Operating Systems

M.Sc. (CA)- 104

SELF LEARNING MATERIAL

DIRECTORATE OF DISTANCE EDUCATION

SWAMI VIVEKANAND SUBHARTI UNIVERSITY
MEERUT – 250 005,
UTTAR PRADESH (INDIA)

SLM Module Developed By :

Author:

Reviewed by :

1.

2.

Assessed by:

Study Material Assessment Committee, as per the SVSU ordinance No. VI (2)

Published by: Gayatri Sales

Typeset at: Micron Computers **Printed at:** Gayatri Sales, Meerut.

OPERATING SYSTEM

(M.Sc.(CA)-104

Unit - I

Operating Systems and Resource Manager, Operating system classifications, simple monitor, multiprogramming, timesharing, real time systems, multiprocessor systems, operating systems services.

Unit - II

File System : File supports, access methods, allocation methods-contiguous linked and index allocation; directory systems single level, tree-structure, a cyclic graph and general graph directory, file protection.

Unit - III

CPU Scheduling: Basic scheduling concepts, Process overviews, process states, multiprogramming, Schedulers, and Scheduling algorithms, multiple- processor scheduling.

Unit - IV

Memory Management: Bare machine approach, resident monitor, Partition, Paging and segmentation, virtual memory, demand paging.,Deadlocks: Deadlock Characterizations, deadlock prevention, avoidance detection and recovery.

Unit - V

Resource Protections : Mechanisms, Policies & domain of protection, Access matrix and its implementation, dynamic protection structures.Case Study of Windows-NT: Design Principle; System components, Environment subsystem; File System, Programmer Interface.

UNIT 1

OVERVIEW OF AN OPERATING SYSTEM

STRUCTURE

1.0 LEARNING OBJECTIVES

After reading this unit, you will be able to:

• explain you idea about. Software organization

• know about computer Hardware and software

• define software and operating systems

• explain the operating systems Architecture

• explain the various function of operating system.

1.1 SOFTWARE ORGANISATION

Computer Hardware and Software

Digital computers are comprised of hardware (equipment) and software (instriictions that make the equipment operate).

Computer. hardware consists of the following electronic or' electromechanical device

• Memory: a collection of registers and storage devices that store the computer's state map.

• Central Process mg Unit (CPU): the "lvai).)s" of the 'computer, which does the work (S. changing the state map stored in memory.

• Input/Output (I/O) Processor: manages and performs the work associated with reading (or writing) information that is added to (subtracted or copied from) portions of the computer's state map.

• Peripherals: include (a), devices that store ancillary software and data, (b) output devices such as printers and plotters, input devices such as a mouse or scanner, as well as the keyboard or display device (e.g., the monitor). The following schematic illustration depicts a typical arrangement of hardware in a personal computer or basic workstation with a sequential proc

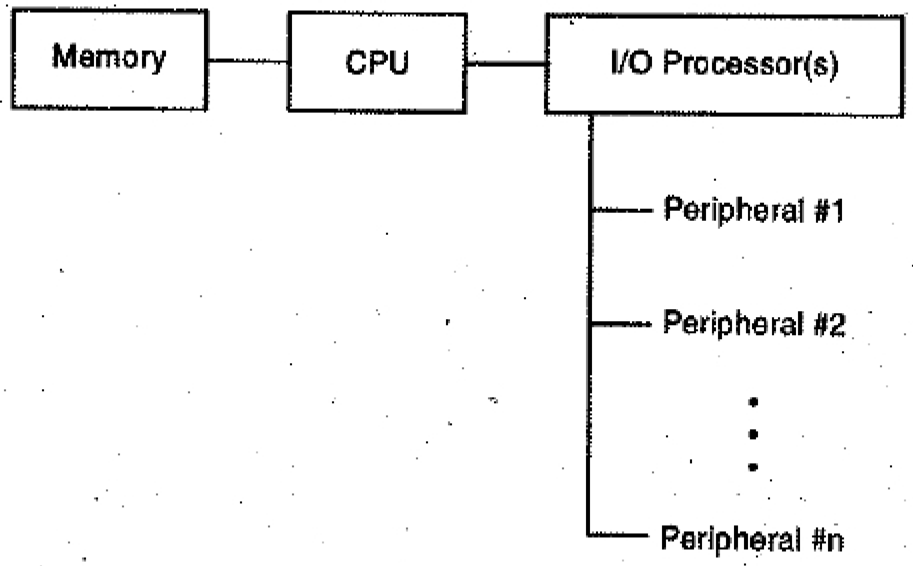


Figure 1.1. Organization of computer hardware in a von Neumann architecture. Such an arrangement is frequently called a von Neumann architecture (VNA), so named for John von Neumann, who helped develop this method of connecting computer components. The VNA currently comprises over 95 per cent of computer processors currently in use.

Software and Operating Systems

Computer software includes sets of instructions (also called programs) such as:

- Microcode: low-level instructions that make the CPU run correctly.

• Machine Code: slightly higher-level . than microcode, these instructions. are loaded directly into memory and comprise a stored, executable program.

• I/O and Computational Libraries: sets of procedures and instruc-tions that the I/O processor and CPU use to input or output data, as well as compute various arithmetic, math, and data handling operations.

• Application Software: programs that users run to accomplish various home, business, and scientific tasks such as word. processing (e.g., Word Perfect® or Microsoft Wore), spreadsheets (e.g., Quattro® or Excel®), drawing or sketching (e.g., AutoCAD® or Corel-Draw®), and graphics (e.g., Adobe Photoshop8).

• Operating System: interfaces application software with libraries and (in very few cases) microcode in a convenient manner that is transparent to (i.e., unseen by) the user.

• Ancillary Utilities: in addition to the utilities provided by the computer's operating system, there may be available to the user compilers, linkers, and loaders for programming languages such as Pascal, FORTRAN, C, and C++.

The following illustration schematically depicts the arrangement of software in a program development sequence.

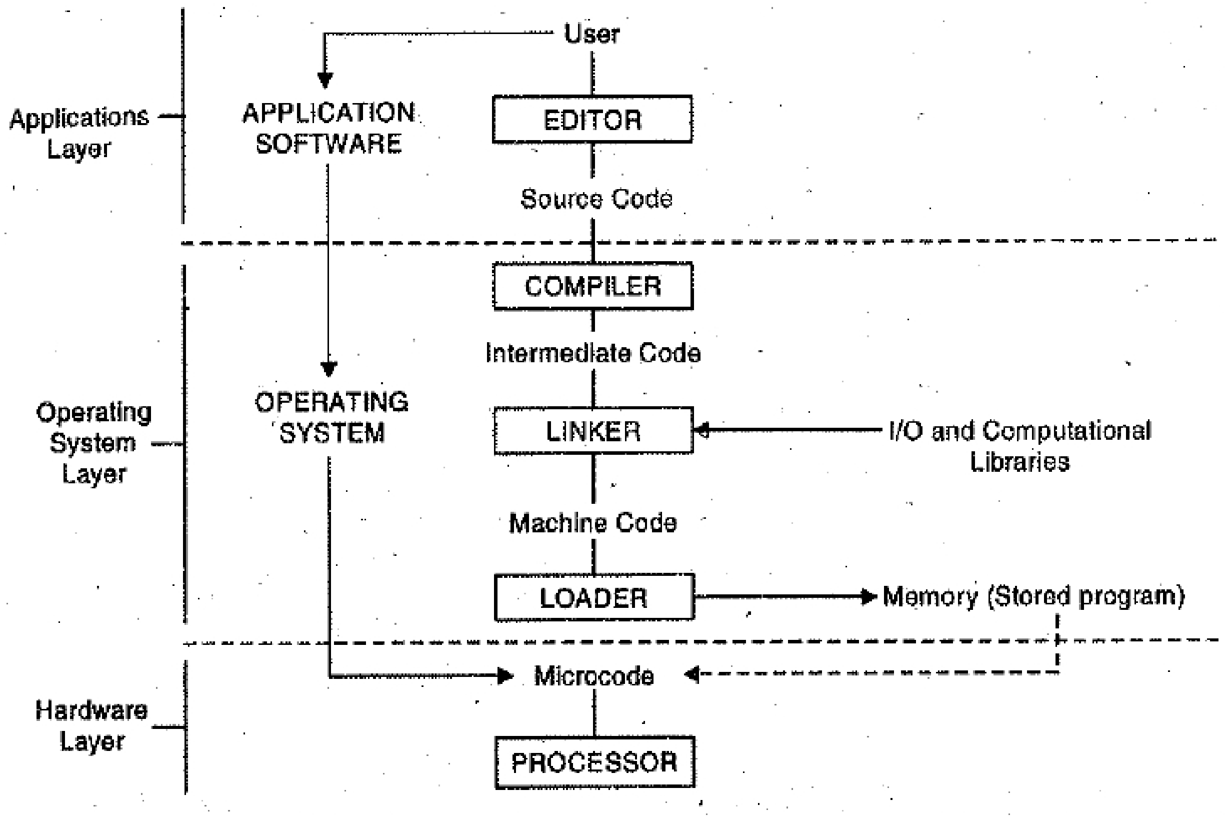


Figure 1.2. Organization of computer software for coinpilation and applications software execution: Light solid lines denote flow of data, while light dotted lines denote flow of control.

Of particular interest is the operating system, which performs a variety

of functions that include:

• Timekeeping: interrogates the system clock (hardware) to determine time-of-day and date as well as the duration of each sequence of instructions executed. •

• Input / Output: whereby instructions or data are moved to or from memory or secondary storage (e.g., a disk drive). •

• User Interface and I/O Management: which includes managing the layout and. display parameters (e.g., color and resolution) of windows in which applicatiOn software • is run, • inputting instructions from pointing devices such as. the mouse, and managing user input from. the keyboard.

- Compilation fifcinagement: including tasks that constitute portions of the software developMent process, such as (a) editing programs developed by system users, (b) program translation into low-level machine code, and (c) linking user programs with various libraries to achieve versatility. This process is shown schematically in

Figure 1.2.

File and Directory Management: organizing information stored in disk and tape drives in an orderly, hierarchical fashion.

Note that the operating system serves as a link between the lbw-level functions of the hardware and the higher-level processes that occur in application software. This linking process must be accomplished in a manner that is transparent to (i.e., unseen by) the user, so that the user will not waste time or cause problems by becoming involved in low-level implernentational details of program compilation and execution. Additionally, the operating syStern must perform these functions in a convenient manner, to avoid distracting the user. from performing tasks that the application software is designed to perform or assist.

In the past (i.e., early days of computing), operating systems were more cumbersome and provided fewer utilities. This was due in part to the small amount of memory available (resulting from relatively primitive technology), as well as the simpler types of software in, use at that time. Modern operating systems support graphical user interfaces, multi-window systems, and multiprocessing (running several different programs at the same time)..

1.2 OPERATING SYSTEMS CLASSIFICATION.

The variations and differences in the nature of different operating systems may give the impression that all operating systems are absolutely different from each-other. But thiS is not true. All operating systems contain the same components whose functionalities are almost the same. For instance, all the operating systems perform the functions of storage management, process management, protection of users from one-another, etc. The procedures and methods that are used to perform these functions might be different but the fundamental concepts behind these techniques are just the same. Operating systems in general, perform similar functions but may have distinguishing. features. Therefore, they can be classified into different categories on different bases. Let us quickly look at the different types of operating system's.

Single User-Single Processing System

The simplest all the computer systems is a single use-single processor system. It has a single processor, runs a single program and interacts with a single user at a time. The operating system for this system is very simple to design and implement. However, the CPU is not utilized to its full potential, because it sits idle for most of the time.

Figure 1.3.

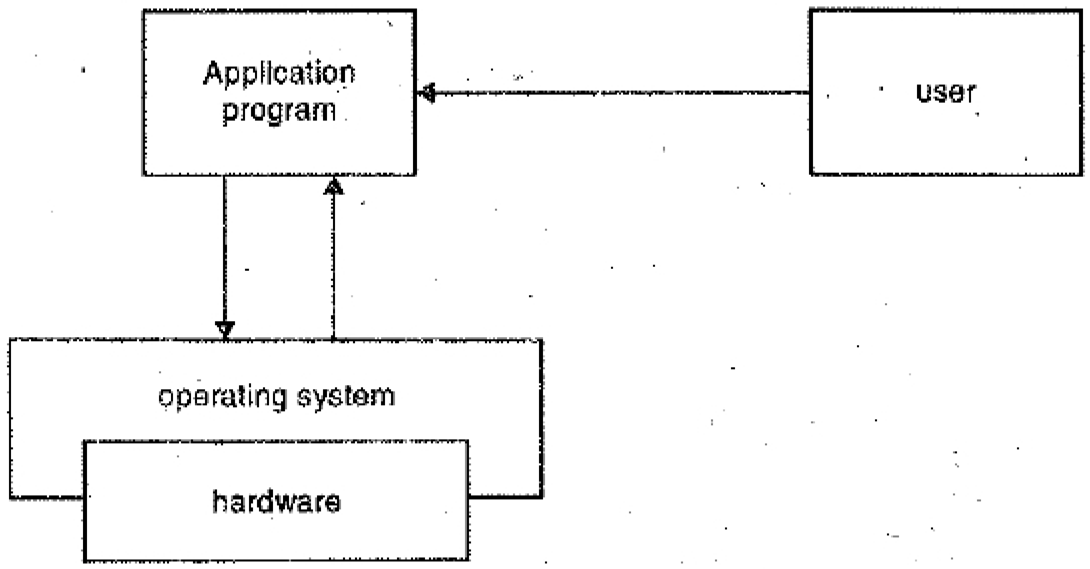


Figure 1.3. Single user-single processor system.

