0 indicates that the block is allocated, and 1 indicates a free block.

The main problem with this scheme is the size of the allocation unit. The smaller the allocation unit, the larger the bit map has to be. But if we choose larger allocation unit, we may waste memory. The other problem with this scheme is to allocate memory to process. For this reason bit maps are not often used.



⊛. Linked lists:

It is the another way to keep track of memory of allocated and free memory segments, where segment either contain a process or is a empty hole between two processes. In this approach, the free disk blocks are linked together i.e, a free block contains a pointer to the next free block. The block number of very first disk block is stored at a seperate location on disk and is also cached in memory.



⊛. Memory Allocation Strategies:-

a) First fit → In this approach the process manager scans along the list of segements until the first free partition with large enough hole which can accomodate the process is found. It finishes after finding the first suitable free partition.

Advantage → Fastest algorithm because it searches as little as possible.

Disadvantage → Wastage of remaining unused memory left after allocation.

b) Next fit → It works same as first fit, except that it keeps track of where it is, whenever it finds a suitable hole. The next time, it starts searching the list from the place where it left off last time, instead of always at the beginning, as first fit does.

(this can be escaped)

c) Best fit → The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first search the entire list of free partition and consider the smallest hole that is sufficient for the requesting process.

Advantage → Memory utilization is much better than first fit.

Disadvantage → It is slower & may even tend to fill up memory with tiny useless holes.

d) Worst fit → In worst fit, approach is to allocate largest available free partition so that the partition left will be big enough to be useful. It is the reverse of the best fit.

Advantage → Reduce the rate of production of small gaps.

Disadvantage → If a process requiring larger memory arrives at a later stage then it cannot be accomodated as the largest hole is already split & occupied.

Numerical Problem: Given memory partitions of 100K, 500K, 200K, 300K and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K and 426K (in order)? Which algorithm makes the most efficient use of memory?

Solution:

For First-fit

i) 212K is put in 500K partition.
ii) 417K is put in 600K partition.
iii) 112K is put in 288K partion (new partition 288K = 500K - 212K)
iv) 426K must wait.

For Best-fit

i) 212K is put in 300K partition.
ii) 417K is put in 500K partition.
iii) 112K is put in 200K partition.
iv) 426K is put in 600K partition.

For Worst-fit

i) 212K is put in 600K partition.
ii) 417K is put in 500K partition.
iii) 112K is put in 388K partition.
iv) 426K must wait.

⇒ In this example, Best-fit turnsout to be the best.

#Virtual Memory

Virtual memory is a technique of executing programs/instructions that may not fit entirely in the system memory. It is implemented with the help of secondry storage device by transferring data from the main memory to secondry memory and vice-versa. The main objective of this method is to provide multiprogramming. The use of virtual memory has its own limitations mainly with speed. So, it is generally better to have as much physical memory as possible so programs work directly from RAM or physical memory.

⊛. Paging :

Paging is a non-contiguous memory allocation technique which allows the physical address space of a process to be non-contigious. Logical address space is divided into equal size pages but physical address space is divided into equal size frames. 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 disk that set upto emulate the computers RAM.

Paging is a memory management technique in which process address space is broken into blocks of same size

called pages (size is power of 2). Similarly main memory is divided into small fixed size block of memory called frames & the size of frame is kept the same as that of page to have optimum utilization of the main memory & to avoid external fragmentation.

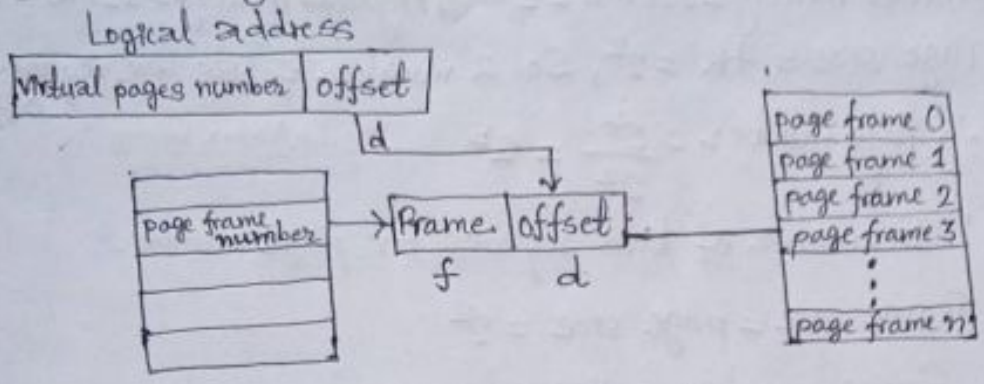


fig: paging hardware.

⊛. Page Table:

A page table is a data structure, used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses. It holds information such as page number, offset, present/absent bit, modified bit etc. It acts as a bridge between virtual pages and page frames thus mathematically we can say that the page table is a function, with the virtual page number as argument and the physical frame number as result. Page table entry has the following information:

Frame Number	Present/Absent	Protection	Refrence	Caching	Dirty

The number of bits required depends on the number of frames. Number of bits for frame = $\frac{\text{Size of physical memory}}{\text{Frame size}}$.

Virtual address space = size of process.

Process size = 2^x bytes, no. of bits in virtual address space = x bits.

frame size = page size.

Page number = $\frac{\text{virtual memory}}{\text{page size}}$

Q. Consider a virtual memory and physical memory of size 128 MB and 32 MB respectively. Assume that page size is 4K, what will be the number of bits required for the page number, frame number and offset?

Solution:

Virtual memory = 128 MB = 2^7 & Physical memory = 32 MB = 2^5

Page size = 4K = 2^2, So, 2 number of bits are required for offset.

$\therefore$ Page number = $\frac{2^7}{2^2} = 2^5$ (i.e. $\frac{\text{Virtual memory}}{\text{page size}}$)

$\therefore$ 5 number of bits required for page number.

Frame size = page size = 2^2

$\therefore$ Frame number = $\frac{\text{Physical memory}}{\text{frame size}}$

$= \frac{2^5}{2^2}$

$= 2^3$

$\therefore$ 3 number of bits required for frame number.

⊛. Handling Page Faults:-

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So, when page fault occurs then the following sequence of event happens:

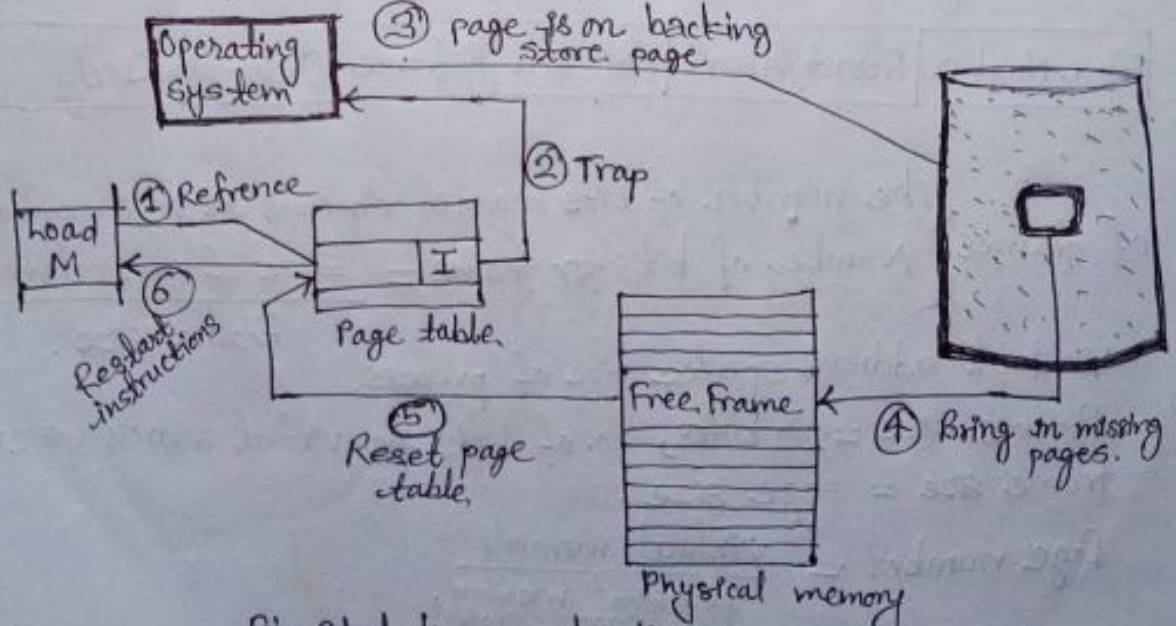


Fig: Block diagram handling page faults.

→ The memory address requested is first checked, to make sure it was a valid memory request.
→ If the refrence was invalid, the process is terminated. Otherwise, the page must be paged in.
→ A frame is located, possibly from a free-frame list.
→ A disk operation is scheduled to bring in the necessary page from disk to allow or block processes on an I/O.
→ When the I/O operation is complete, the processes page table is updated with the new frame number and the invalid bit is changed to indicate that this is now a valid page refrence.
→ The instruction that caused the page fault must now be restarted from the beginning.

⊛ TLB's:

On the cache miss, there will be two memory accesses. Each virtual memory refrence can cause two physical memory accesses: one to fetch the page table and other to fetch the data. To overcome this problem a high-speed cache is set up for page table entries called a Translation Look aside Buffer (TLB). TLB is nothing but a special cache used to keep track of recently used transactions.

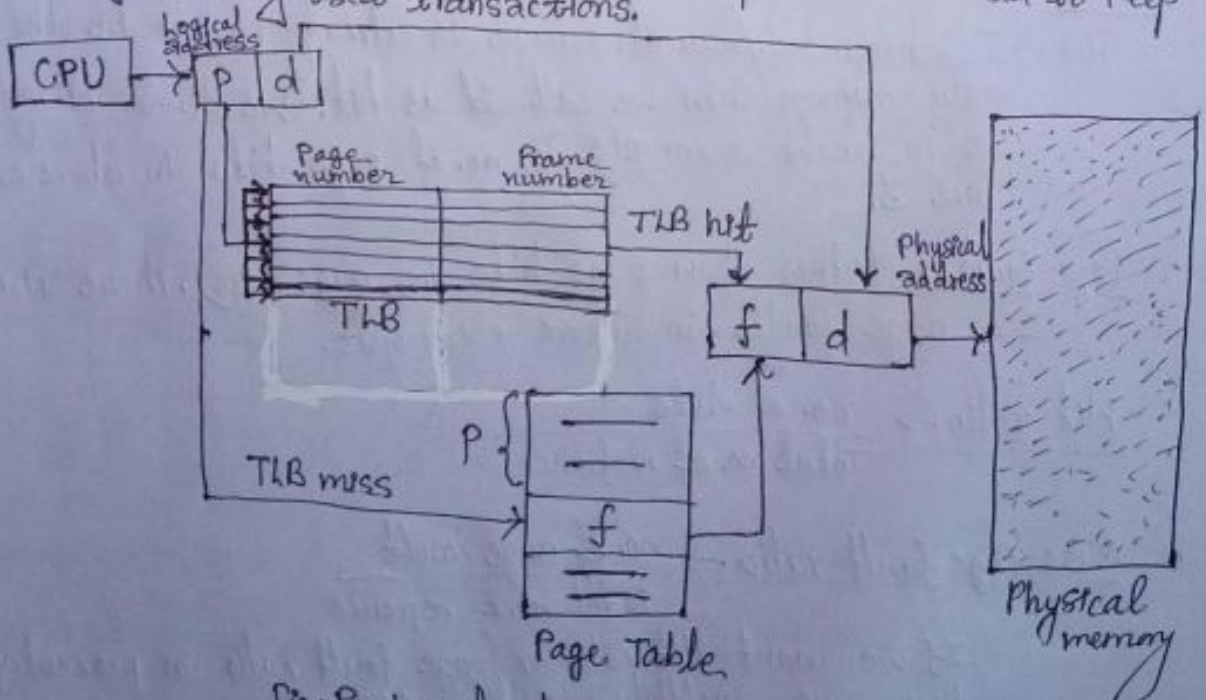


Fig: Paging hardware with TLB.

#Page Replacement Algorithms:

A page replacement algorithm is needed to decide which page needs to be replaced when new page comes in. Since physical memory is much smaller than virtual memory, page faults happen. So, OS might have to replace one of the existing page with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace so that no. of page faults may be reduced.

1) First In First Out (FIFO) Page Replacement Algorithm:

It is the simplest page replacement algorithm. The idea behind this is replace a page which is the oldest page of all pages of main memory.

For Example:- Refrence string sequence (7,0,1,2,0,3,0,4,2,3,0,3,1,2,0) (15 page refrences in disk) with 3 frames.

Solution:

	7	0	1	2	0	3	0	4	2	3	0	3	1	2	0
F_1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
F_2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
F_3			1	1	1	1	0	0	0	3	3	3	3	2	2
					H							H			H

(3 frames)

(Any page staying for longest time in memory is to be replaced)

(Since 3 is already present in the memory so placed at same place replacing 2 like)

Note:

Hit → If demanded/requested page is already present in the main memory then we call it as hit. We denote hit by H as in above example. So, no. of page hits in above example are 3.

Page fault → (other than page hits i.e, page miss). There are 12 page faults in above example.

Hit ratio → $\frac{\text{no. of hits}}{\text{Total no. of refrence}}$

Miss/Page fault ratio → $\frac{\text{no. of page fault}}{\text{Total no. of refrence}}$

If we want Hit ratio or page fault ratio in percentage then we multiply result by 100.

Advantages:
→ Easy to understand and program
→ Distribute fair chance to all.

Disadvantages:
→ FIFO is likely to replace heavily used pages and they are still needed for further processing.
→ System needs to keep track of each frame.

*) Belady's anomaly:- Belady's anomaly is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns. This phenomenon is commonly experienced when using FIFO page replacement algorithm.

For example:- If we consider refrence string 1,2,3,4,1,2,5,1,2,3,4,5 and no. of frames = 3, we get 9 total page faults, but if we increase and make no. of frames = 4, we will get 10 page faults.
[We can solve and show this by FIFO technique as we did before].

2) Second chance page replacement algorithms:

It is the modified form of the FIFO page replacement algorithm. One way to implement is to have circular queue. If the value is equal to 0, then we proceed to replace the page. But if the refrence bit is equal to 1, then we give the page second chance. When the page gets second chance, its refrence bit is cleared.

Example:- Consider refrence string 2,3,2,1,5,2,4,5,3,2,5,2 and no. of frames = 3, we get 7 total page faults by using this algorithm.

	2	3	2	1	5	2	4	5	3	2	5	2
F_1	2(0)	2(0)	2(1)	2(1)	2(0)	2(1)	2(0)	2(0)	3(0)	3(0)	3(0)	3(0)
F_2		3(0)	3(0)	3(0)	5(0)	5(0)	5(0)	5(1)	5(1)	5(0)	5(1)	5(1)
F_3				1(0)	1(0)	1(0)	4(0)	4(0)	4(0)	2(0)	2(0)	2(1)
			H			H		H			H	H

Total page faults = 7
& no. of hits = 5

Advantages:
→ Improvement over FIFO
→ Allows commonly used pages to stay in queue.

Disadvantage:
→ Suffers from Belady's anomaly.

3) Optimal page replacement algorithm:

In this algorithm pages are replaced which would not be used for the longest duration of time in the future. It has lowest page fault rate of all the algorithm.

Example: Refrence sequence (7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1) with 3 frames.

Solution:

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F_1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
F_2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
F_3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
					H		H		H	H			H	H		H	H	H		H	H

no. of refrences = 20

no. of hits = 11

no. of misses = 9

∴ Hit rate = $\frac{11}{20} \times 100\% = 55\%$

& Miss rate = $\frac{9}{20} \times 100\% = 45\%$

Advantages:

→ lowest page fault rate.

→ Never ~~suffrs~~ suffers Belady's anomaly.

Disadvantages:-

→ Not all OS can implement this algorithm.

→ Error detection is harder.

4) LRU Page Replacement Algorithms:-

In this algorithm, the page that has not been used for the longest period of time has to be replaced. This page replacement algorithm looks backward in time, rather than forward.

Example:- Refrence sequence (7,0,1,2,0,3,0,4,2,3,2,1,2,0,1,7) with 3 frames.

Solution:

	7	0	1	2	0	3	0	4	2	3	2	1	2	0	1	7
F_1	7	7	7	2	2	2	2	4	4	4	4	1	1	1	1	1
F_2		0	0	0	0	0	0	0	0	3	3	3	3	0	0	0
F_3			1	1	1	3	3	3	2	2	2	2	2	2	2	7
					H		H				H		H		H	

5) Least Frequently Used (LFU) Page Replacement Algorithm:

This algorithm selects a page for replacement if the page has not been used often in the past or replace page that has smallest count. In LFU we check the old page as well as the frequency of that page and if the frequency of the page is larger than the old page we cannot remove it and if all the old pages are having same frequency then take FIFO method for that and remove that page.

Example:- Refrence sequence (7,0,1,2,0,3,0,4,2,3,0,3,2,1,2) with 3 frames.

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
F_1	7	7	7	2	2	2	2	4	4	3	3	3	3	1	1
F_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0
F_3			1	1	1	3	3	3	2	2	2	2	2	2	2
				H		H					H	H	H		H

Frequency table

Initially frequencies of all are set to 0

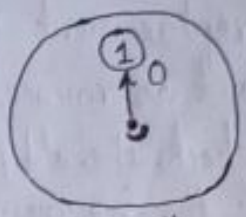
Refrence	frequencies					
7	0	1	0			
0	0	1	2	3	4	
1	0	1	0	1		
2	0	1	0	1	2	3
3	0	1	0	1	2	0
4	0	1	0			

6) Clock Page Replacement Algorithm:-

The clock algorithm keeps a circular list of pages in the memory, with the "hand" pointing to the last examined page frame in the list. When a page fault occurs and no empty frames exist, then the R (refrenced) bit is inspected at the hand's location. If R is 0, the new page is put in place of the page the "hand" points to, and the hand is advanced one position. Otherwise, the R bit is cleared, then the clock hand is incremented and the process is repeated until a page is replaced.

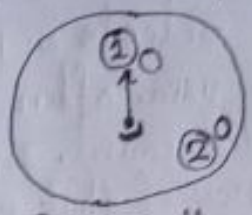
Example:- Input Refrences: {1,2,3,4,3,1,2,1,5,4}
Size of memory: 3

Upon accessing 1



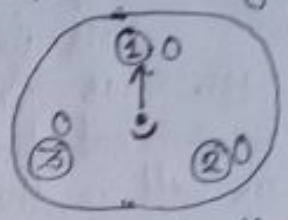
Page fault

Upon accessing 2



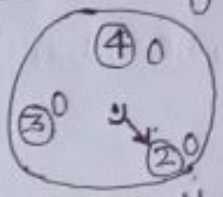
Page fault.

Upon accessing 3



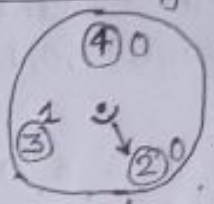
Page fault

Upon accessing 4



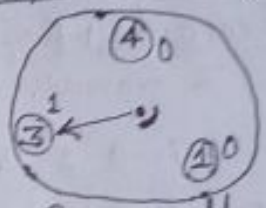
Page fault

Upon accessing 3



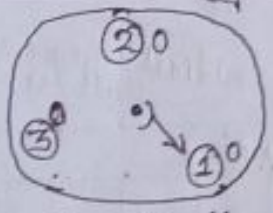
Hit

Upon accessing 1



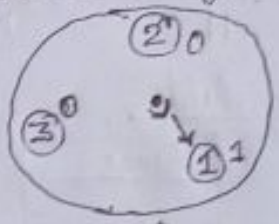
Page fault.

Upon accessing 2



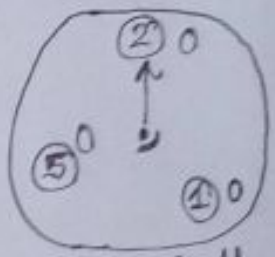
Page fault

Upon accessing 1



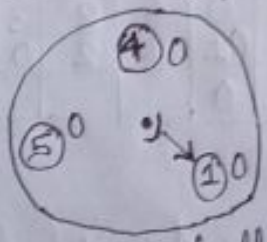
Hit.

Upon accessing 5



Page fault.

Upon accessing 4



Page fault.

Hence
total no. of page faults = 8
& total no. of page hits = 2.

7) WS-Clock Page Replacement Algorithm:

It is an improved form of clock page replacement algorithm. Lesser important in comparision to other. We can study this from youtube if we want.

[I have escaped this].

⊛. Locality of refrence:-

Locality of refrence is the tendency of a processor to access the same set of memory locations repetitively over a short period of time. Subroutines tend to localize the refrences to memory for fetching instructions, this is the reason for this property. There are two basic types of locality of refrence: temporal and spatial. Locality of refrence makes cache memory to work.

i) Temporal locality → The information which is used currently is likely to be in use in near future. For e.g, Reuse of information in loops.

ii) Spatial locality → If a word is accessed, adjacent (near) words are likely to be accessed soon. For eg, Related data items (arrays) are usually stored together, Instructions are executed sequentially.

Segmentation

Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process. The details about each segment are stored in a table called as segment table. Segment table is also stored in one (or many) of the segments. Segment table contains mainly two information about segments:

i) Base → It is the base address of the segment.
ii) Limit → It is the length of the segment.

⊛. Why segmentation is required ?