In this configuration, all the computing resources are available to the user all the time. Therefore, operating system has very simple responsibility. A representative example of this. category of operating system is MS-DOS.

Batch Processing Systems

The main function of a batch processing system is to automatically keep executing one job to the. next job in the batch (Figure 1.4). The main idea behind a batch processing system is to reduce the interference of the operator during the processing or execution of jobs by the computer. All functions of a batch processing system are 'carried out by the batch monitor. The batch monitor permanently resides in the low end of the main store: The current jobs out of the whole batch are executed in the remaining storage area. In other words, a. batch monitor is responsible for controlling all the environment of the system operation. The batch monitor accepts batch initiation commands- from the operator, processes a job, performs the job of job termination and batch termination, In a batch processing System, we generally make use of the term 'turn around time'. It is defined as the time from which a user job is given to the time when its output is given back to the user. This time includes the batch formation time, time taken to -execute a batch, time taken to print results and the time required to physically sort the printed outputs' that belong o different jobs. As the printing. and sorting of the results is done for all the jobs of batch together, the turn around time for a job , becomes the function of the execution time requirement of all jobs in the batch. You can reduce the turn around time for different jobs by recording the jobs or faster, input output media like magnetic tape or disk surfaces. takes very less tiny.! to read a record from these media. For inbtanco, if takes rotmd abiiut. five milliseconds for a magnetic 'tape' and about one millisecond for a fast fixed-head disk in comparison to a card reader or printer that takes around 50-100 milliseconds. Thus, if you use a disk or tape, it reduces the amount of time the central processor has to wait for are- input output operation to finish before resuming processing. This would reduce the time taken to process a job which indirectly would bring down the turn-around times for all the jobs in the batch.

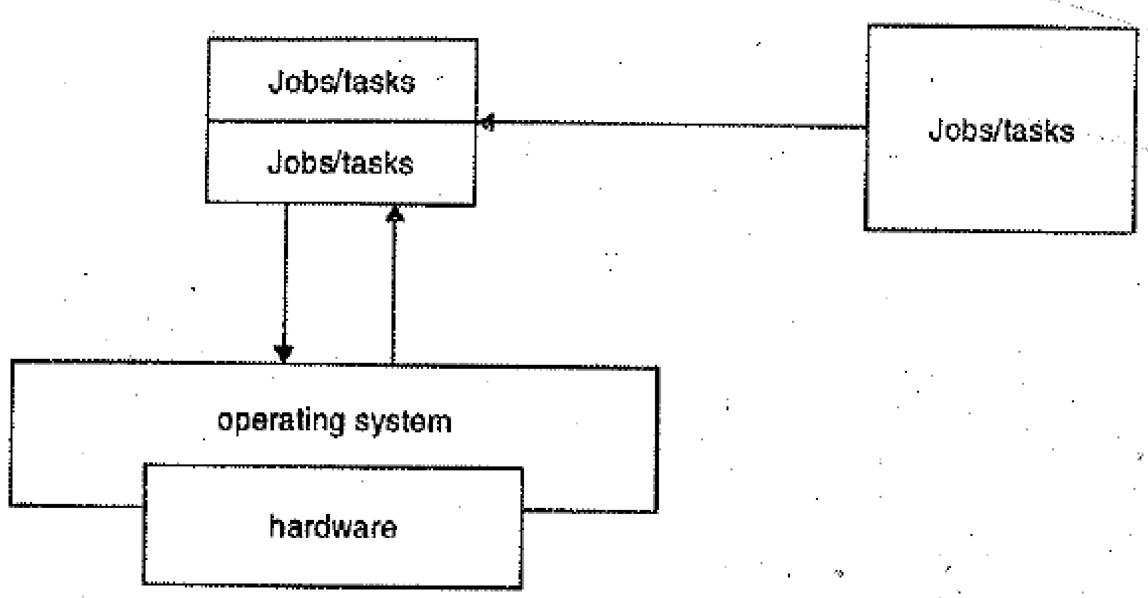


Figure 1.4. Batch processing system

. Another term that is commonly used in a batch processing system is Job Scheduling. Job scheduling is the process of sequencing jobs so that they can be executed on the processor. It recognizes different jobs on the basis of first-come-first-served (FCFS) basis. It is because of the sequential nature of the batch. The batch monitor always starts the next job in the batch. However, in exceptional cases, you could.. also arrange the different jobs in the batch depending upon the priority of each batch. Sequencing • of jobs according to some criteria require scheduling the jobs at the time of creating or executing a batch: On the basis of relative importance of jobs, certain 'priorities' could be set for each batch of jobs. Several batches could be formed on the same criteria of priorities. So, the batch having the highest priority could be made to run earlier than other batches. This would give a better turn around service to the selected jobs.

Now, we discuss the concept of storage management. At any Point of time, the main store of the computer is shared by the batch monitor program and the current user job of a batch, The big question that comes in our mind is how much storage has to be kept for the monitor program and how much has to be provided for the user jobs of a batch. However, if trio much main storage is provided to the monitor, then the user programs will not get enough storage. Therefore; an overlay structure has to be devised so that the unwanted sections of monitor code don't occupy storage simultaneously. Next we will discuss the concept', of sharing and protection. The efficiency of utilization of a computer system is recognized by its ability of 'sharing the system's hardware and software resources amongst its users. Whenever, the idea of sharing the system resources comes in your mind certain doubts also arise about the fairness and security of the system. Every user wants that all his reasonable requests should be taken care of and no intentional and unintentional acts of other users should fiddle with his data. A batch processing system guarantees the fulfillment of these user requirements. All the user jobs are performed one after the other. There is no simultaneous execution of more than one job at a time. So, all the system resources like storage JO devices, central processing unit, etc., are shared sequentially or serially. This is how sharing of resources is enforced on a batch processing system. Now, arises the question of protection. Though all the jobs are processed simultaneously, this too can lead to loss of security or protection. Let us suppose that there are two users A and B. User A creates a file of his own. User B deletes the file created by User A. There are so many other similar instances that can occur in our day to day life. So, the files and other data of all the user's should be protected against unauthorized usage. In order to avoid such loss of protection, each user is bound around certain rules and regulations. This takes the form of a set of control statements, which every user is required to follow.

Multiprogramming Operating System

The objective of a multiprogramming operating system is to increase the system utilization efficiency. The batch processing system tries to reduce the CPU idle time through operator interaction. However, it cannot reduce the idle time due to IO operations. So, when some 10 is being performed by the currently executing job of a batch, the CPU sits idle without any work to do. Thus, the multiprogramming operating system tries to eliminate such idle times by providing multiple computational tasks for the CPU -to perform. This is achieved by keeping multiple jobs in the main store. So, when the job that is being currently executed on the CPU needs some I/O, the CPU passes its requirement over to the I/ 0 processor. Till the time the 1/0 operation is being carried out, the CPU is free to carry out some other job. The presence of independent jobs guarantees that the CPU and I/O activities are totally independent of each other. However, if it was not so, then it could lead to some erroneous situations leading to some time-dependent errors. Some of the most popular multiprogramming operating systems are: UNIX, VMS, Windows NT etc.

A multiprogramming supervisor has a very difficult job of managing all the activities that take place simultaneously in the system. He has to monitor many different activities and react to a large number of different situations in the course of working.. The multiprogramming supervisor has to look through the following control functions :

Time Sharing or Multitasking System

. Time sharing or multitasking, is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them, but the switches occur so frequently that the users may interact with each program while it is running.

An interactive, or hands-on, computer system provides on-line communication between the user and the system. The user gives instructions to the operating system or to a program directly, and receives an immediate response.

Usually, a keyboard is used to provide input, and a display screen (such as a cathode-ray tube (CRT), or monitor) is used to. provide output. When the operating system finishes the execution of one command, it seeks the next "control statement" not from a card reader; but rather from the user's keyboard. The user gives a command, waits for the response, and decides on the next command,. based on the result of. the previous one. The user can easily experiment, and can see results immediately. Most systems have an interactive text editor for entering programs, and 'an interactive debugger for assisting in debugging programs.

If users are to be able to access, both data and code conveniently, an On-- line file system must be available. A file is a collection of related information defined by its creator. Commonly, files represent. programs (both source and object forms) and data: Data files may be numeric, . alphabetic, or alphanumeric. Files may be free-form, such as text files, or may be rigidly formatted. In general,..a file is a sequence of bits, bytes, lines, or records whose meaning is defined by its creator: and . user. The operating system implements the. abstract concept of a file by. managing mass-storage devices, such as tapes and disks. Files are normally Organized' into' logical. clusters, or directories, which make them easier to locate and access. Since multiple. users have access to. files, it is desirable to control by whom and in what ways files may be accessed. Batch systems are appropriate for executing large jobs that need little interaction. .The user can submit jobs and return later for the results; it is. not . necessary for the user to wait while the job is processed.

Interactive jobs tend to be Composed Of Many short' actions, where the results of the next command May be unpredictable. The user submits the command and then waits for the results. Accordingly, the response time should be .short, on the order of seconds at most.

An interactive system is used when a short-response lie is required. Early computers 'with a single user were interactive systems. That is, the entire system was. at the immediate disposal of the programmer/ operator. This situation allowed the programmer great flexibility and freedom in program testing and development. But, as we saw, this arrangement resulted in substantial idle time while the CPU waited for some action to be taken by the programmer/operator. Because of the high cost of these early computers, idle CPU time was undesirable. Batch operating systems were developed to avoid this problem. Batch systems improved system utilization for the owners of the computer systems.

Time-sharing systems were developed 'to provide interactive use of a computer system at a reasonable cost. A time-shared operating system uses CPU scheduling and multiprogramming to provide each .user with a small portion of a time-shared computer.

Each user has at least one separate program in memory. A program that is loaded into memory and is executing is commonly referred to as a process. When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O. I/O may be interactive; that is, output is to a display for the user and input is from a user keyboard.

Since interactive I/O typically runs at people speeds, it may take a long time to complete. Input, for example, may be bounded by the user's typing speed;' five characters per second is fairly fast for people, but is incredibly slow for computers. Rather than let. the. CPU sit idle when this interactive input takes place, the operating system will rapidly switch the CPU to the program of some other user.

A time-shared operating system allows the many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that she has her own computer, whereas actually one computer is being. shared among many users. The idea of. time-sharing was demonstrated as early as 1960, but since, time-shared. systems are difficult and expensive to build, they did not become common until the early 1970s. As the popularity' of time-sharing has grown, researchers have. attempted to merge batch and time-shared systems. Many Computer' systems that were designed as primarily batch systems have been modified to create a time-sharing subsystem. For example, IBM's 'OS/360; a batch system, was 'modified to support the time-sharing option (TS0): At the same tithe, time-sharing systems have often added a .batch• subsystem. Today, most systems provide both batch processing- and time sharing, although their basic design.and Use tends to. be one or the other type.. Time-sharing operating systems are even more complex than are multi-programmed operating Systems. As in multiprogramming, several jobs trust be kept. simultaneously' in memory, which requires some form of memory management and protection. So that a reasonable response time can be obtained, jobs may have to be swapped in and out of main memory. Many universities And businesses have large numbers of workstations tied together with local-area networks. As PCs gain more sophisticated hardware and software, the line dividing the two categories is blurring.

Parallel Systems

Most systems to date are single-processor systems; that is, they have only one main CPU. However, there is a trend toward multiprocessor systems. Such systems have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices. These systems are referred to as tightly coupled systems.

There are several reasons for building such systems. One advantage is increased throughput. By increasing the number of processors, we hope to get more work done in a shorter period of time. The speed-up ratio with n processors is not n, however, but rather is less than n. 'When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources lowers .the expected gain from additional processors. Similarly, a group of n programmers working closely together does not result in n times the amount of work being accomplished.

Multiprocessors can also save money compared to multiple single systems because the processors can share peripherals, cabinets, and power supplies. If several programs are to operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them, rather than to have many computers with local disks and many copies of the data.

Another reason for multiprocessor systems is that they increase reliability. If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, but rather will only slow it down. If 'we have 10 processors and one fails, then each of the remaining nine processors must pick up a share of the work of the failed processor. Thus, the entire system runs only 10 per cent slower, rather than failing altogether.

This ability to continue providing service proportional to the level of surviving hardware is called graceful degradation. Systems that are designed for graceful degradation are also called fault-tolerant.

Continued. operation in the presence of failures requires, a mechanism to allow the failure to be detected, diagnosed, and corrected (if possible). The Tandem system uses both hardware and software duplication to ensure continued operation despite faults. The system consists of two identical processors, each with its own local memory. The processors are connected by a bus. One processor is the primary, and the other is the backup. Two copies are kept of each process; one on the primary machine and the other on the backup. At fixed checkpoints in the execution of the system, the state information of each job (including a copy of the memory image) is copied from the primary machine to the backup. If a failure is detected, the backup copy is activated, and is restarted from the most recent checkpoint.

This solution is obviously an expensive one, since there is considerable hardware duplication. The most common multiple-processor systems now use the symmetric multiprocessing model, in which each processor runs an identical copy of the operating system, and the copies communicate with one another as needed. Some systems use 'symmetric multiprocessing, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors.

An example of the symmetric multiprocessing system is Encore's version of UNIX for the Multimax computer. This computer can be configured to employ dozens of processors, all running a copy of UNIX. The benefit of this model is that many processes can run at once (N processes if there are N CPUs) without causing a deterioration of performance. However, we must carefully control I/O to ensure that data reach the appropriate processor. Also, since the CPUs are separate, one may be sitting idler while another- is overloaded, resulting in inefficiencies. To avoid these inefficiencies, the processors can share' certain data structures. A multiprocessor system of this form will allow jobs and resources to be shared dynamically among the various processors, and can lower the variance among the systems. However, such a system must be written carefully.

Asymmetric multiprocessing is more common in extremely large systems, where one of the most time-consuming activities is simply processing I/ O. In older batch systems, small processors, located at some distance from the main CPU, were used to run card readers and line printers and to transfer these jobs to and from the main computer. These locations are called remote-job-entry (ERIE) sites. In a time-sharing system, a main I/O activity is proCessing the 1/0 of characters between the. terminals and the computer. If the main CPU must be interrupted for every character for every terminal, it may spend all its time simply processing characters. So that this situation is avoided, most systems have a separate front-end processor that handles all the terminal I/O. For 'example, a large IBM system\ might use an IBM Series/I minicomputer as a front-end.

The front-end acts as a buffer between the terminals and the main CPU, allowing the main CPU to handle lines and blocks of characters, instead of individual characters. Such systems suffer from decreased reliability through increased specialization. It is important to recognize that the difference between symmetric and asymmetric multiprocessing may be the result of either hardware or software.

Special hardware may exist to differentiate the multiple processors, or the software may be written to allow only one master and multiple slaves. For instance, Sun's operating system SunOS Version 4 provides asymmetric multiprocessing, whereas Version 5 (Solaris 2) is symmetric.

As microprocessors become less expensive and more powerful, additional operating system functions are off-loaded to slave-processors, or back-ends. For example, it is fairly easy to add a microprocessor with' its own memory to manage a disk system. The microprocessor could receive a sequence of requests from the main CPU and implement its own disk queue and scheduling algorithm. This arrangement relieves the main CPU of the overhead of disk scheduling. PCs contain a microprocessor in the keyboard to convert the key strokes into codes to be sent to the CPU. In fact, this use of microprocessors has become so common that it is no longer considered multiprocessing.

Distributed Systems

.A recent trend in computer systems is to distribute computation among several processors. In contrast to the tightly coupled syst6ms, the processors do not share memory or a clock. instead, each 'processor has its own memory and clock. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines.

These systems are usually referred to as loosely coupled systems, or distributed systems. The processors in a distributed system may vary in size and function. They may include small microprocessors, workstations, minicomputers, and large general-purpose computer systems. These processors are referred to by a number Of different names, such as sites, nodes, computers, and so on, depending on the context in which they. are mentioned.

There are a variety of reasons for building distributed systems, the major ones being :

Resource .sharing. If a number of different sites (with different capabilities) are .connected to one another, then' a user at one site may be able to use the resources available at another. For example, a 'user at site A may be using a laser printer available only at site B. Meanwhile, a user at B may access a file that resides at A. In general, resource sharing in a distributed system provides mechanisms for sharing files at remote sites, processing information in a distributed database, printing files at remote sites, using remote specialized hardware devices (such as a high-speed array processor), and performing other operations.

Computation speedup. If a particular computation can be partitioned into a number of sub computations that can run concurrently, then a distributed system may allow us to distribute the computation among the various sites—to run that computation concurrently. In addition, if a particular site is currently overloaded with jobs, some of them may be moved to other, lightly loaded, sites. This movement- of jobs is called load sharing.

Reliability. If one site fails in a distributed system, the remaining sites can potentially continue operating. If the system is composed of a number of large autonomous installations (that is, general-purpose computers), the failure of one of them should not affect the rest. If, on the other hand, the system is composed of a number of small machines, each of which is responsible for some crucial system function (such as terminal character I/O or the file system), then a single failure may effectively halt the operation of the whole system. In general, if sufficient redundancy exists in the system (in both hardware and data), the system can continue with its operation,. even if some of its sites have failed.

Communication. "There are many instances in which programs need to exchange data with one another on one system. Window systems are one example, since they frequently share data or transfer data between displays. When many sites are connected to one another by a communication network, the processes at different sites have the opportunity to exchange information. Users may initiate file transfers or communicate with one another via electronic mail. A user can send mail to another user at the same site or at a different site.

Real-Time Systems

Another form- of a special-purpose operating system is the real-time system. A real-time system is used when there are rigid time requirements on the operation of a processor or the flow of data, and thus is often used as a control device in a dedicated application. Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs. Systems that control scientific experiments, medical imaging systems, industrial control systems, and some display systems are real-time systems. Also included are some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems:

A real-time operating system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system. will fail. For instance, it would not do for a robot arm to be instructed to halt after it had smashed into the car it was building. A real-time system is considered to function correctly • only if it returns the correct result within any time Constraints. Contrast this requirement.: o a 'tithe-sharing system, where it is desirable (but not mandatory) to respond quickly, or to a batch system, where there may be no time constraints at all.

There are two flavors of real-time systems. A hard real-time Systerri guarantees that critical tasks complete on time. This goal requires that all delays in the system be bounded, from the retrieval of stored data to the time that it takes the operating system to finish any request made of it. Such time constraints dictate the facilities that are available in hard real-time systems. Secondary storage of any sort is :usually limited or missing, with data instead being stored in short-term memory, or in Read Only Memory (ROM). ROM is located on nonvolatile storage devices that retain their contents even in the case of electric outage; most other types of memory are volatile.

Most advanced operating system features are absent too, since they tend to separate the user further from the hardware, and that' separation results in uncertainty about the amount of time an operation • will take. For instance, virtual memory is almost never found on real-time systems. Therefore, hard real-time systems conflict with the operation of time-sharing systems, and the two cannot be .mixed. Since none of the existing general-purpose operating systems support hard real-time functionality, we do not .concern ourselves 'with this type of system in this text.

A less restrictive type of real-time system is a soft real-time system, where a critical real-time task gets priority over other tasks, and retains that priority until it completes.

1.3 OPERATING SYSTEMS ARCHITECTURE

A system as large and complex as a modern operating system must be engineered carefully if it is to • function properly and to be modified easily. .A common approach is to partition the task into small components, rather than have one monolithic system. Each of. these . modules should be a well-defined portion of the system, with; carefully defined inputs, outputs, and function. We have already dismissed briefly the common components of operating systems. In this section, we discuss the way that these components are interconnected and melded into a kernel.

Simple Structure

There are numerous commercial systems that do not have a well-defined structure. Frequently, :such operating systems started' as small; simple and limited systems, and then grew beyond. their original scope. &Dos is an example of such a system. It was originally designed and plernented by a few people who had no idea that it would become so pular. It was written to provide the most functionality in the least • ace, because of the limited hardware on which it ran, so it was not 'vided into modules carefully.. **Figure 1.5 shows its structure.**

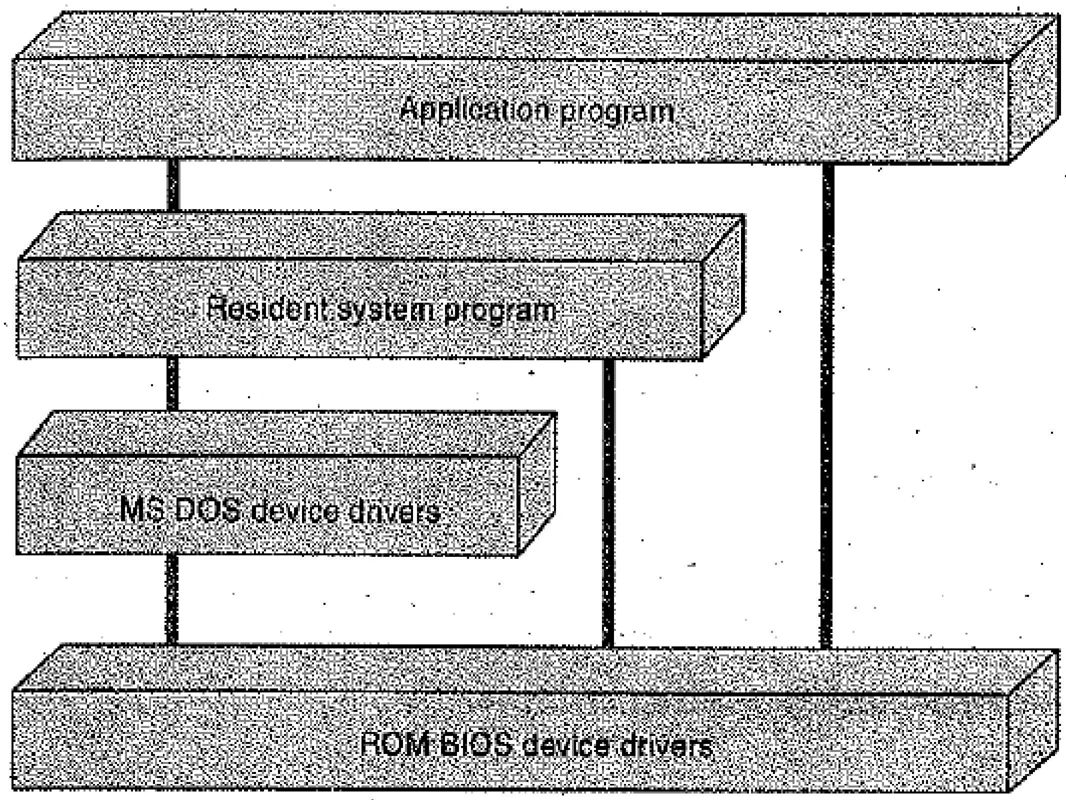


Figure 1.5. MS-DOS layer structure.

IN MS-DOS, the interfaces and levels of functionality are not well separated. For instance, application programs are able to access the basic I/0 routines to write directly to the display and disk drives. Such freedom leaves ,MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail. Of course, MS-)DOS was also limited by the hardware of its era. Because the Intel 8088 or which it was written provides no dual mode and no hardware protection, the designers of MS-DOS had no choice but to leave the rase hardware accessible.

Another example of limited structuring is the original UNIX operating system. UNIX is another system that initially was limited by hardware. functionality. It consists of two separable parts : the kernel and these systems programs.

The kernel is further separated into a series of interfaces and device ivers, which have been added and expanded over the years as UNIX as evolved. We can view the UNIX operating system as being layered s shown in Figure 1.6.

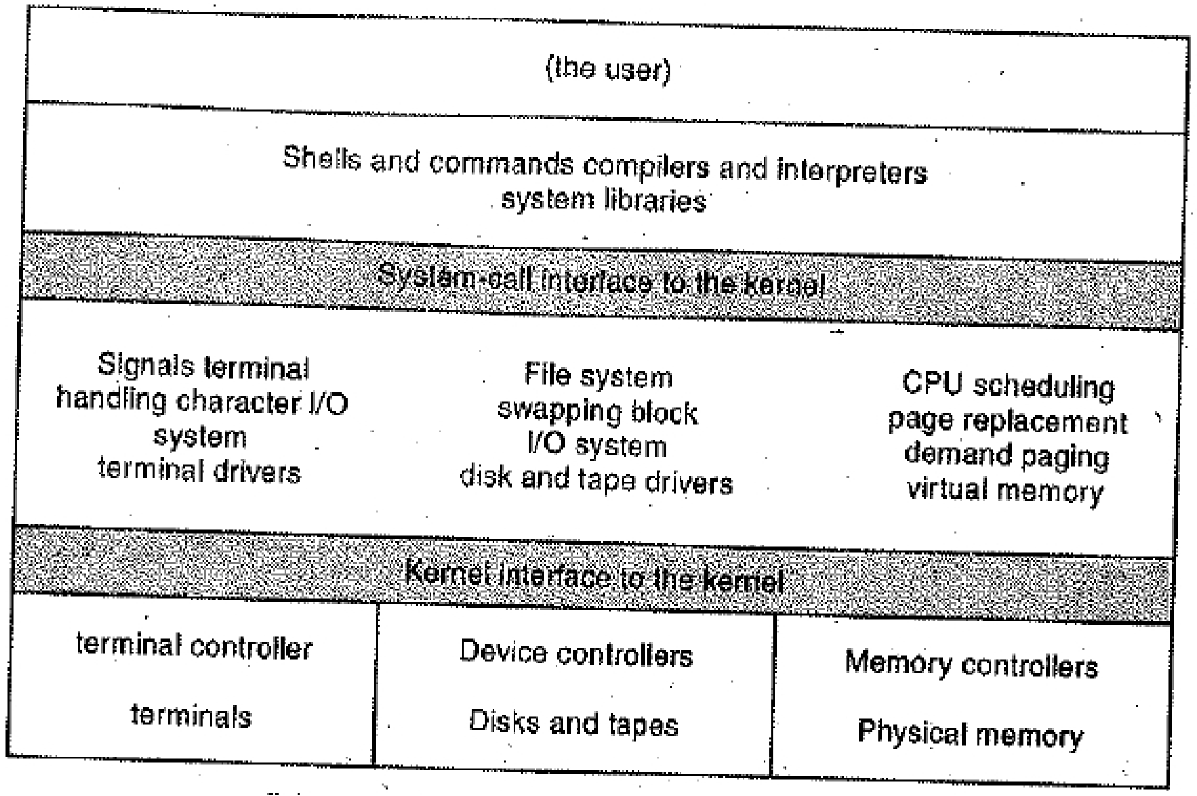

(the user)
Shells and commands compilers and interpreters system libraries
System-call interface to the kernel
Signals terminal handling character I/O system terminal drivers
File system swapping block I/O system disk and tape drivers
CPU scheduling page replacement demand paging virtual memory
Kernel interface to the kernel
terminal controller
terminals
Device controllers
Disks and tapes
Memory controllers
Physical memory

Figure 1.6. UNIX system structure.