⇒ Paging is more close to operating system rather than the user. Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.

It is better to have segmentation which divides the process into the segments. Each segment contain same type of functions such as main function can be included in one

segment and the library functions can be included in the other segment.

Advantages of segmentation:

→ No internal fragmentation
→ Less overhead.
→ It is easier to reallocate segments than entire address space.
→ The segment table is of lesser size as compared to the page table in paging.

Disadvantages of segmentation:

→ It can have external fragmentation.
→ It is difficult to allocate contigious memory to variable sized partition.
→ Costly memory management algorithms.

Paging vs Segmentation

Paging	Segmentation.
i) Page is always of fixed block size.	i) Segment is of variable size.
ii) Paging may lead to internal fragmentation as page is of fixed size block.	ii) Segmentation may lead to external fragmentation as the memory is filled with the variable sized blocks.
iii) In paging the user only provides a single integer as the address which is divided by hardware ~~number~~ into a page number & offset.	iii) In segmentation the user specifies the address into two quantities ie, segment number and offset.
iv) The size of the page is decided or specified by the hardware.	iv) The size of the segment is specified by the user.

⊛. Segmentation with Paging (MULTICS):

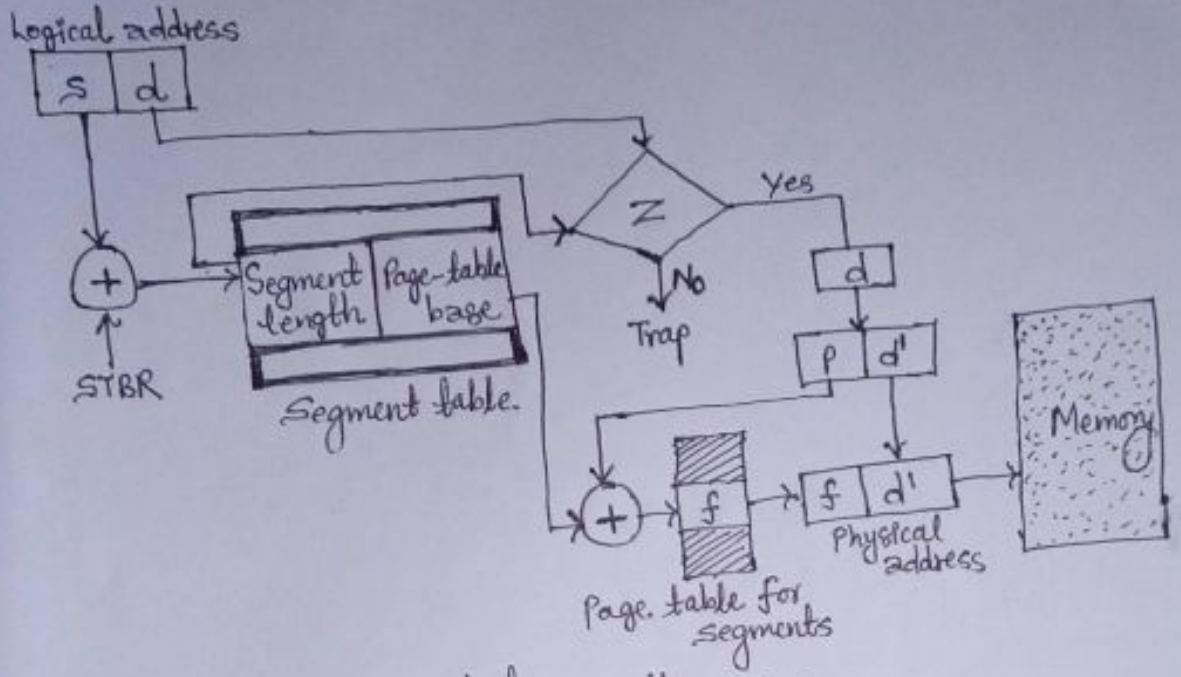


Fig: Segmentation with paging

→ The segment number is used to find segment descriptor.

→ A check is made if the segment's page table is in memory. If it is, it is located. If not, a segment fault occured.

→ The page table entry for the requested virtual page is examined. If the page itself is not in memory, a page fault was triggered. If it is in memory, the main-memory address of the start of the page is extracted from the page table entry.

→ The offset is added to the page origin to give the main memory address where the word is located.

→ The read or write finally takes place.

UNIT-5

File Management

1) File Overview:

A file is a named collection of related information namely records on a secondry storage device such as disk or tape. Commonly file represents programs and data. Data files may be numeric, alphanumeric or binary. Files are logical units of information created by process.

File naming: Files are logical units of information created by process. When a process creates a file, it gives the file a name. When the process terminates, the file continues to exist and can be accessed by other processes using its name. The exact rules for file naming vary somewhat from system to system, but all current operating systems allows strings of one to eight letters as legal file names. Frequently digits and special characters are also permitted and are often valid as well. Many file systems support as long as 255 characters.

The newer operating systems have a much more advanced native file system (NTFS) that has different properties. There is second file system for Windows 8 known as ReFS (Resilient File System), but it is targated at the server version of Windows 8.

File Structure:- Files can be structed in several ways. The most common structures are:

1) **Unstructured** → It consists of unstructured sequence of bytes or words. Os does not care what is in the file. Any meaning must be imposed by user level programs. It provides maximum facility user can put anything they want and name them in their own convinient way. Both UNIX and WINDOWS use this approach.

fig. unstructured file

ii) Record structured: It is a sequence of fixed length records each with some internal structure. Each read operation returns one record & write operation overwrites or append one record. Many old mainframe system use this structure.

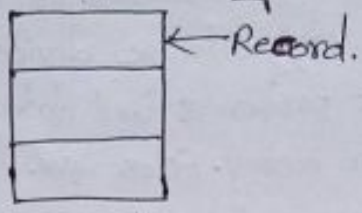


fig. Record structured

iii) Tree structured: It consists of tree of records, not necessarily all the same length. Each record containing a key field in a fixed position in the record and sorted on the key to allow the rapid searching. The operation is to get the record with the specific key. It is used in large mainframe systems for commercial data processing.

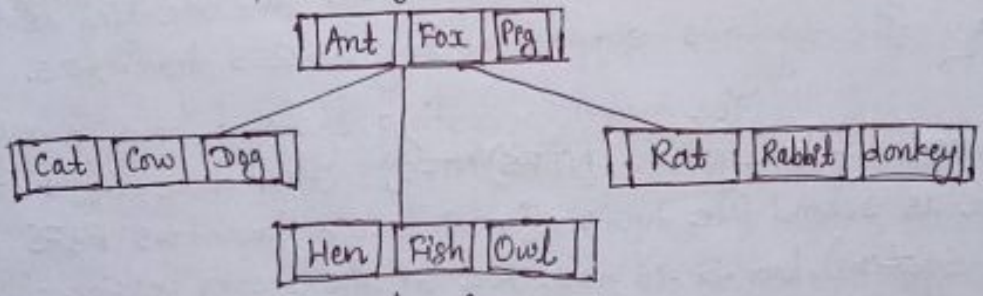


fig. tree structure

File Types:

i) Regular files → Contain user information, are generally ASCII or binary.

ii) Directories → System files for maintaining the structure of file system.

iii) Character special file → Related to I/O of computer to model serial I/O device such as terminals.

iv) ASCII files → Consists of line of text. They can be displayed and printed as it can be edited ~~by or~~ with ordinary text editor.

v) Binary files → Consists of sequence of bytes only. They have some internal structure known to program that use them. Executable files are its examples.

File Access:

i) Sequential file access → A sequential file is an ordered file. It is set of continiously stored records on a physical storage devices such as magnetic disk or CD ROM. To locate a particular record a program scan the file from the beginning until the desired record is located.

ii) Direct file access → File whose bytes or records can be read in any order is known as direct file access. It is based on the disk model of file, since disk allows random access to any block. Direct access is obtained by hashing method.

File Attributes: File attributes are settings associated with computer files that grant or deny certain rights to how a user or operating system can access that file. Some of the attributes of file are:-

Name → It is the only information which is in human-readable form.

Identifier → The file is identified by a unique tag (number) within file system.

Type → It is needed for systems that support different ~~file systems~~ types of files.

Location → Pointer to file location on device.

Size → The current size of the file.

Protection → This controls and assigns the power of reading, writing, executing.

Archeive → Tells Windows backup to backup the file.

File Operations:

i) Create → The file is created with no data. The purpose of the call is to announce that the file is comming and to set some of the attributes.

ii) Delete → When the file is no longer needed, it has to be deleted to free up disk space.

iii) Open → Before using a file, a process must open it. The purpose of the open call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.

iv) Close → When all the accesses are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up internal table space.

v) Read → Data are read from file.

vi) Write → Data are written to the file.

vii) Append → It can add data only to the end of the file.

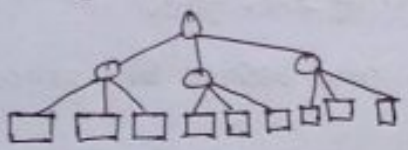
viii) Rename → It is used when the user needs to change the name of an existing file.

Directories:-

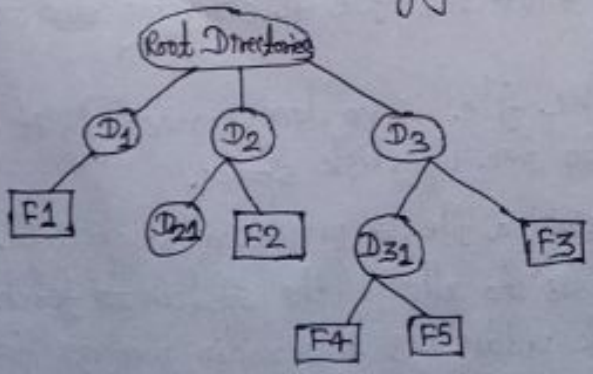
To keep track of files, file systems normally have directories or folders which are themselves files.

i) Single-level directory systems → The simplest form of directory system is having one directory containing all the files. Sometimes it is called the root directory, but since it is the only one, the name does not matter much.

ii) Two-level directory systems → It contains separate directory for each user. It is used on multi-user computer. It has problem when user want to co-operate on some task & to access one another files.



iii) Hierarchical directory systems → It is the generalization of two level structure to a tree of arbitary height. This allows the user to create their own sub-directories and to organize their file accordingly.



File-System Layout:

A file system is a set of files, directories and other structures. File systems maintain information and identify where a file or directory's data is located on the disk. In addition to files and directories, file systems contain a boot block, a super block, bitmaps and one or more allocation groups. An allocation group contains disk i-nodes and fragments. Each file system occupies one logical volume.

The boot block occupies the first 4096 bytes of the file system starting at byte offset 0 on the disk. The boot block is available to start the operating system. The superblock is 4096 bytes in size and starts at byte offset 4096 on the disk. The super-block maintains information about the entire file system.