Everything below the system-call interface and above the physici hardware is the kernel. The kernel provides the file system, CI scheduling, memory management, and other operating system functio4 through system calls.

Taken in sum, that is an enormous amount of functionality to be combined into one level. Systems programs use the kernel-supported system c to provide useful functions, such as compilation and file manipulation

System calls define the programmer interface to UNIX; the set systems programs commonly available defines the user interface. T programmer and user interfaces define the context that the kern must support. -Several versions of UNIX have been developed in whit the kernel is partitioned further- along functional boundaries. The operating system, IBM's version of UN separates the• kernel into t parts. Mach, from Carnegie

Mellon University, reduces the kernel to small set of core functions by moving all nonessentials into and even into user-level programs. What remains is a. microker operating system implementing only a small set of necessary primitiv

Layered Approach

These new UNIX versions are designed to use more advanced hardware Given proper hardware support, operating systems may be broken in smaller, more' appropriate pieces than those allowed by the origin MS-DOS or UNIX. The operating system can then retain much great control over the computer and • the applications that make use of the computer. Implementors have more freedom to make changes to tl inner workings of the system. Familiar techniques are used to aid: the creation of modular operating systems. Under the top-down approach, the overall functionality and features can be determined an separated into components. Information hiding is also important, leave programmers free to implement the low-level routines as they see fit, Med that the external interface of the routine stays unchanged and .::routine itself performs the advertised task.

The modularization of a system can be' done in many ways; the most pealing is the layered approach, which consists of breaking the grating system into a number Of layers (levels), each built on top of war layers. The bottom. layer (layer 0) is the hardware; the highest gayer N) is the user interface.

An operating system layer is an implementation of an abstract object at is the encapsulation of data, and operations that can manipulate ose data. A typical operating system layer—say layer M—is depicted Figure 1.7.

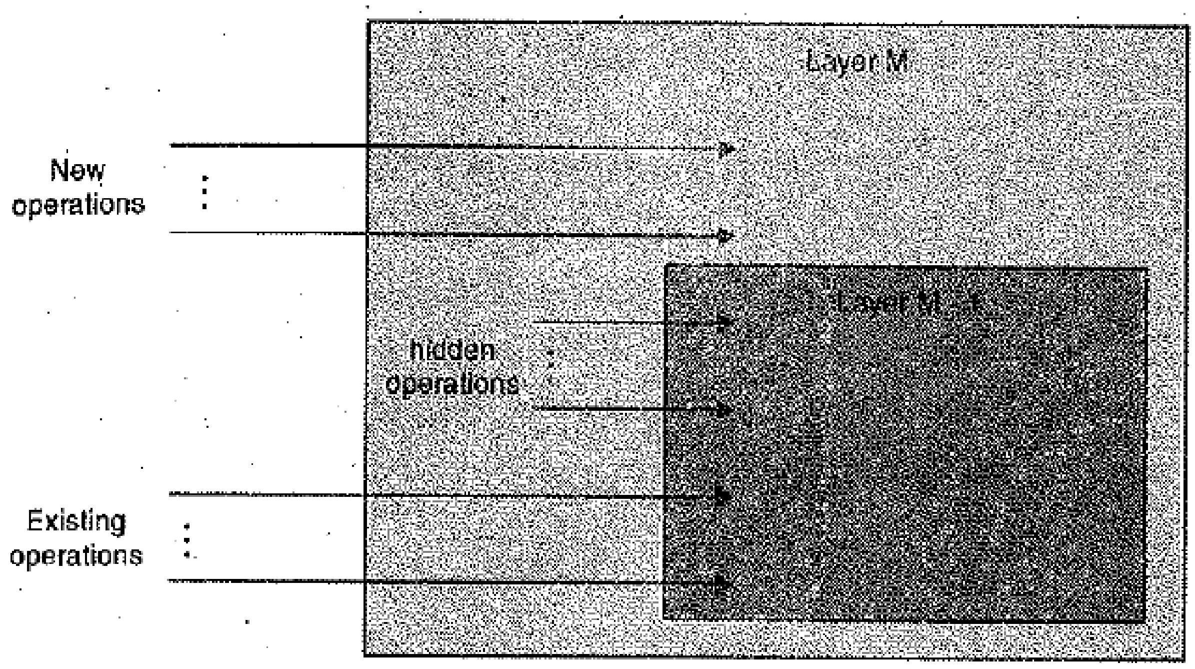


Figure 1.7. *An operating system layer.*

It consists of some data structures and a set of routines that can be invoked by. higher-level layers. Layer M, in return, can invoke operations lower-level layers.

The main advantage of the layered approach is modularity. The layers are selected such that each uses functions (operations) and services of only lower-level layers.This approach simplifies debugging and system verification. The first layer can be debugged without any concern for the rest of the system, because, by definition, it uses, only the basic hardware (which is assumed correct) to implement its functions. Once the first layer is debugged, its correct functioning can be assumed while the second layer is worked on, and so on. Ie an error is found during the debugging of a particular layer, we know that the error must be on that layer, because the layers below it are already debugged. Thus, the design and implementation of the system is simplified when the system is broken down into layers.

Each layer is implemented using only those operations provided by lower-level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations Hence, each layer hides the existence of certain data structur operations, and hardware from higher-level layers.

The layer approach to design was first used in the THE operation' system at the Technische Hogeschool Eindhoven., The THE system w defined in six layers, as shown in Figure 1.8. The bottom layer was t hardware. The next layer implemented CPU scheduling. The next la implemented memory management; the memory management sche was virtual :memory. Layer 3 contained the device driver for t operator's console. Because it and 1/0 buffering (level 4) were plac above memory management, the device buffers could be placed in virtu memory. The I/O buffering was also above the operator's console, that I/O error conditions could be output to the operator's console.

layer 5 : user programs

layer 4 : buffering for input and output devices

layer 3 : operator-console . device driver

layer 2 : memory management

layer 1 : CPU scheduling

layer 0 : hardware

Figure 1.8. THE layer structure.

This approach can be used in many ways. For example, the Ven.tki system was also designed using a layered approach. The lower layer .(0 to 4), dealing with CPU scheduling and memory management, were then put into microcode. This decision provided the advantages O additional speed of execution and a clearly defined interface between the microcoded layers and the higher layers.

The major difficulty with the layered approach involves the appropria4- definition of the various layers. Because a layer can use only those. layers that are at a lower level, careful planning is necessary.. FO' example, the device driver for the backing store (disk space used . 1; virtual-memory algorithms) must be at a level lower than that of tli‘4 memory-management routines, because memory management require4 the ability to use the backing store.

Other requirements may not be so obvious. The backing-store driver4 would normally be above the CPU scheduler, because the driver ma, need to wait for I/O. and the CPU can be rescheduled during this time However, on a large. system, the CPU scheduler may have more information about all the active processes than can fit in memory.. Therefore, this information may need to be swapped in and out of. memory, requiring the backing-store driver routine to be below the .CPU scheduler.

A 'final problem with layered implementations is that they tend to be less. efficient than other types. For instance,. for a user program to execute an I/O operation, it. executes a, system call which is trapped to the .110 layer, which calls the memory-management layer, through to !'.- the CPU scheduling. layer, and finally to the 'hardware. At each layer, the parameters may be modified, data may need to be passed, and so on. Each layer adds overhead to the system call. and the net result' is a system call that takes longer than one does on a nonlayered system.

These limitations have :caused. a small backlash against layering in recent years. Fewer 'layers with more 'functionality are being designed, providing Most of the advantages of modularized code while avoiding the 'difficult problems of layer definition and interaction. For instance, ()S./2 is a descendant of MS-DOS that adds multitasking and dual-mode operation, as well as -other new features. • . Because of this added complexity and the more powerful hardware for which OS/2 was designed the system was implemented in a more layered fashion. Contrast the MS-DOS structure to that of the OS/2. It should be clear that, from both the system-design and implementation standpoints, OS/2 'has the advantage. For instance, direct 'user access to low-level facilities is not allowed, providing the operating system with more control over the hardware and more knowledge of which resources each user 'program is using.

As a further example, consider the history of Windows NT. The first release -had a very layer-oriented organization. However, this version suffered low performance compared to that of Windows .95. Windows. NT 4.0 redressed some of these performance issues by moving layers from user space to -kernel space and More closely integrating them.

1.4 VARIOUS FUNCTIONS OF OPERATING SYSTEM

: An operating :system is a program that acts as an intermediary between a user and the computer hardware. 'The purpose of an. operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner. The operating system must ensure the correct operation of the computer system. To prevent user programs from interfering with the proper operation of the System the hardware must provide. appropriate mechanisms to ensure such. proper-behavior .

The operating system provides certain services to programs and to the users of those programs in order to make the programming task easier. Basically the functions of an operating system are:

1. Program execution

2.. I/O operations

3. File system

4. Error detection

5. Resource allocation

6. Protection

The first and, basic need of an OS is to overcome the idle time of the CPU. in the beginning, jobs were run one at a time with the computer operator literally operating the computer by loading time cards and pushing buttons on the computer console. As time progressed, resident monitors, the first OS's, sat in main memory and automatically transferred control from one job to the next. This reduced the idle time of the CPU between jobs but not during the execution of jobs.

With this automatic sequencing, however, the CPU is often idle. The problem is the speed of the mechanical I/O devices, which are intrinsically slower than electronic devices. Even a slow CPU works in the microsecond range, with millions of instructions executed per second. A fast disk, on the other hand, might have a response time of 15 milliseconds and 'a transfer rate of 1 megabits/second, on an order of a magnitude slower than a typical CPU.

The slowness of the I/O devices can mean that the CPU is often waiting for 110. As an example, as an example m an assembler or compiler may be able to process 300 or more cards per second. A fast card reader, on the other hand, might read only 17 cards per t second. This means that assembling a 1200 card program would require only 4 seconds of CPU time, but 60 seconds to read. Thus, the CPU is idle for 56 out of 60 seconds, or 93.3 per cent of the time. The resulting CPU utilization is only 6.7 per cent. The process is similar for output operations. The problem is that, while an I/O operation is occurring, the CPU is idle, waiting for the I/O to complete; while the CPU is executing, the 1/0 devices are idle. Overtime, of course, improvements in technology resulted in faster I/O devices. But CPU speeds increased even faster, so that the problem was not only unresolved, but also worsened.

Off-line Operations

One common solution was to replace the very slow card readers (input devices) and line printers(output devices) with magnetic-tape units. The Majority of computer systems in the late 1950's and early 1960's were batch systems ;reading from card readers and writing to line . printers or card punches. Rather than have the CPU read directly from cards, however, the cards were first copied onto a magnetic tape. When the tape was sufficiently full, it was taken down and carried over to the computer. When a card was needed for input to a program, the equivalent record was read from tape. Similarly, output was written • to the tape and the contents of the tape would be printed later. The card readers and line printers were operated off-line, not by the main computer.

card reader > CPU line printer

(a) on-line card reader —> tape drives CPU tape drives —> line printer

(b) off-line

The main advantage of off-line operation was that the main computer was no longer constrained by the speed of the card, readers and line printers, but was limited by the speed of the much faster magnetic tape units.

Buffering and Spooling

Buffering is the method of overlapping the I/O of a job with its own computation. The idea is quite simple. After data have been read and the CPU is about to start operating on them,- the input device is instructed to begin the next input immediately. The CPU and the input device are then both busy. With luck, by the time that the CPU is ready for the next data item(record), the input device will have finished reading it. The CPU can then begin processing the newly read data, while the input device starts to read the following data. Similarly, this can be done for output. In this case, the CPU creates data that are put into a buffer until an output device can accept them.

Although off-line preparation of jobs continued for some time, it wa's quickly replaced in most systems. Disk systems became widely available and greatly improved on off-line operation. The problem with tape systems was that the card reader could not write onto one end Of the tape while the CPU read from another. The 'entire tape had to be written before it was rewound and read. Disk systems eliminated this problem. Because the head is moved from one area of the disk to another, a disk can rapidly switch from the area of the disk being used by the card reader to store new „cards to the position needed by the CPU to read the 'next' card.

In a disk system, cards are read directly from the card reader onto the disk. The location of card images is recorded in a table kept by the operating system. When a job is executed, the operating system satisfies its requests for card reader input by reading from the disk. Similarly; when the job requests the printer to ouput a line, that line is copied;; into a system buffer and is written to the disk. When the job is completed, the output is actually printed.

This form of processing is called spooling. The name is an acronym simultaneous peripheal operation on-line. Spooling essentially uses • the disk as a very large buffer, for reading as far ahead as possible on input devices and for storing output .files until. the output devices are--: able to accept them.

disk

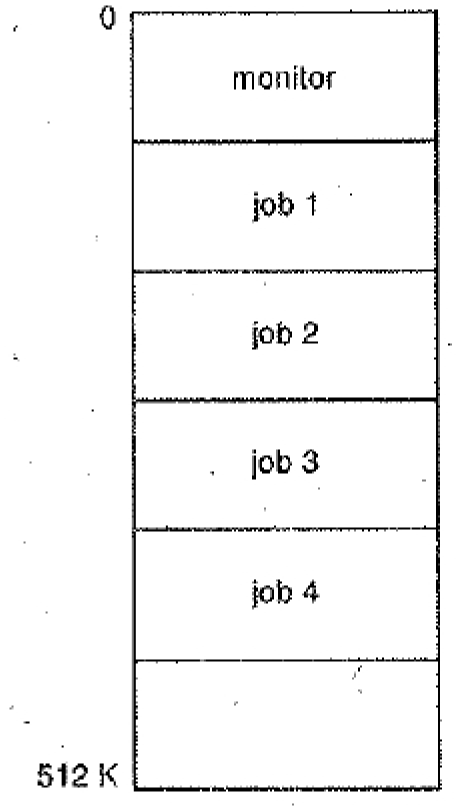
card reader CPU line printer

In addition, spooling provides a very important data structure : a job pool. Spooling will generally result in several jobs that have all ready been read waiting on disk, ready to run. A pool of these jobs on disk. allows the operating system to select which job to run next, in order to increase CPU utilization. When jobs come in directly from tape, it is not possible to skip around and to run jobs in a different order. Jobs must be run sequentially, on a first-come, first-served basis. However, when several jobs are on a direct access device, such as a disk, job scheduling becomes possible.

The most important aspect of job. scheduling is the ability to multi programs. Off-line operation, buffering, and spooling for overlapped I/O have their limitations. A Single user cannot, in general, keep either the' CPU or the I/O devices • busy at all times. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has something to execute.

The operating system picks and begins to execute one of the jobs in the job pool. Eventually, the job may have to wait for some task, such as .a tape to be mounted, a command to be typed on a keyboard, or an I/O Operation to complete. In a non-multi programmed system, the CPU would sit idle. In a multiprogramming system, the operating system simply switches to and executes .another job. When that job needs to wait, the CPU is 'switched to another job, and so on. Eventually, the first job finishes Waiting and gets the CPU back. As long as there is always some job to execute, the CPU will never be idle.

Multi programmed operating 'systems are fairly. sophisticated. To have . several jobs ready to run, the system must keep all of them. in memory simultaneously. Having several programs in memory at the same time requires some form of memory management. In additions , if several jobs. are ready to run at the same time, the system must choose among them. This decision is CPU Scheduling. Finally, multiple jobs running concurrently require that their ability to affect one another be limited in all phases of the operating system, including process scheduling, disk storage, and memory management.



Multitasking

Multi tasking (or time sharing) is a logical extension of multi-programming. Multiple jobs are executed by the CPU switching between them, but • the switches occur so frequently that the users may interact with each program while it is running. To understand the difference, we will first review the batch method.

When batch systems were first developed, they were defined by the `hatching' together of similar jobs. Card- and tape-based systems allowed only sequential access to programs and data, so only one application package(the FORTRAN compiler, linker, and loader, or the COBOL equivalents, for instance) could be used at a time. As on-line disk storage became feasible, it was possible to provide immediate access to' all of the application packages. Modern batch systems are no longer defined by the batching together of similar jobs; other characteristics are used instead.

A batch operating system normally reads a stream `of separate jobs(from a card reader, for example), each with its own control cards that predefine what the job does. When the job is complete, its output is usually printed (on a line printer, for example). The definitive feature of a batch system is the lack of user interaction between the user. and the job while that job is executing. The job is prepared and submitted. At some later time(perhaps minutes, hours, or days), the output appears. The delay between job submission and job completion(called turn around time) inayresult from the amount of computing needed, or from delays before the operating system starts processing the job.

An interactive, or ,hands-on, computer system provides on-line communication between the user and the system. The user gives instructions to the operating system or to a program directly,, and receives an immediate response. When the operating system finishes the execution of one command, it seeks the next 'control statement'. The user can easily experiment and can see results immediately.

Time-sharing systems were developed to provide interactive use of a computer system at a reasonable cost. A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has a separate program in memory. When a program executes, it typically executes for only a short time before it either finishes or needs to perform I/O. I/O may be interactive; that is, output to a display, or input from a keyboard. Five characters per second may be fairly fast for people, but it is very slow for computers. Rather than let the CPU sit idle when this happens, the operating system will rapidly switch the CPU to the program of some other user. Time-sharing operating systems are sophisticated. They provide a mechanism for concurrent execution. Also, as in multiprogramming, several jobs must be kept simultaneously in memory, which requires some form of memory management, protection, and CPU scheduling. So that a reasonable response time can be obtained, jobs may have to be swapped in and out of main memory. Hence, disk management must also be provided. Time-sharing systems must also provide an on-line file system and protection.

System Calls

System calls provide the interface between a .running program and the operating system. These calls are generally available as assembly-language instructions, and are usually listed in the manuals used by assembly-language programmers.

Some systems may allow system calls to be made directly from a higher-level language program in which case the calls normally resemble predefined function or subroutine calls. They may generate a call to a special run-time routine that mak& the system call, or the system call may be generated directly in-line.

Several languages, such as C, Bliss, and PL/360, have been defined to replace assembly language for systems programming. These languages allow system calls' to be made directly.

Three general methods are used to pass parameters to the • operating system. The simplest approach is to pass the parameters in registers. In some cases, however, there may be more parameters than registers. In these cases, the parameters are generally stored in a block or table in memory m and the address of the block is passed as a parameter in a register. Parameters may also be placed, or pushed, onto the stack by the program, and popped off the stack by the operating system.

System calls can be roughly grouped into five major categories : process control, file manipulation, device manipulation, information maintenance, and communications.

(1) Process Control

— End, Abort

— Load, Execute

— Create Process, Terminate Process

— Get Process Attributes, Set Process Attributes

— Wait for Time

— Wait Event, Signal Event

— Allocate Memory, Free Memory

(2) File Manipulation

— Create File, Delete File

— Open, Close

— Read, Write, Reposition

— Get File Attributes, Set File Attributes

(3) Device Manipulation

— Request Device, Release Device

— Read, Write, Reposition

— Get Device Attributes, Set Device Attributes

— Logically Attach or Detach Devices

(4) Information Maintenance

— Get Time or Date, Set Time or Date

— Get System Data, Set System Data

— Get Process, File, or Device Attributes

— Set Process, File, or Device Attributes

(5) Communications

— Create, Delete Communication Connection

— Send, Receive Messages

— Transfer .Status Information

Process Management

A process is a program in execution. In general, a process will need certain, resources-such as CPU time, memory, files, and I/O devices-to accomplish its task. These resources are allocated to the process either when it is created, or while it is executing.

A process is the unit of work in most systems. Such a system consists of a collection of processes operating system processes execute system code, and user processes execute user code. All of these processes can potentially execute concurrently.

The operating system is responsible for the following activities in connection with process management : the creation and deletion of both user and system processes; the scheduling of processes, and the provision of mechanisms for synchronization, communication,. and deadlock handling for processes. A process is more than the program code plus the current activity. A process generally also includes the process stack containing temporary data (such as subroutine parameters, return addresses, and temporary variables), and a data section containing global variables.

A program by itself is not a process; a program is a passive entity, such as the contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute.

Process State

. As a process executes, it changes state. The state of a process is defined in part by that process's current activity. Each process may be in one of three states : new process

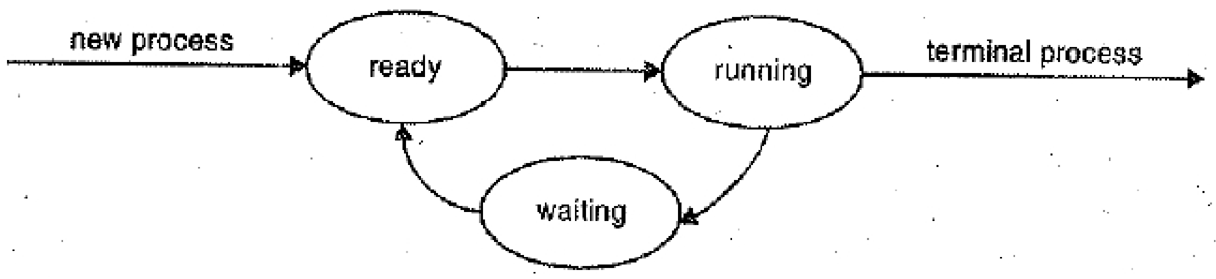


Figure 1.9. Process state diagram.

• **Running**. Instructions are being executed.

• **Waiting**. The process is waiting for some event to occur (such as an I/O completion)

• **Ready.** The process is waiting to be assigned to a processor. These names are rather arbitrary, and vary between operating systems. The states they represent are found on all systems, however. It is important to realize that in a single-processor system, only one process can be running at any instant. Many processes may be ready and waiting, however.

Process Control Block

Each process is represented in the operating system by its own process control block(PCB). A PCB is a data block or record containing many pieces of the information associated with a specific process, including

• **Process State**. The state may be new, ready, running, waiting, or halted.

• **Program Counter**. The counter indicates the address of the next instruction to be executed for this process.

•**CPU State**. This includes the contents of general purpose registers, index registers, stack pointers, and any condition-code information. Along with the program counter, this state information must be saved en an interrupt occurs, to allow the process to be continued correctly afterward.

- **CPU Scheduling Information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling atarneters.
- **Memory-management information**. This information includes limit egisters or page. tables.
- **I/o Status Information**. The information includes outstanding 110 requests and a list of open files.

The PCB serves as the repository for any information that may vary irijin process to process.

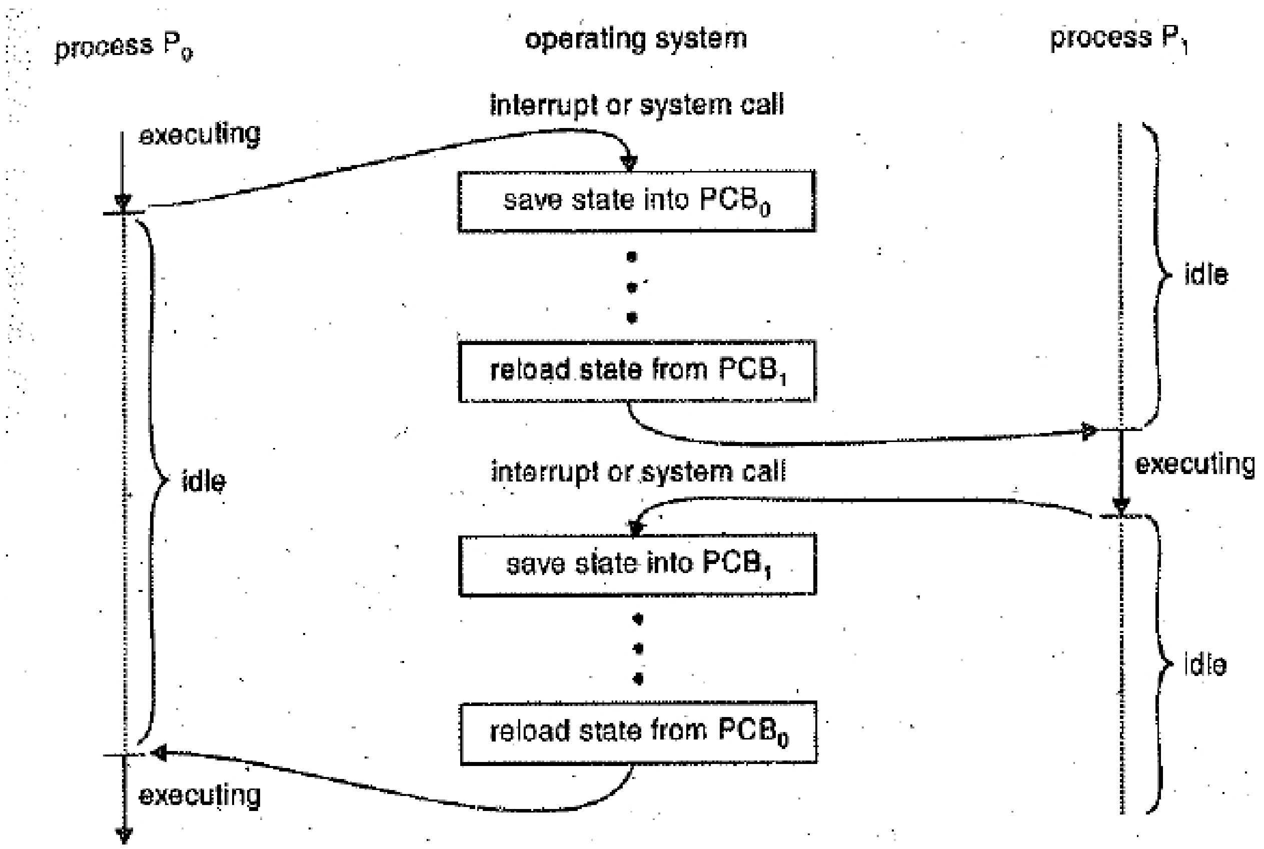


Figure 1.10.' The CPU can be switched from process to process.