2) Implementing Files:

Implementing file storage is keeping track of which disk blocks go with which file. Following are some of the methods.

a) Continious Allocation:

The simplest allocation scheme is to store each file as a contiguous run of disk blocks. Thus on a disk with 1-KB blocks, a 50-KB file would be allocated 50 consecutive blocks. With 2-KB blocks, it would be allocated 25 consecutive blocks. Each file begins at the start of a new block, so if some space is wasted then the wasted space is at the end of the last block.

Contiguous disk-space allocation has two significant advantages. First it is simple to implement & Second, the read performance is excellent. The disadvantages of contigious allocation are: for new files it is very difficult to find spaces, we can't extend the file and difficulty about fragmentation.

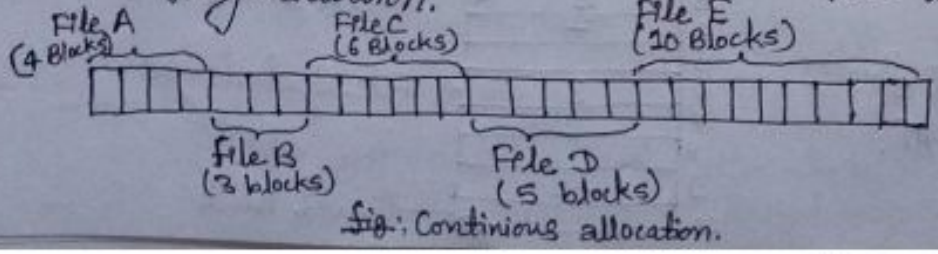


fig: Continious allocation.

b) Linked-List Allocation:

This method for storing files is to keep each one file as a linked list of blocks. The first word of each block is used as a pointer to the next one. The rest of the block is for data.

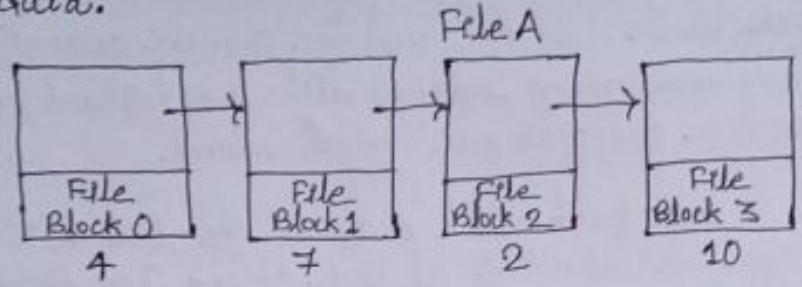


Fig: Storing file as a linked list of disk blocks.

Unlike contigious allocation, every disk block can be used in this method. Except for internal fragmentation in the last block no space is lost to disk fragmentation. To get to block n, the operating system has to start at the beginning and read the n-1 blocks prior to it, one at a time.

Problems: It solves all the problems of contigious allocation but it can be used only for sequential file access, random access is excessively slow. It also requires space for pointer. Each block access requires disk seek.

c) Linked-List Allocation using FAT (File Allocation Table / Table in Memory):

Using this organization, the entire block is available for data. The FAT is used as a linked list. The directory entry contains the block number of the first block of the file. The FAT is looked to find the next block until a special end of file value is reached.

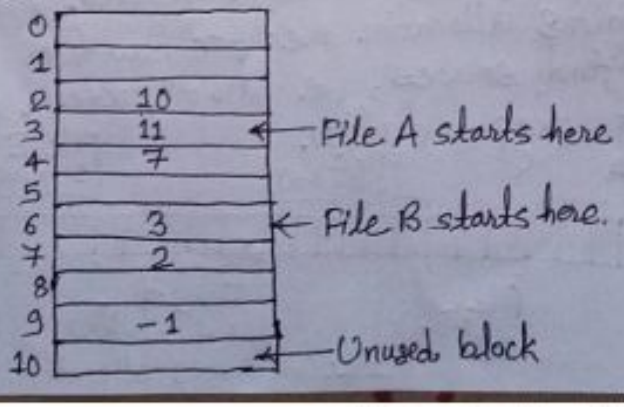


fig. Linked-list allocation using FAT.

Advantages:
→ Entire block is available for data.
→ Result the significant number of disk seek.
→ Random access time is included.

Problems: The entire table must be in memory all the time to make it work.
With a 1-TB disk and a 1-KB block size, the table needs 1 billion entries, one for each of the 1 billion disk blocks. Each entry has to be a minimum of 3 bytes. For speed in lookup, they should be 4 bytes. Thus the table will take up 3 GB or 2.4 GB of main memory all the time. Clearly the FAT idea does not scale well to large disks.

d) I-nodes: The method for keeping track of which blocks belong to which file is to associate with each file a data structure called an i-node (index-node), which lists the attributes and disk addresses of the file's blocks.

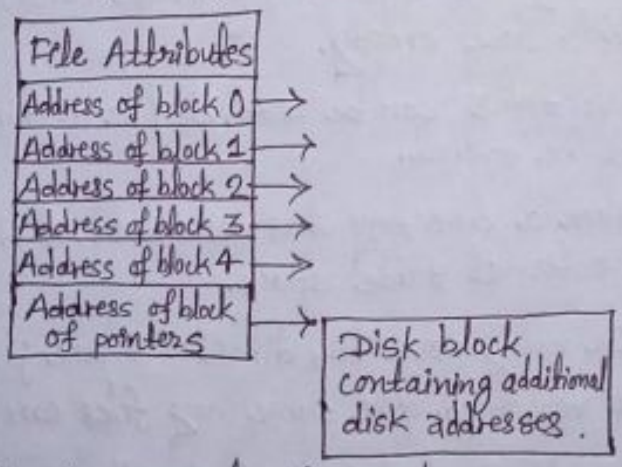


Fig: An example of i-node.

This array is usually far smaller than the space occupied by the file table. The i-node scheme requires an array in memory ~~size of~~ whose size is proportional to the maximum number of files that may be open at once. It does not matter if the disk is 100 GB, 1000 GB, or 10,000 GB.

Advantages:
→ This supports direct access to the blocks occupied by file so it provides fast access to file blocks.
→ It overcomes the problem of external fragmentation.

Disadvantages:

→ The pointer overhead for indexd allocation is greater than linked-list allocation.

→ For very small files, the indexd allocation would keep one entire block for the pointers which is inefficient in terms of memory utilization.

3) Directory:

Directory is a place/location where a set of file(s) will be stored. It is a folder which contains details about files, file size and time when they are created and last modified. To keep track of files, file systems normally have directories or folders which are themselves files.

Directory Operations:-

i) Create → A directory is created. It is empty except for dot and dotdot, which are put there automatically by the system.

ii) Delete → A directory is deleted. Only those directory can be deleted which are empty.

iii) Opendir → Directories can be read. Before a directory can be read, it must be opened.

iv) Closedir → When a directory has been read, it should be closed to free up internal table space.

v) Rename → In many respects, directories are just like files and can be renamed the same way files can be.

vi) Link → Linking is a technique that allows a file to appear in more than one directory.

vii) Unlink → A directory entry is removed.

viii) Readdir → This call returns the next entry in an open directory.

Path names: Each file and directory can be reached by a unique path, known as path name. The path name specifies the location of a file or directory within the file system. Path names cannot exceed 1023 characters in length. Following two different methods are commonly used for path names:

1) **Absolute path name** → In this method, each file is given an absolute path name consisting of the path from root directory to the file. Absolute path names always start at the root directory and are unique. Some programs need to access a specific file without regard to what the working directory is. In this case, we should always use absolute path names.

In UNIX the components of the path are separated by '/'. In Windows, the seperator is '\'.

Windows \usr\ast\mailbox.
UNIX /usr/ast/mailbox.

2) **Relative path name** → This is used in conjuction with the concept of working directory. A user can designate one directory as the current working directory, in which all path names not belonging at the root directory are taken relative to the working directory.

Directory Implementation:

There are 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 widely used.

1) **Linear List** → In this algorithm, all the files in the directory are maintained as singly linked list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

Characteristics:

→ 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.

→ The list needs to be traversed in case of every operation (creation, deletion, updating etc.) on the files therefore the systems become inefficient.

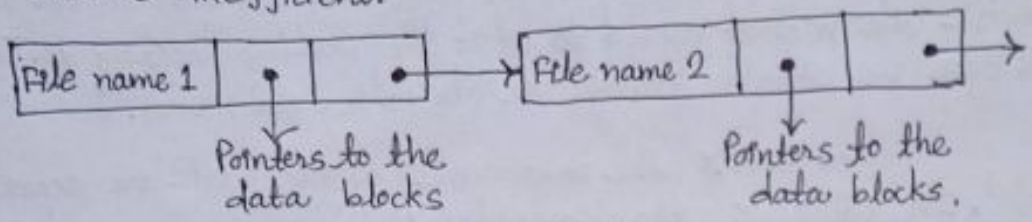
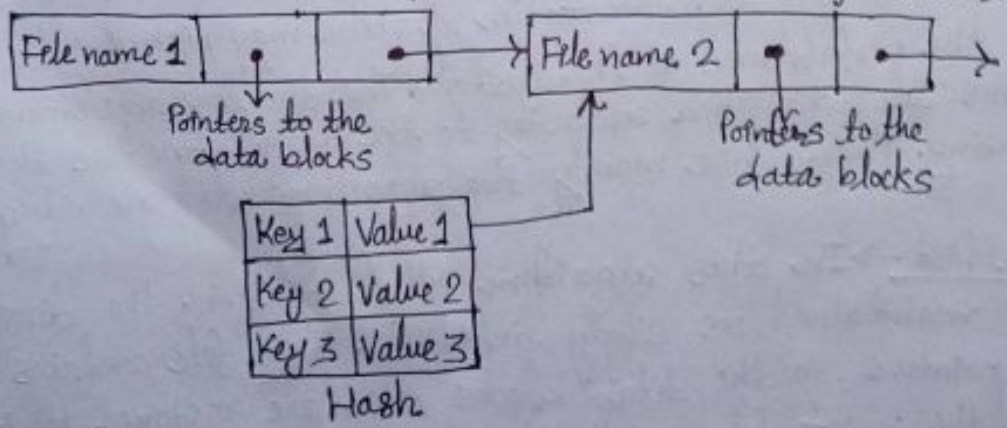


Fig: linear list.

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 it greatly decrease the file search time which is it's advantage. The problem with this method is that it has fixed size and dependence of the hash function on that size.



Shared Files:

When several users are working together on a project, they often need to share files. It is often convenient for a shared file to appear simultaneously in different directories belonging to different users. Thus it is better to represent file system by using directed acyclic graph (DAG) rather than tree structure as shown figure below;

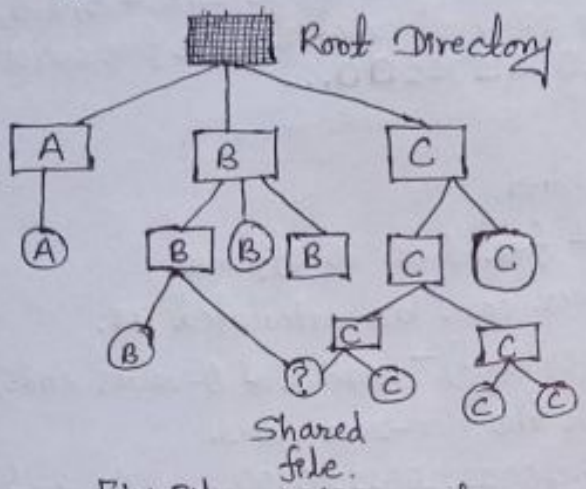


Fig: File system containing a shared file.

It is not good idea to make a copy of a file is being shared. If directories contain disk addresses problem may occur in synchronizing the change made to the file by multiple users. If one user appends new block in the file new block will be listed only in the directory of the user doing the append. Following two approaches can be used to solve this problem.

i) Directory entry that only points to i-nodes.
ii) Directory entry that points to link file.

4) Free Space Management:

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes very important. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly by Bitmaps and linked lists.

i) Bitmap or Bit vector → A Bitmap or Bit vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1, where 0 indicates that the block is allocated and 1 indicates a free block. Initially all the blocks are empty therefore each bit in the bit map vector contains 1. A disk with n blocks requires a bitmap with n bits.

Example: Consider a disk where block 2,3,4,5,8,9,10,12,13... are free, and rest are allocated. Then the free space bit map would be 0011110011101100...

(Since block starts from 0)

Advantages:

→ Simple to understand.

→ Finding the first free block is efficient.

Note: The block number can be calculated as:

(number of bits per word) * (number of 0-values words) + offset of bit first 1 in the non-zero word.

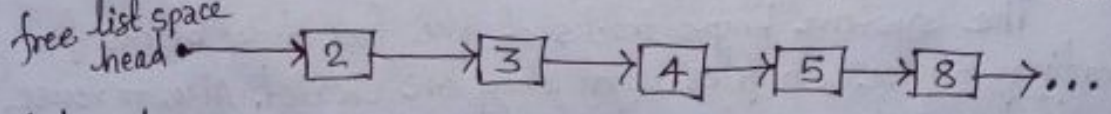
(A 0-valued word has all bits 0)

⇒ From above bitmap i.e, 0011110011101100..., The first group of 8 bits (00111100) constitute a non-zero word since all bits are not 0. Scanning from first 3rd bit is non-zero so, offset = 3.

Therefore, the first free block number = 8*0 + 3
= 3.

ii) Linked List → In this approach, the free disk blocks are linked together. i.e, a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

Example: Consider a disk where block 2,3,4,5,8 are free and rest are allocated then it can be represented in linked list as;



Advantage: Only one block is kept in the memory

Problem: Not efficient to traverse list, it must reach each block.

UNIT=6

Device Management:

Device management is an important function of the operating system. Device management is responsible for managing all the hardware devices of the computer system. It may also include the management of the storage device as well as the management of all the input and output devices of the computer system.

An operating system manages the devices in a computer system with the help of device controllers and device drivers. All these device controllers are connected with each other through a system bus. Device management generally performs the following:

→ Installing device and component-level drivers and related software.
→ Configuring a device so it performs as expected.
→ Implementing security measures and processes.

⊛. Classification of I/O devices:

i) Machine readable or Block devices → A block device is one with which the driver communicates by sending entire blocks of data. Each block has its own address and can be read from/write to independently. For example: Hard disks, USB cameras, Disk-On-Key, tapes, sensors etc.

ii) User readable or Character devices → A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). It accepts a stream of characters with no attention paid to block structure. It is not addressable and has no seek ability. For example: printers, graphical terminals, screen, keyboard, mouse, serial ports, sound cards etc.

iii) Communication devices → A communication device is a hardware device capable of transmitting an analog or digital signal over the telephone, or other communication wire, or wirelessly. The best example of a communication device is a Modem, which is capable of sending and receiving a signal to allow computers to communicate with other computers.

⊛. Device Controllers:

I/O units often consist of a mechanical component and an electronic component. The electronic component is called the device controller or adapter. A single controller can handle multiple devices; some devices have their own build-in controller. The controller has one or more registers for data and signals. The processor communicates with the controller by reading and writing bit patterns in these registers.

The controller's job is to convert the serial bit stream into a block of bytes and perform any error correction if necessary. The block of bytes is typically first assembled bit by bit, in a buffer inside the controller. After its checksum has been verified and the block has been declared to be error free, it can then be copied to main memory.

⊛. Memory-Mapped I/O:

Each control register is assigned a unique memory address to which no memory is assigned. This system is called memory-mapped I/O. In most systems, the assigned addresses are at the top of the address space or near the top of the address space.

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. Memory mapped I/O is used for high-speed I/O devices like disks, communication interfaces. CPU communicates with the control registers and the device data buffers in following three ways:

- → Using I/O port
- → Using memory mapped I/O
- → Using hybrid system.

The advantage of memory-mapped I/O is that it can be implemented in high level languages such as C and no separate protection is needed. The disadvantage is that it adds extra complexity to both hardware and OS.

⊛. DMA Operation:

Direct Memory Access (DMA) is a process for data transfer between memory and I/O, controlled by an external circuit called DMA controller, without the involvement of CPU. Most of the data that is input or output from computer is processed by the CPU, but some data does not require processing or can be processed by another device. In these situations, DMA can save processing time and is more efficient way to move data from the computers memory to other devices. For example; A PCI controller and a hard drive controller each have their own set of DMA channels.

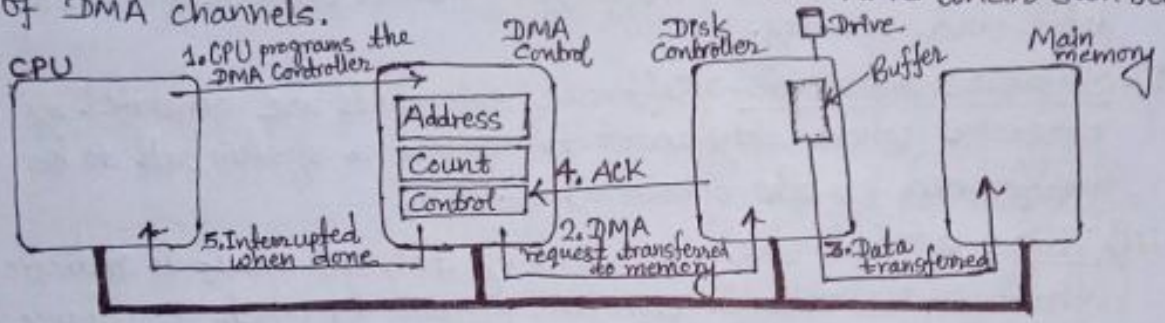


Fig: Operations of a DMA transfer

Working Procedure of DMA:

→ The CPU programs the DMA controller by its registers so it knows what to transfer where. When valid data are in disks controller buffer, DMA can begin.

→ The DMA controller initiates the transfer by issuing a read request over the bus to disk controller.

→ Data is transferred from disk controller to memory.

→ When data transfer is completed, the disk controller sends an acknowledgment signal to DMA controller. The DMA controller then increments the memory address to use and decrement the byte count. This continues until the byte count greater than 0.

→ When transfer is completed finally the DMA controller interrupt the CPU.

⊛ Interrupts:

The hardware mechanism that enables a device to notify the CPU is called an interrupt. Interrupt forces CPU to stop what it is doing and start doing something else. Interrupts are signals sent to the CPU by external devices, normally I/O devices. There are three types of interrupts:

i) Hardware interrupts → Hardware interrupts are generated by hardware devices to signal that they need some attention from the OS. They may have just received some data or they have just completed a task which the operating system provides requested, such as transferring the data between the hard drive and memory.

ii) Software interrupts → Software interrupts are generated by programs when they want to request a system call to be performed by the operating system.

iii) Traps → Traps are generated by the CPU itself to indicate that some error or condition occured for which assistance from the operating system is needed.

#IO Handling:

⊛ Goals of the I/O Software:

i) Device independence → A key concept in the design of I/O software is known as device independence. It means that we should be able to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a hard disk, a DVD, or on a USB stick without having to be modified for each different device.

ii) Uniform Naming → Uniform Naming, simply be a string or an integer and not depend on the device in any way. The name of file or device should be some specific string or number. It must not depend upon device in any way. All files and devices are addressed the same way: by a path name.

iii) Error handling → Generally, errors should be handled as close as possible to the computer hardware. It should try to correct the error itself if it can in case of the controller discovers a read error. And in case if it can't then the device driver should handle it.

iv) Synchronous (blocking) and Asynchronous (interrupt-driven) transfers →

Most physical I/O is asynchronous; however, some very high performance applications need to control all the details of the I/O, so some operating systems make asynchronous I/O available to them. User programs are much easier to write if the I/O operations are blocking.

v) Buffering → Sometimes data that come off a device can't be stored directly in its final destination. Some devices have several real-time constraints, so data must be put into output buffer in advance to decouple the rate at which the buffer is filled from the rate at which it is emptied, in order to avoid buffer underruns.

vi) Sharable and Dedicated devices → Some I/O devices such as disks, can be used by many users at the same time (ie, sharable). Other devices such as printers, have to be dedicated to a single user until that user is finished. The OS must be able to handle both shared and dedicated devices in a way that avoids problems.

⑧. Handling I/O:-

Fundamentally there are three different ways of performing I/O operations which are as follows:

i) Programmed I/O → It is the simplest form of I/O having the CPU do all the work. With programmed I/O, data are exchanged between the processor and the I/O module. The processor executes a program that ~~are exchanged between the processor and the I/O module.~~ gives it direct control of the I/O operation including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.

[Note: If Programmed I/O and Interrupt driven I/O are difficult then escape and read the differences between them, that is on next page and write in exam in a way asked.]

ii) Interrupt-Driven I/O → Interrupt driven I/O is a way of controlling I/O activity by a peripheral or terminal that needs to make or receive a data transfer sends a signal. This will cause a program interrupt to be set. This strategy allows the CPU to carry on with its other operations until the module is ready to transfer data.

iii) I/O using DMA → DMA uses an additional piece of hardware - a DMA controller. The DMA controller can take over the system bus and transfer data between an I/O module and memory without the involvement of CPU. Whenever the CPU wants to transfer data, it tells the DMA controller the direction of the transfer, the I/O module involved, the location of the data in the memory, and the size of the block of data to be transferred. It can then continue with other instructions and the DMA controller will interrupt it when the transfer is complete.

⊗ Differentiate between programmed I/O and Interrupt Driven I/O.