Concurrent Processes

The processes in the system 'can execute "concurrently; that is, many processes may be multitasked on a CPU. There are 'several reasons for allowing concurrent execution :
• **Physical Resource Sharing**. Since the computer hardware resources are limited, we may be forced to share them in a multiuser environment.

• **Logical Resource Sharing**. Since several processes may be interested in the same piece of information (for instance, a shared file), we must provide an environment to allow concurrent processes access to these types of resources.

• **Computation speedup**. If we want a particular task to run faster, we may break it into sub-tasks, each of which will be executing in parallel with the others. Notice that such a speed up can be achieved only if the computer has multiple processing elements (such as CPUs or channels)

- **Modularity.** We may want to construct the system in a modulat. fashion; dividing the system functions into separate processes. * Convenience. Even an individual user may have many tasks tO work on at one time. For instance, a user may be editing, printind and compiling in parallel. Concurrent execution that requires cooperation among the processe0 requires a mechanism for process synchronization and communication

Threads

Recall that a process is defined by the resources it uses and by they location at which it is executing. There are many instances, however,1 in which it would be useful for resources to be shared and accessedl concurrently. This situation is similar to the case where a fork systenil call is invoked with a new thread of control executing within the samel virtual address space. This concept is so useful that several newt operating systems are providing a mechanism to support it through' the thread facility. A thread is a basic unit of CPU utilization. It has little nonshared state. A group of peer threads share code, addresd space, and operating system resources. The environment in which al thread executes is called a task. A traditional(heavyweight) process is equal to a task with one thread. A task does nothing if no threads are in it, and a thread must be in exactly one task. An individual-:thread has at least its own register state, and usually its own stack, Thel extensive sharing makes CPU switching among peer threads ancli threads' creation inexpensive, compared with context switches among?: heavyweight processes, Thus, blocking a thread and switching to another:; thread is a reasonable solution to the problem of how a server cad efficiently handle many requests, Threads are gaining in popularity because they have some of the characteristics of heavyweight processes but can execute morel efficiently. There are many applications where this combination is -^ useful. For instance, the UNIX kernel is single-tasking; Only one taskl can be executing in the kernel at a time Many problems, such as synchronization of data access(locking of data structures wh4e are being modified) are avoided because only one process is allowed to be doing the modification. Mach, on the other hand, is multithreaded,d allowing the kernel to service many requests simultaneously. In this case, the threads themselves are synchronous : another thread in the same group may run only if the currently executing thread relinquishes" control. Of course, the current thread would only relinquish control: only when it was not modifying shared data On systems on which.'" threads are asynchronous, some explicit locking mechanism .must be used, just as in systems where multiple processes share data

SUMMARY

An operating system is the most important program in a computer system. is is one program that runs all the time, as long as the computer is sperational and exits only when the computer is shut down. .

- Operating. systems are the programs that make computers operational, hence the name.
- Operating systems are computers' resource manager.
- The main advantage of the layered approach is modularity. The layers are selected such that each uses functions (operations) and services of only lower level layers.
- System calls provide the interface between a process and the operating system. These calls are generally available as assembly-language instructions, and are usually listed in the manuals used by assembly-language programmers.

REVIEW QUESTIONS

1. What is the main advantage of layered approach to operating system design? Explain.

2. How does a distributed system enhance resource sharing?

3. Multiprogramming is essentially a sequential execution of programs. Comment.

4. What are the constraints of a real time system?

5. What are the benefits of multiprogramming?

6. What are the characteristics of real time operating systems?

7. Classify operating system, and explain the 'various functions of operating system.

FURTHER READING

1. Operating Systems: A practical Approach: Rajiv Chopra, S. Chand Publisher, 2010

2. Operating Systems: Principles and Design: Pabitra Pal Ch.oudhury, PHI Learning
3. Operating Systemes and Software Diagnostics: Ramesh Bangia and Balvir Singh, Laxini Pub., 2007

4. Principles of Operating Systmes: Sri V. Ramesh, Laxmi Pub., 2010

UNIT II

FILE SYSTEM AND MANAGEMENT

STRUCTURE

2.0 LEARNING OBJECTIVES

After reading this unit, you will be able to:

- define file management and systems
- discuss the idea about file operations
- explain different levels of directory
- discuss the .disk space allocation methods
- know about file access methods
- explain the sharing of files
- discuss the 'examples idea about file protection mechanism.

2.1 FILE. MANAGEMENT

For all types of secondary storage media, OS provides a uniform logic view of information storage. The logical unit for such storage is called, File, which is abstracted from the physical properties of the underlying device, on which it is stored. A File can be viewed as a named collection :..of, related information, mapped onto a secondary storage device. All :seondary media are non-volatile storage of information, where the Contents persist, even when power is turned off. A file could contain any :.kind of information; like source code, object code, executable binary code, :textual data, numeric data, graphics information or audio data etc.

File Attributes

Necessary Attributes of a file are:

NAME. It is a string of alphanumeric characters and some special characters like underscore. Most of the systems expect an alphabet as the first character of a file name and some systems are case sensitive to the file names. It is a displayable parameter, which is used for referencing a file

. IDENTIFIER. It is a unique identification of a file, which is internal to the system. OS uses it for making references to the related file.

TYPE. It is normally expressed as an extension to file name and indicates the type of file; like A.cpp indicates that it is a C++ Source Code File, A.obj indicates that it is an object file and A.doc indicates that it is MS WORD file.

LOCATION. This is expressed as access path of the file for locating it on the device like C:/WINDOWS/all files/A.cpp

SIZE. It indicates current size of the file in bytes ,or blocks.

PROTECTION. This is Access Control Information. It indicates `WHO' has got 'WHAT' Access Rights on the file i.e., who is owner of the file, who all are allowed to only read the _file and who all are permitted to modify, delete or execute etc: TIME, DATE, USER IDENTIFICATION. Records of information pertaining to the creation and last update of file. VERSION NUMBER. This indicates version number of a file, along with creation/update date/time.

File System

The file system consists of two distinct sub-coniporients:

(a) Collection' of .files,: each storing related data.

(b) Directory Structure under which the files are organized. This provides information regarding all the files in a system.

File Organization

File organization refers to the manner in which the records of a file are

organized on the secondary storage.

• A file is a set of logical records.

• It is allocated a disk storage space in terms of physical blocks (block size is normally 512 bytes).

• All basic operations like read, write etc., are in terms' of blocks.

• Last block of a file may not be fully occupied, resulting in loss of some storage space, called Internal Fragmentation. Average disk space lost due to internal fragmentation is of the order of half block per file. So, larger the block size, larger will be the disk space lost due to internal fragmentation. There is no way of recovering this loss.

File Control Block (FCB)

This contains complete information about a file; like its name, ownership, size, pointer to the storage blocks where the file is stored etc. Initially, it will be residing on secondary storage, but when the file is opened, its FCB is loaded into memory.

File-organization schemes

Sequential. The file records are stored in the same order as they occur physically in the file. Direct. The records are placed in any order, which is suited for application. The system supports random or direct access of any record in the file. Indexed. The records are arranged in a logical sequence according to a `key' contained in each record. The system maintains an index, which contains the, physical address of some of the records. Partitioned. This refers to a set of sequential sub-files, which together form a partitioned file. Each sequential sub-file is called a member of the partitioned file.

Major. OS Functions in File Management

1. Creation, manipulation and deletion of files as well as directories.

2. Protection of File System. OS controls the Access Rights of files and directories

3. Control sharing of files

4. Support backup and recovery of files

5. Support Encryption and decryption of sensitive files

Implementation of a File System

File system implementation has two major components—one is disk-based and the other is memory-based.

1. On-disk structure. The on-disk structure includes the following:

(a) Partition' Control. Block. For each partition, it contains details such as the number of blocks in the partitiOn, size of each block, number of free blocks, pointer to the free blocks' list, number of free File Control Blocks (FCBs) and pointer to the free FCBs..

(b) Directory Structure. The structure, in which the files are organized.

(c) File Control Blocks (FCBs). Each file has a File Control Block (FCB), which contains necessary information about the file; like its ownership, access rights and pointers to the disk blocks occupied by the file etc.

(d) Boot Control Block. It contains information needed by the system to boot an OS from that partition. It is typically the first block of a partition.

2. In-memory Structure. The in-memory structure contains following informations:

(a) Partition Table. It contains information about each mounted partition.

(b) Directory Structure. It contains information about the recently accessed directories, along with the pointers to partition table.

(c) System-wide 'Open-File-Table'. Contains a copy of the FCB of each open file. Each entry in the table will also contain a "File-Open-Count", which indicates the number of processes, currently accessing the file.

(d) Per-process 'Open-File-Table' „This table contains a pointer to the appropriate entry in the System-wide 'Open-File-Table'. It also contains a Pointer that points to the file location, where it would be accessed next. All operations are performed through this pointer only. Note that different processes accessing a file concurrently will have different pointers. But these operations must be compatible with each other.

2.2 FILE OPERATIONS

1. Creation of a file. A new file can be created either by a System Call made from a process or by a System Command issued by an interactive user. The file system will respond to the call/ command, as follows:

• Assign a new FCB to the file to be created.

• Allocate necessary, disk-storage-space to the newly created file.

• Load the appropriate directory, under which the new file is to be created, from disk to memory.

• Update the directory by linking the new FCB to the directory.

• Write back the updated directory onto the disk.

2. Opening a file for access.When OS receives a call to Open a File, 'it would respond as follows:

• Search the directory structure for the given file name. To speed up this search operation, some parts of directory structure are always cached into the memory.

• Once the file is found, its FCB is copied into the System-wide 'Open-Files-Table'. The 'Open-File-Count' is incremented by one. Then, an entry is made

• Make entry in the Per-process 'Open-Files-Table'. This will have a pointer to the corresponding entry in the System-wide 'Open-File-Table'. This table also contains file pointer for accessing the file. Initially, it will point to the -Beginning of File.

3. Read. To read data from a specified location in a file. In sequential read, the file pointer will be automatically repositioned at the next record, after the read operation is completed.

4. Write. To write at the current position of the file pointer. If file pointer is at End Of File (EOF), then the file-size is automatically increased.

5. Seek. This repositions the file pointer to the specified location. This is done in direct access mode.

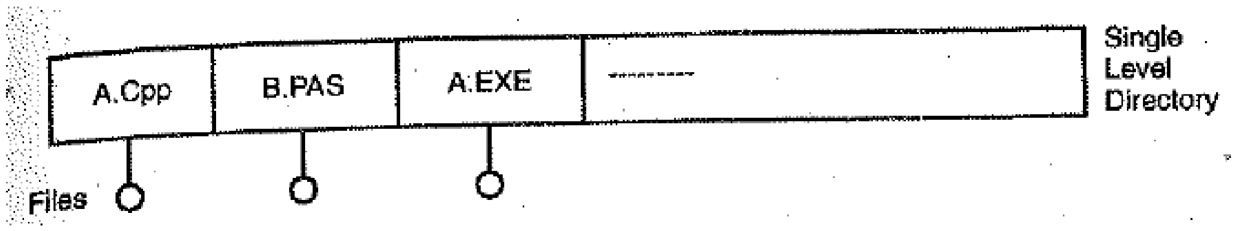
6. Append. To append the information at the end of file

7. Close File. When a process issues a Close-File call, the OS responds by removing the file entry from the per-process open file table. Also, the 'open-file..count', in the System-Wide 'Open-File-Table', is decremented by one. If this count becomes zero; the file entry is removed from the System-wide 'Open-File-Table'.

8. Delete File. When a file is deleted, the blocks allocated to it are returned back to the system. Its FCB is de-linked from the directory and FCB is declared to be free.

2.3 DIRECTORY STRUCTURE

Single Level Directory This is the simplest structure. All the files are contained in the same directory structure.



Limitations of Single-Level Directory

1.