Programmed I/O	Interrupt Driven I/O.
i) In programmed I/O processor has to check each I/O devices in sequence and in effect ask each one if it needs communication with the processor.	i) In Interrupt driven I/O, processor does not have to check whether I/O device needs it service or not.
ii) In programmed I/O, processor is busy, so it decrements the system throughput.	ii) In Interrupt driven I/O, processor is not busy, so it increments the system throughput.
iii) It is implemented without interrupt hardware support.	iii) It is implemented using interrupt hardware support.
iv) It does not depend on interrupt status.	iv) Interrupt must be enabled to process interrup driven I/O.
v) It does not need initialization of stack.	v) It needs initialization of stack.
vi) System throughput decreases as number of I/O devices connected in the system increases.	vi) System throughput does not depend on the number of I/O devices connected in the system.

⊛ IO Software Layers:

IO software is organized in following four layers:
→ Interrupt handlers
→ Device drivers.
→ Device-independent OS software,
→ User level I/O software.

1) Interrupt Handlers → An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or a callback function in an OS, whose execution is triggered by the reception of an interrupt. When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt. Then it can unblock the driver that was waiting for it.

The interrupt mechanism accepts an address, which is a number that selects a specific interrupt handling function from a small set. In most architecture, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

2) Device Drivers → Device driver refers to a special kind of software program which controls a specific hardware device that enables different hardware devices for communication with the computer's OS. Without the required device driver, the corresponding hardware device fails to work. Device drivers are operating system specific and hardware dependent.

A device driver performs the following jobs:-
→ To accept request from the device independent software above to it.
→ Interact with the device controller to take and give I/O and perform required error handling.
→ Making sure that the request is executed successfully.

The main purpose of device drivers is to provide abstraction by acting as a translator between a hardware device and the applications that use it.

#Disk Management

⑧ Disk Structure:

A Disk is usually divided into Tracks, Cylinders and Sectors. Hard disk drives are organized as a concentric stack of disks or 'platters'. Each platter has 2 surfaces and two read/write heads for each surface. Each platter has the same no. of tracks. The speed of the disk is measured as two parts:

i) Transfer rate → This is the rate at which the data moves from disk to the computer.

ii) Random access time → It is the sum of the seek time and rotational latency.

Disk structure key terms:

i) Seek Time → Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

ii) Rotational latency → It is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads.

iii) Transfer Time → It is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

iv) Disk Access Time → Disk access time is given as;

✓ Disk Access Time = Rotational latency + Seek Time + Transfer Time

v) Disk Response Time → It is the average of time spent by a request waiting to perform its I/O operation.

vi) Transmission Time → The time taken to access the whole record is called transmission time.

Important Formulas for Numericals:

1) Disk capacity = surfaces * (tracks/surface) * (sectors/track) * (bytes/sector) [Unit = MB]

2) Data transfer rate = no. of rotations per second * (sectors/track) * (bytes/sector) * no. of surfaces. [Unit = MB/sec].

(RPM in 1 sec)

3) Average Access Time = 1/(RPM in 1 sec) * 2 [Unit = msec].

4) Average time to read a single sector = $T_s + \frac{1}{2r} + \frac{1}{r * \text{sectors per track}}$.

where, T_s = Avg. seek time
r = Rotational speed.

Note that T_s and r should be in seconds.

⇒ Expected time to read n continious sectors on the same track is given by;

$= (\text{average time to read single sector}) + \frac{1}{n*r} * (n-1)$ [Unit=ms]

⇒ Time to read n sectors scattered over the disk = (average time to read a single sector) * sectors per track.

5). Access time for n KB block = $T_s + \frac{1}{2r} + \frac{1024 * n}{r * N}$ [Unit=ms]

where, N = no. of bytes per track.

⊛. Disk Scheduling Algorithms:

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

i) Multiple I/O requests may arrive by different processes and only I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.

ii) Two or more request may be far from each other so can result in greater disk arm movement.

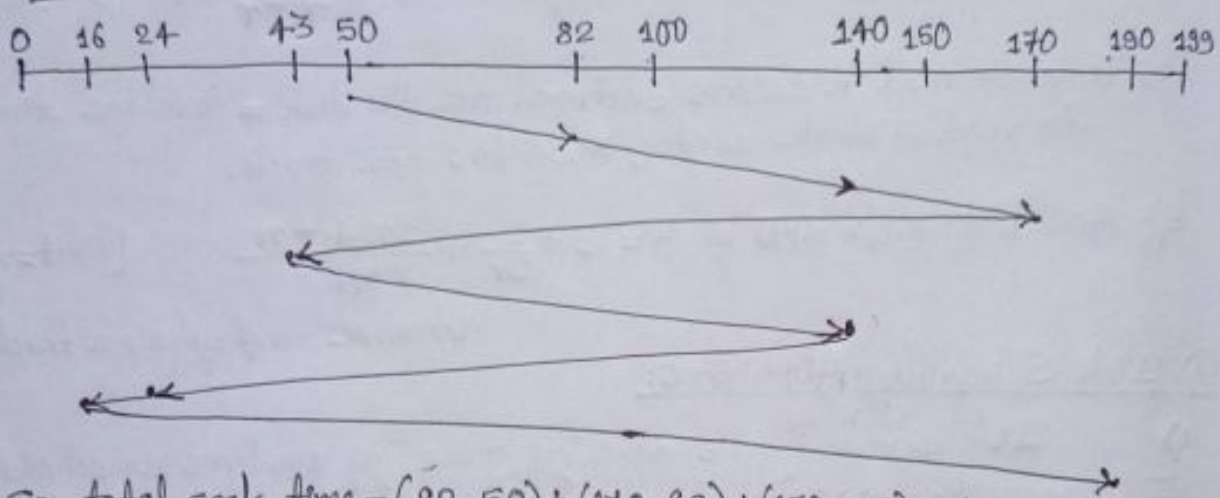
iii) Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

There are many disk scheduling algorithms which are as follows:-

a) First Come-First Serve (FCFS) → FCFS is the simplest of all the disk scheduling algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Since no reordering of request takes place the head may move almost randomly accross the surface of the disk.

Example :- Suppose the order of request is: (82,170,43,140,24,16,190) and current position of Read/Write head is 50. Then find the total head movement or total seek time, if it's tail track or end is at 199.

Solution:



So, total seek time = (82-50)+(170-82)+(170-43)+(140-43)
+(140-24)+(24-16)+(190-16)
= 642

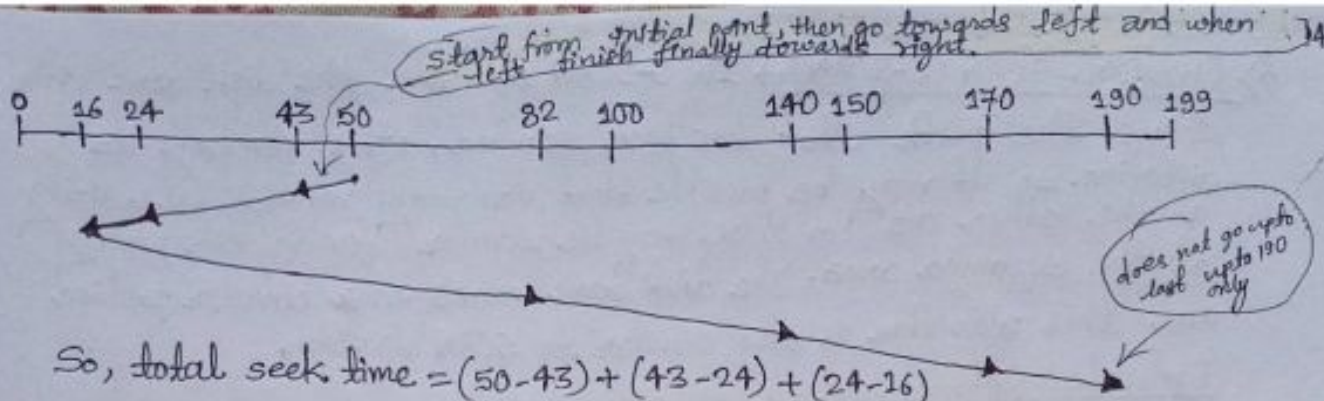
Advantages:
→ Every request gets a fair chance.
→ No indefinite postponement.

Disadvantages:
→ Does not try to optmize seek time.
→ May not provide the best possible service.

b) Shortest Seek Time First (SSTF) → In SSTF, requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in queue and then they are scheduled according to their calculated seek time. As a result the request near the disk arm will get executed first.

Example: (We will take same question of FCFS, since it will be easier and will also help to compare which one is better).

Solution:



So, total seek time = (50-43) + (43-24) + (24-16)
+ (82-16) + (140-82) + (170-40) + (190-170)
= 208

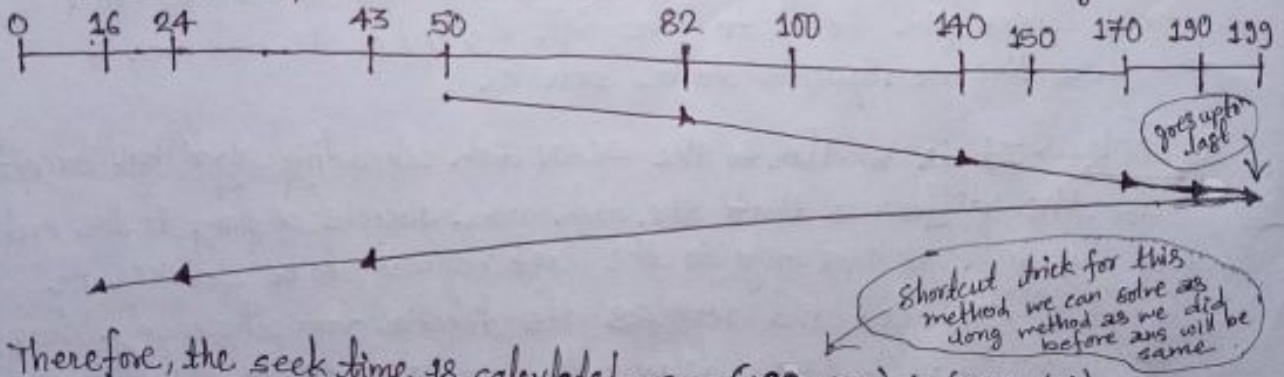
Advantages:
→ Average response time decreases.
→ Throughput increases.

Disadvantages:
→ Overhead to calculate seek time in advance.
→ Can cause starvation for a request if it has higher seek time as compared to incomming requests.
→ High variance of response time as SSTF favours only some requests.

c) Elevator (SCAN) → In SCAN algorithm the disk arm moves into a particular direction and services the requests comming in it's path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path.

Example: [We are taking same example]

Solution: Suppose that the disk arm should move "towards larger value".



Therefore, the seek time is calculated as = (199-50) + (199-16)
= 332

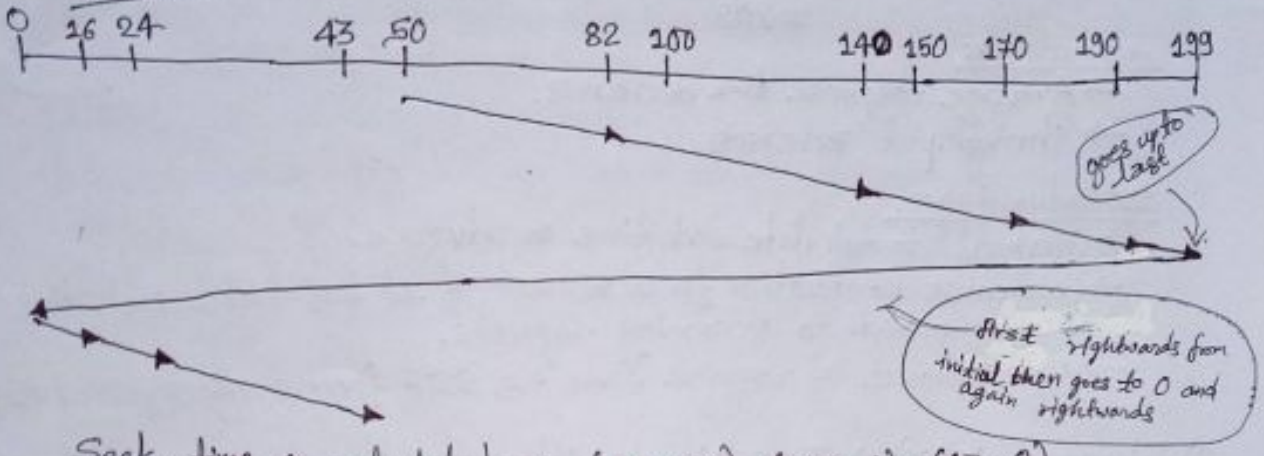
Advantages:
→ High throughput
→ Low variance of response time
→ Average response time.

Disadvantages:
→ Long waiting time for requests for locations just visited by disk arm.

d) Circular-SCAN (C-SCAN) → In C-SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area. The disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm.

Example: [Same example question].

Solution:



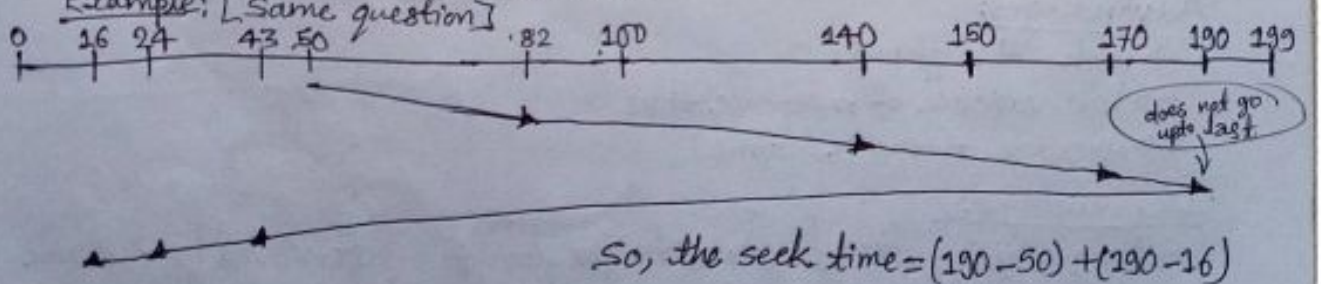
Seek time is calculated as = (199-50)+(199-0)+(43-0)
= 391

Advantages:
- → Provides uniform waiting time.
- → Provides better response time.

Disadvantages:
- → It causes more seek movements as compared to SCAN algorithm.
- → It causes the head to move till the end of the disk even if there are no requests to be serviced.

e) LOOK → It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm inspite of going to the end of the disk, it goes only to the last request to be serviced in front of the head and reverses its direction from there only. Thus it prevents the extra delay which occured due to the unnecessary traversal to the end of the disk.

Example: [Same question]



So, the seek time = (190-50)+(190-16)
= 314.

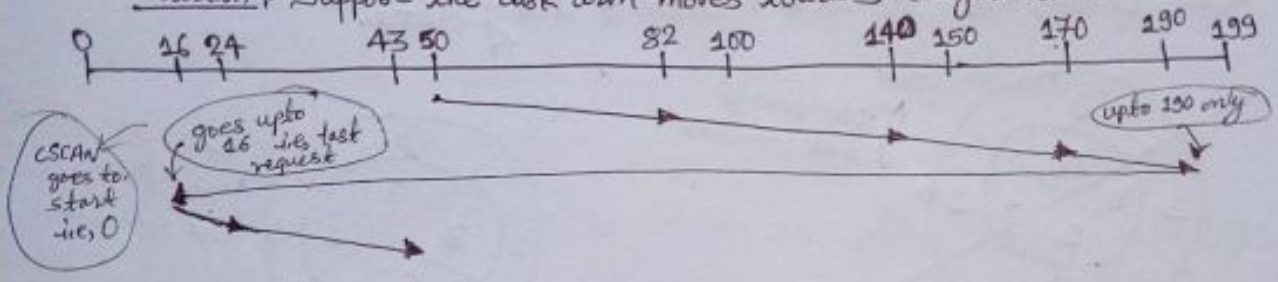
Advantages:
→ It provides better performance compared to SCAN.
→ It does not lead to starvation.

Disadvantages:
→ There is an overhead of finding the end requests.
→ It causes long waiting time for the cylinders just visited by the head.

f) C-LOOK → Circular-LOOK is an improved version of LOOK algorithm. It is similar to CSCAN disk scheduling algorithm. The difference is that the disk arm in spite of going to the end goes only to the last request to be serviced.

Example: [Same example question].

Solution:- Suppose the disk arm moves towards larger value.



So, the seek time = (190-50) + (196-16) + (43-16)
= 341

Advantages:
→ It provides better performance compared to CSCAN
→ It does not lead to starvation.

Disadvantage:
→ There is an overhead of finding the end requests.

⊛. Disk Formatting:

Disk formatting is a process to configure the data-storage devices such as hard-drive, floppy disk and flash drive when we are going to use them for the very first time. Disk formatting is usually required when new operating system is going to be used by the user. It is also done when there is space issue and we require additional space for the storage of more data in the drives. When we format the disk then the existing files within the disk are also erased. Disk formatting has also the capability to erase the bad applications and various sophisticated viruses. The frequent formatting of disk decreases the life of hard-drive.

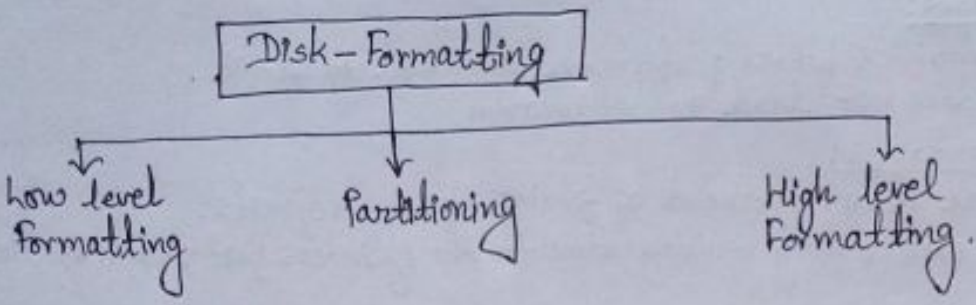


Fig: Formatting process of disk.

a) Cylinder Skew → If the position of sector s_i on each track is offset from the previous track then such an offset is called cylinder skew. It allows the disk to read multiple tracks in one continious operation without losing data.

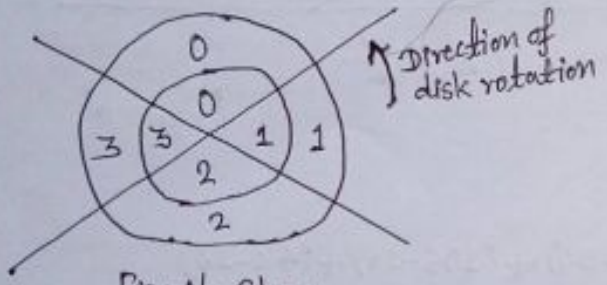


Fig: No Skew.

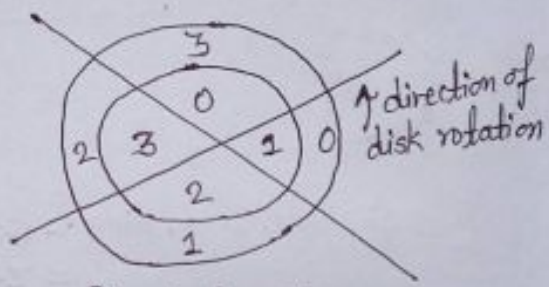


Fig: Sector Skew.

b) Interleaving → Interleave is the process through which gaps are placed between two sectors on the platter of a disk. This was done in earlier days when the computers were not quick enough to read continious streams of data.

Without interleaving there would be no gaps between the sectors and data would arrive immediately before the reading unit is ready. Due to this to read the same data a complete rotation of the disk would be required again. The interleaving ratio was not fixed and can be set by the end user depending on their system specs. Nowdays the ratio of interleaving is 1:1 ie, no interleaving is used.

c) Error Handling → Manufacturing defects introduce bad sectors; If defect is small (say few bits). Then the hard drive is shipped and ECC corrects the error every time the sector is accessed. If error is bigger it cannot be masked. There are two ways for error correction:

i) When one of the sectors is bad the controller remaps between the bad and spare sectors.
ii) If controller cannot remap, the operating system does the same thing in software.

⊛. Redundant Array of Independent Disks (RAID):

RAID is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both. It is the way of storing the same data in different places on multiple hard disks to protect data in the case of a drive failure. RAID works by placing data on multiple disks and allowing I/O operations to overlap in a balanced way, improving performance.

Stripping in RAID → The process of dividing large data into multiple disks for faster access is called stripping in RAID.

Mirroring in RAID → It is a mechanism in which the same data is written to another disk drive.

Parity in RAID → It is a method used to rebuild data in case of failure of one of the disks.

Hot Spares in RAID → Hot Spare is an extra drive added to the disk array, to increase the fault tolerance.

There are various RAID levels that can be used. Selecting the suitable raid level for our application depend on following things:

→ We can select a raid level based on the performance that it provides.
→ ~~RAID~~ RAID level based on the level of redundancy it provides.
→ RAID level based on read and write operations.

UNIT-7

Linux Case Study

LINUX is one of popular version of UNIX Operating System. It is open source as its source code is freely available and it is free to use. Linux is one of the most reliable, secure and worry-free operating system available. Linux is generally far less vulnerable to ransomware, malware, or viruses. Linux has a number of different versions to suit any type of user. LINUX MINT, MANJARO, DEBIAN, UBUNTU, SOLUS etc. are popular linux distributions.