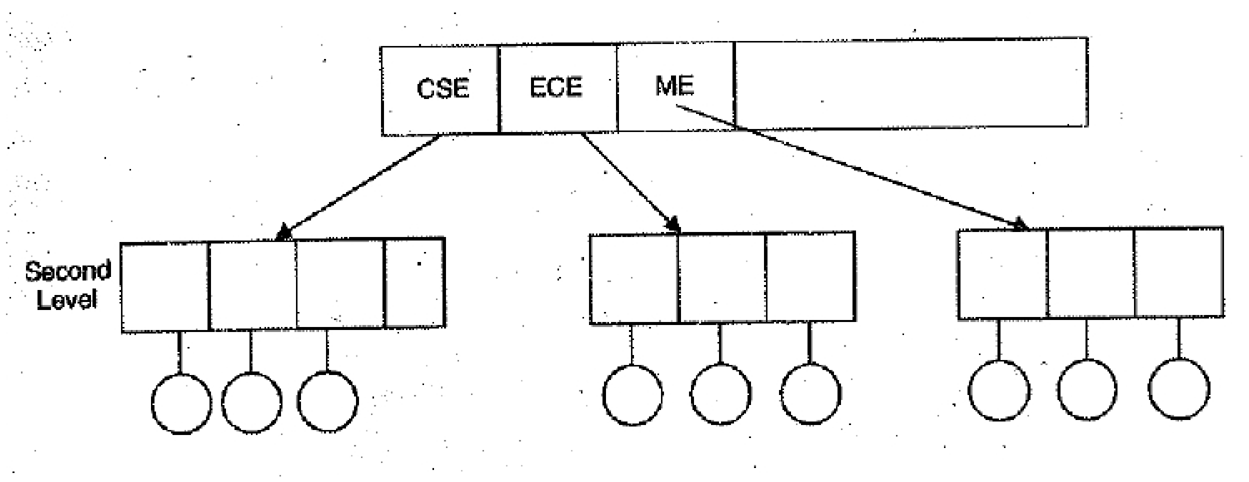
Single Level Directory

1.It becomes highly unwieldy when system has more than one users or the numbers of files is large.

2. Two files by different users, with same name, cannot be accommodated; one of the files will need to be renamed.

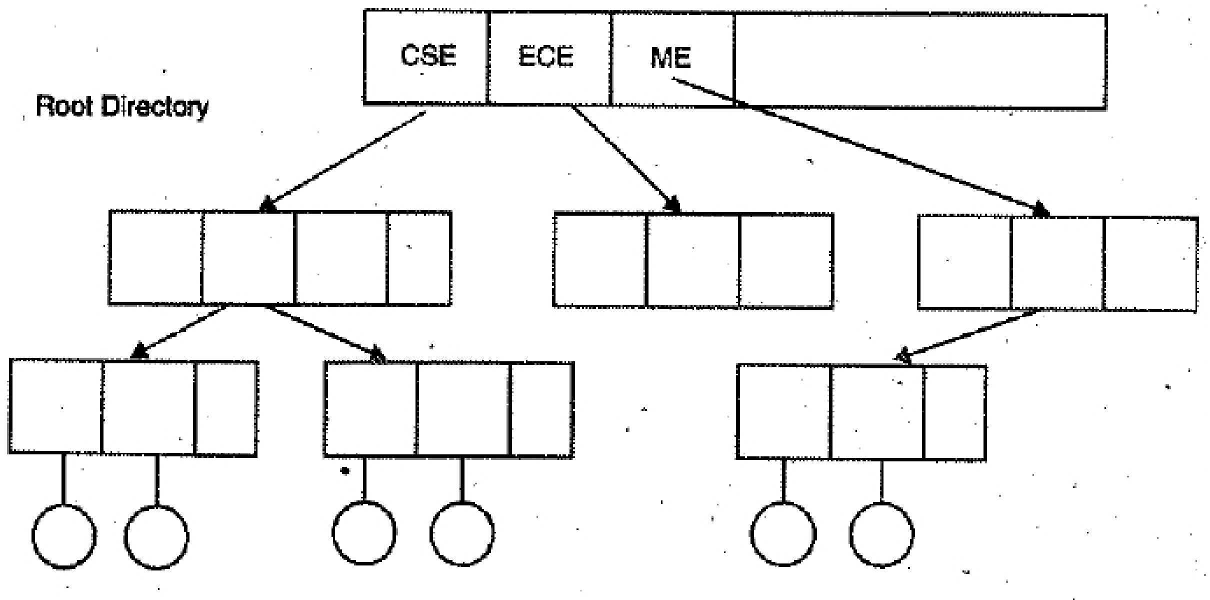
Two Level Directory

This solution can be used to create multiple User File Directories (UFDs), one for each user, under a Master Fib Directory (MFD). The 1Y1FD is indexed by the User. Name or User Account Number, to provide €111 access to the relevant UFD. When a. user references any of its files, a': search is carried out only in the User File Directory (UFD) of that particular User.



Tree Structured Directory

The two-level directory is in fact 'a two-level tree. The Tree Structured Directory is a generalization of the Two Level Directory and it forms a tree of an arbitrary height. This permits the Users to create their own sub-directories under the respective UFDs. A directory may contain a set of files and a set of sub-directories. Each file in the system has a unique access path. Users may use absolute path name or relative path name (relative to the current directory being accessed by the user) to access sub-directories. New sub-directories can be created and existing sub-directories can be deleted. Some systems insist that a sub-directory can be deleted only if it is empty.

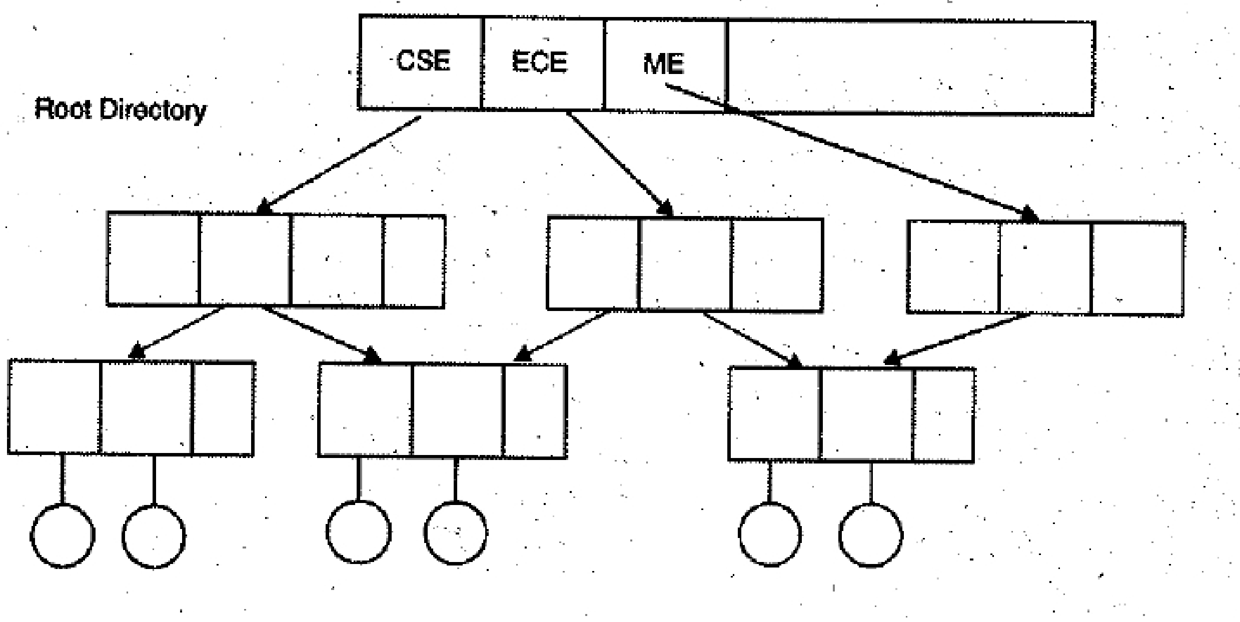


A-cyclic Graph. Directory -

Suppose multiple users are working on a project, the project files c be stored in a. common sub-directory of the multiple users. This t of directory is called A-cyclic Graph Directory. The common directory will be declared a shared directory. Same way, system may provid shared files. Unlike the Tree Structure, a file may have multiple acce paths. However, the graph will contain no cycles. With shared files changes made by one user are made visible to other users. The majd advantage of a-cyclic graph directory is support for shared files an directories. The issues to be resolves are

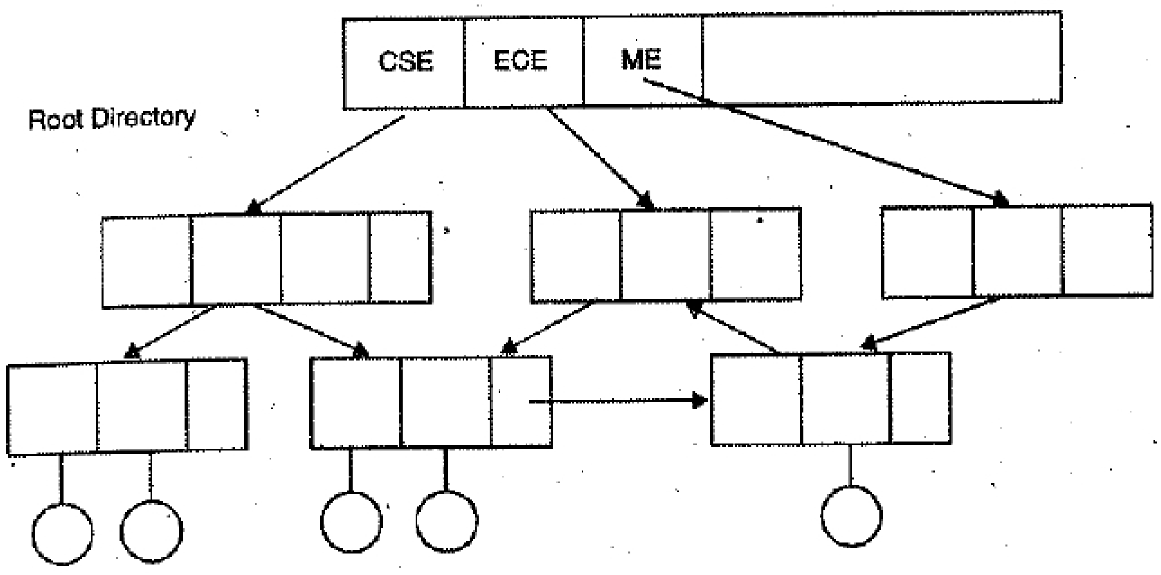
(a) A file may now have multiple absolute paths. So, distinct fi names may refer to the same file.

(b) Whenever, a shared directory/file is deleted, all pointers- to the directory/file are to be removed.



General Graph Directory

General Graph Directory permits cycles. One major disadvantage is that a poorly designed search algorithm may get into infinite loop; while searching for a file.



2.4 DISK SPACE ALLOCATION METHODS

Contiguous Allocation

A file- is allocated a set contiguous set of blocks.

Advantages

1. The number of disk-seeks to access all blocks of a file is minimal, since the blocks reside on same or neighbouring tracks. So, access will be more efficient.

Disadvantages

1. Finding contiguous space of a required size space for a new file will be time consuming.

2. Subsequent 'extension of a file will be difficult. The whole may have to be relocated elsewhere on the disk.

3. The scheme suffers from external fragmentation. System will have to perform de-fragmentation quite often0, to recover the disk space rendered un-usable by external fragmentation.

Discontiguous Space Allocation

The schemes used, for discontiguous space allocation on disk, are:

(a) Linked Blocks Allocation

(b) Indexed Allocation, using:

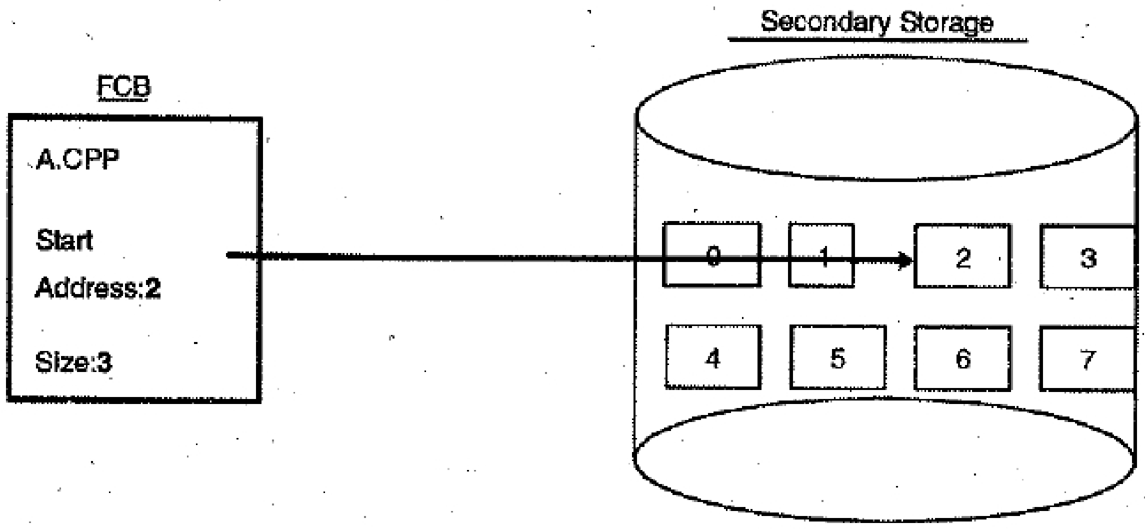
(i) 'Linked Index Blocks

(ii) Hierarchical Indexing

(iii) I-Nodes

Linked Blocks Allocation

The disk blocks allocated to a file are linked to each other; the address of next block being contained in the previous one. The FCB contains pointer to first block and last block in the link list. The pointer to last blocks is handy at the time of appending new blocks to the list. The disk blocks, allocated to a file, may be scattered all over the disk space.