History: The History of LINUX began in the 1991 with the beginning of a personal project by a Finland student Linus Torvalds to create a new free operating system kernel. Since then, the resulting LINUX Kernel has been marked by constant growth throughout the history.

→ In the year 1991, LINUX was introduced by a Finland student Linus Torvalds.
→ In the year 1992, Hewlett Packard 9.0 was released.
→ In the year 1993, NetBSD 0.8 and FreeBSD 1.0 released.
→ In the year 1994, Red Hat LINUX was introduced.
→ In the year 1995, FreeBSD 2.0 and HP UX 10.0 were released.
→ In the year 1997, HP-UX 11.0 was released.
→ In the year 1998, the fifth generation of SGI Unix was released.
→ In the year 2001, Linus Torvalds released LINUX 2.4
→ In the year 2004, Ubuntu was released.
→ In the year 2006, Oracle released its own distribution of Red Hat.
→ In the year 2011, LINUX kernel 3.0 versions were released.
→ In the year 2013, Google LINUX based Android claimed 75% of the smartphone market share, in terms of no of phones shipped.
→ In the year 2014, Ubuntu claimed 22,000,000 users.

Kernel Modules: The LINUX kernel is a monolithic kernel i.e, it is one single large program where all the functional components of the kernel have access to all of its internal data structures and routines. A kernel module is an object file that contains code that can extend the kernel functionality at runtime. When a kernel module is no longer needed it can be unloaded. Most of

the device drivers are used in the form of kernel modules. Kernel modules are usually stored in the /lib/modules subdirectories. Kernel Modules includes following:

1) Applications and OS services → These are the user application running on the LINUX system. OS services include utilities and services like shells, libraries, compilers etc.

2) LINUX Kernel → Kernel abstracts the hardware to the upper layers. It mediates and controls access to system resources.

3) Hardware → This layer consists of the physical resources of the system that finally do the actual work. It includes the CPU, the hard disk, system RAM etc.

⊛. Process Management:

A process refers to a program in execution; it's a running instance of a program. A process generally takes an input, processes it and gives the appropriate output. Tuning or controlling a process is called Process Management. In LINUX two vectors define a process: argument vector and environment vector. The argument vector has the command line arguments used by the process. The environment vector has a (name, value) list where different environment variable values are specified. There are two types of processes in Linux:

1) Foreground process → By default, all the processes run in the foreground. When a process is run in foreground, no other process can be run on the same terminal until the process is finished or killed. When issuing this type of process, the system receives input from the keyboard (stdin) and gives output to the screen (stdout).

2) Background process → Adding '&' to a foreground command makes it a background process. A background process runs on its own without input from the keyboard (stdin) and waits for input from the keyboard. While the process runs in the background, other processes can be run in the foreground. The background process will be in stop state till input from the keyboard

is given (usually 'Enter' key), then becomes a foreground process and gets executed. Only after the background process becomes a foreground process, that process gets completed else it will be a stop state.

⊛ **Linux Scheduling:-** Linux scheduling is based on the time sharing technique: Several processes run in "time multiplexing" because the CPU time is divided into slices, one for each runnable process. If a currently running process is not terminated when its time slice or quantum expires, a process switch may take place. Time sharing relies on timer interrupts and thus transparent to processes. No additional code needs to be inserted in the programs to ensure CPU time sharing. The schedular always succeeds in finding a process to be executed. Every Linux process is always scheduled according to one of the following scheduling classes;

SCHED_FIFO → It is a First-In, First-Out real-time process. When the schedular assigns the CPU to the process, it leaves the process descriptor in it's current position in the run queue list. If no other higher-priority real-time process is runnable, the process continues to use the CPU as long as it wishes, even if other real-time processes that have the same priority are runnable.

SCHED_RR → It is a Round Robin real-time process. When the schedular assigns the CPU to the process, it puts the process descriptor at the end of the run queue list. This policy ensures a fair assignment of CPU time to all SCHED_RR real-time processes that have the same priority.

SCHED_NORMAL → It is a conventional, time-shared process.

⊛. Inter-process Communication:

Inter-Process Communication (IPC) refers to a mechanism, where the operating systems allow various processes to communicate with each other. This involves synchronizing their actions and managing shared data. For inter-process communication LINUX has three main components:

i) Module Management → For new modules this is done at two levels- the management of kernel refrenced symbols and the management of the code in kernel memory. The LINUX kernel maintains a symbol table and symbols defined here can be exported explicitly. The module management system also defines all the required communication interfaces for newly inserted module. With this done, processes can request the services from this module.

ii) Driver registration → Usually the registeration of drivers is maintained in a registeration table of the module. The registeration of drivers contains the following:

→ Driver context identification as a bulk device or network driver.
→ File system context to store files in LINUX virtual file system or network file system like NFS.
→ Network protocols and packet filtering rules.
→ File formats for executable and other files.

iii) Conflict Resolution → The PC hardware configuration is supported by a large number of chip set configurations and with a large range of drivers for SCSI devices, video display devices and adaptors, network cards. This results in the situation where we have module device drivers which vary over a very wide range of capibilities and options. This necessitates a conflict resolution mechanism to resolve accesses in a variety of conflicting concurrent accesses. The conflict resolution mechanisms help in preventing modules from having an access conflict.

⊛. Memory Management:-

Following are the two major components in LINUX memory management:

1) The page management → Pages are usually of a size which is a power of 2. LINUX allocates a group of pages using a buddy system. "Page allocator" software is responsible for both allocation, as well as freeing the memory. The basic memory allocator uses a buddy heap which allocates contiguous are of size 2^n > the required memory with minimum n obtained by successive generation of "buddies" of equal size.

Example: Suppose we need memory of size 1556 words. Starting with a memory size 16K we would precede as follows:
→ First create 2 buddies of size 8K from the given memory size i.e, 16K
→ From one of the 8K buddy create two buddies of size 4K each.
→ From one of the 4K buddy create two buddies of size 2K each.
→ Use one of the most recently generated buddies to accommodate the 1556 size memory requirement.

i.e, 2000 word size

#Note that for a requirement of 1556 words, memory chunk of size 2K words satisfies the property of being the smallest chunk larger than the required size.

2) Virtual Memory Management → Linux supports virtual memory, that is, using a disk as an extension of RAM so that the effective size of the usable memory grows correspondingly. The kernel will write the contents of a currently unused block of memory to the hard disk so that the memory can be used for another purpose. When the original contents are needed again, they are read back into memory. Reading and writing the hard disk is slower than the using real memory, so programs don't run as fast it should. The part of hard disk that is used as virtual memory is called the swap space. Linux can also use normal file system or a seperate partition for swap space. A swap partition is faster, so we can go for swap partition if needed.

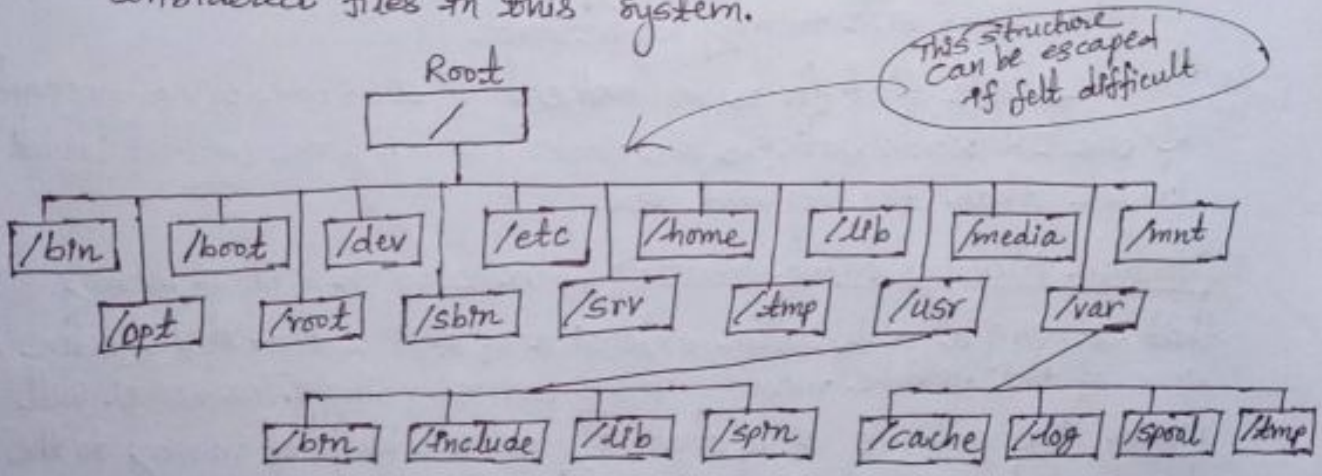
⊛. File System Management:

Linux file system management refers to how linux-based computers organize, store and track system files. The file system is basically a combination of directories or folders that serve as a placeholder for addresses of other files. There is no distinction between a file and a directory in Linux file system, because a directory is considered to be a file containing names of other files. In Linux, all the files and directories are located in a tree-like structure. Linux has three types of files:

i) Regular Files → It includes files like text files, images, binary files etc. Such files can be created using the touch command.

ii) Directories → Windows call these directories as folders. The root directory (/) is the base of the system. We could create new directories with mkdir command.

iii) Special Files → It includes physical devices such as a printer which is used for I/O operations. I/O devices are also considered files in this system.



⇒ The list below provides a short overview of the most important high-level directories on a linux system.

Directory	Contents
/	→ Root directory which is the starting point of the directory tree.
/bin	→ Essential binary files, such as commands that are needed by both the system administrator and normal users.

/boot → Static files of the boot loader.
/dev → Files needed to access host specific devices.
/etc → Host-specific configuration files.
/lib → Essential shared libraries and kernel modules.
/media → Mount points for removable media.
/mnt → Mount point for temporarily mounting a file system.
/opt → Add-on application software packages.
/root → Home directory for the superuser root.
/sbin → Essential system binaries.
/srv → Data for services provided by the system.
/tmp → temporary files.
/usr → Secondry hierarchy with read-only data.
/var → Variable data such as log files.

⊛. Device Management:

Linux device management includes the management of I/O and other hardware devices. Modern Linux distributions are capable of identifying a hardware component which is plugged into an already-running system. There are a lot of user-friendly distributions like Ubuntu, which will automatically run specific applications like Rhythmbox when a portable device like an iPod is plugged into the system.

The process of inserting devices into a running system is achieved in a Linux distribution by a combination of three components:- Udev, HAL and Dbus. Udev creates or removes the device node files in the /dev directory as they are plugged in or taken out. The HAL gets information from the Udev ~~device~~ service, when a device is attached to the system and it creates a XML representation of that device. Dbus is like a system bus which is used for inter-process communication.