The FCB indicates the start Address as 2 and size of the file as 3 in the above case. So, the file A.CPP occupies three contiguous blocks i.e., 2, 3 & 4.

Advantages

1. This method does not suffer from external fragmentation; only last block allotted to a file may not be fully occupied (internal fragmentation). So, disk storage space is optimally utilized. Disadvantages 1. Accessing such files is more time consuming, since address of next block needs to be determined from the previous block. –

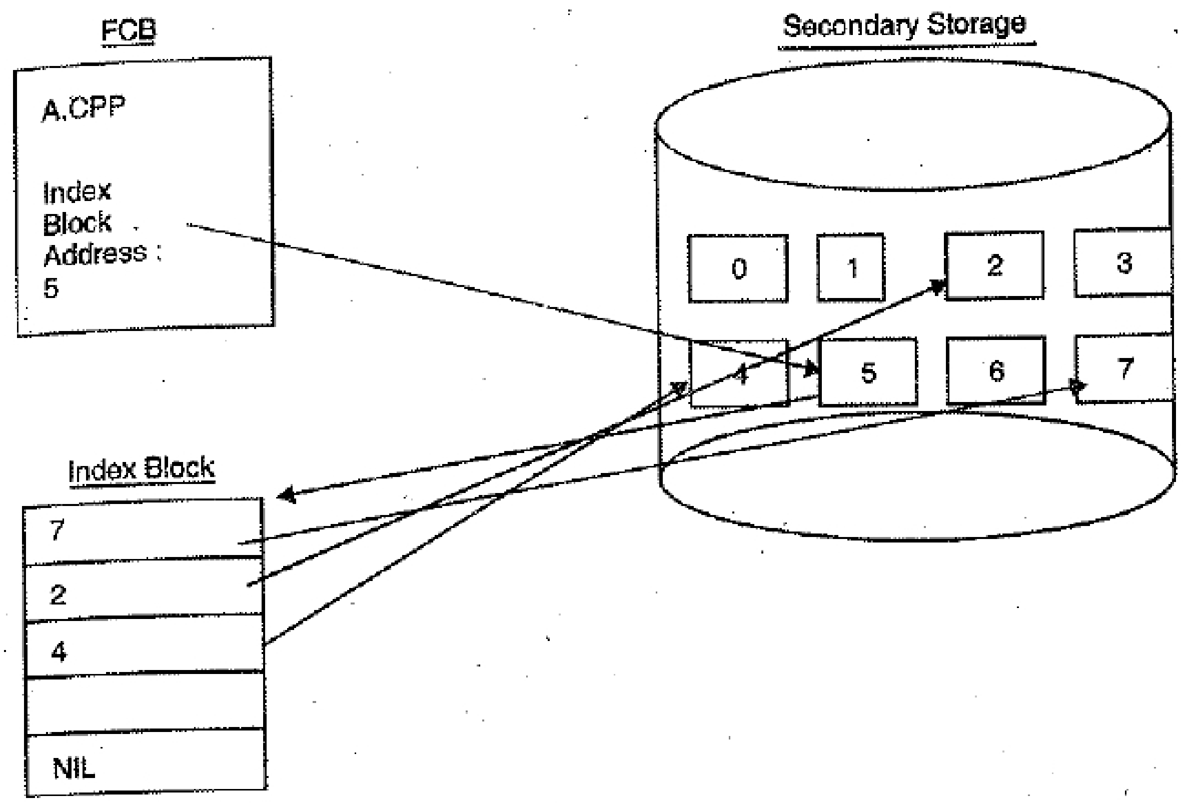
2. Number of disk seeks to access all blocks of a file may be large. Sophisticated disk scheduling will be required to optimize the head movement during seeks.

3. The allocated blocks cannot be accessed randomly.

Indexed Allocation

The method supports random access of the allocated blocks. Each file has its own index block, which is an array of disk block addresses. Indexing can be of the following types:

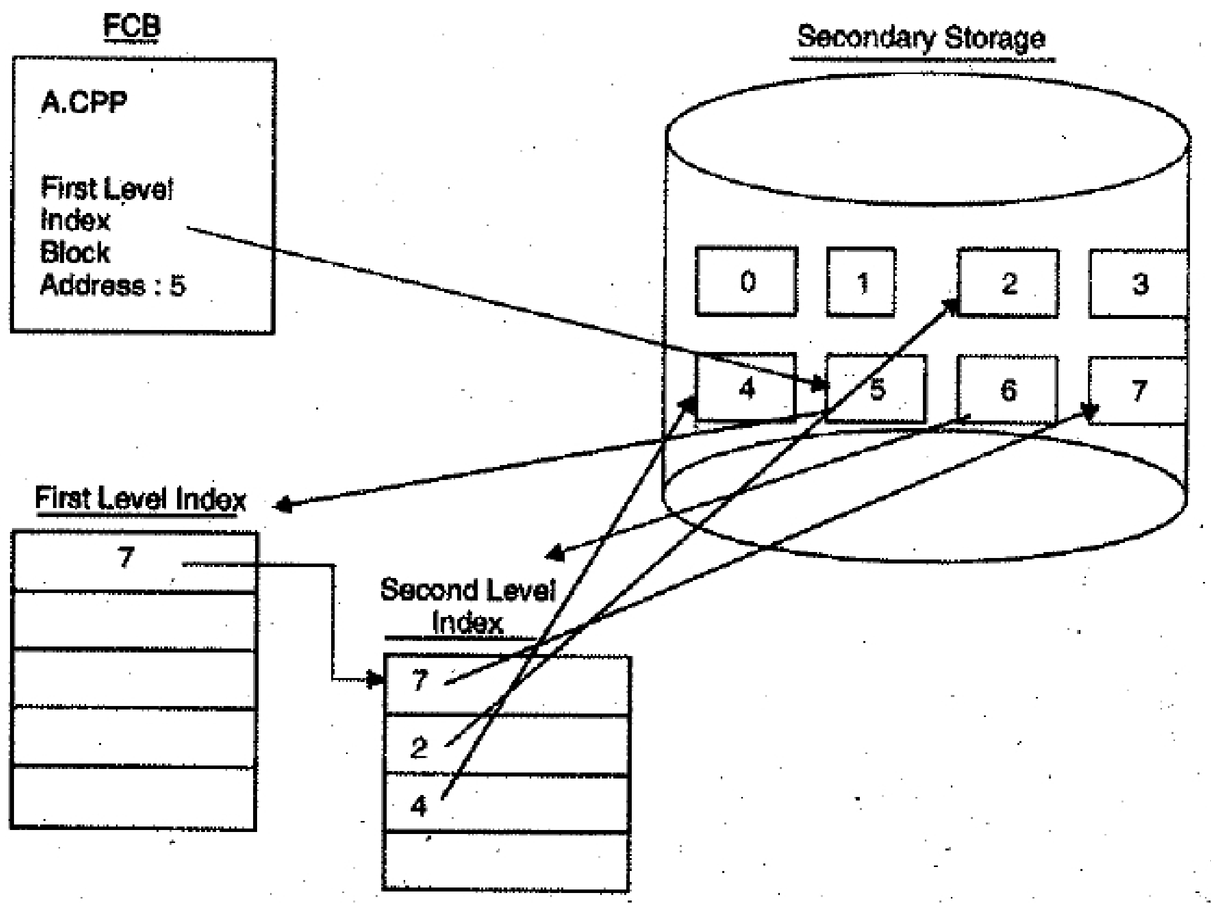
(a) Indexed Allocation, using Linked Index Blocks. The **FOB contains the Disk Address of the first Index Block**. More than one index blocks may be used by linking together the index blocks. For this, the last location in the Index Block indicates the disk address of next Index Block. This address will be NIL in the last Index Block.



The above diagram indicates that the Index Block of file A.CPP is in disk block number 5. First, the Index Block is loaded and then the file is accessed via the disk addresses contained in the Index File. The addresses are in the same sequence as the logical sequence of file blocks. The above Index Block indicates that the file is stored in blocks no. 7, 2 & 4 in that order. Also, the last address being NIL indicates that there are no more Index Blocks.

(b) Hierarchical indexing. This method uses multiple level indexing. The FCB contains the disk address of the First Level Index Block. The first level index block contains pointers to the disk blocks containing Second Level Index Blocks. The second level index block point to the disk blocks allocated to the file. The depiction indicates that the first level Index Block is stored in. Disk Block .# 5. First this block is loaded. The first entry in this block indicates that one of the second level index blocks is stored in' disk block # 6. Then, this second level index block is loaded. This block indicates the file is stored in disk blocks 7, 2 & 4 in that order. Then, these file blocks are accessed.

(c) index Nodes (I-Nodes). This scheme is used in UNIX. Each file has an I-Node stored on the disk. When a file is opened, its I-Node is loaded from disk to main memory. The I-Node contains file attributes and some addresses of the disk blocks allocated to the file. For small files, addresses of all the allocated disk blocks are accommodated in the I-Node itself. However, for medium and large sized files, it is not possible to accommodate all the disk addresses in the I-Node itself.



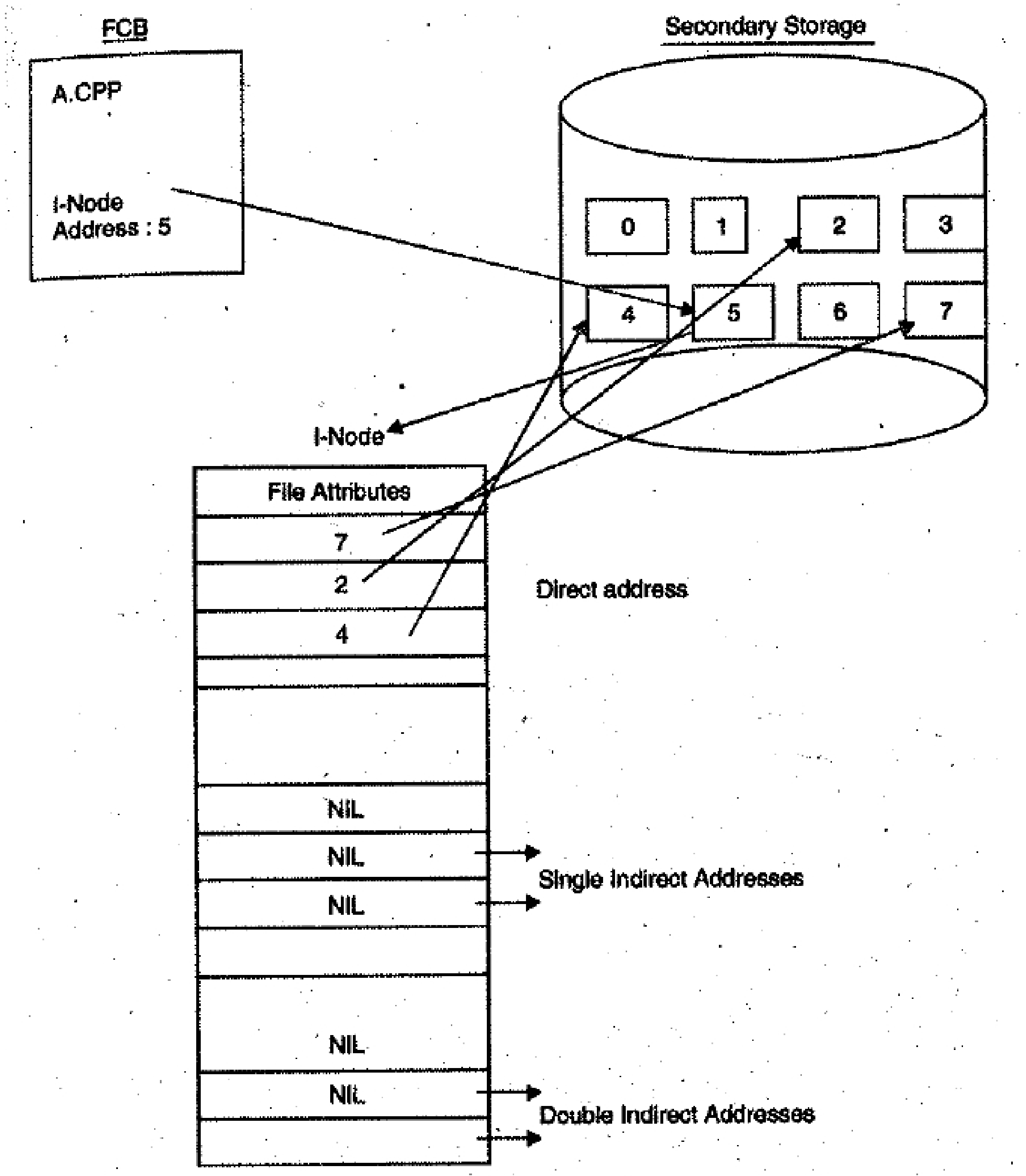
The single indirect pointer, when not set to NIL, will point to index..; blocks, which will contain addresses of file blocks. .:, The double indirect... pointers, when not set to NIL, will contain diski: addresses of first level index blocks, which will further. contain tiled disk. addresses of second level index blocks and that will further contain .5

the addresses of file blocks. For small files, the direct addresses are sufficient to address the disk: blocks allocated to a file. The single indirect pointers and double indirect pointers would be set to NIL.

However, for medium size files, the single indirect pointers are also used along with direct pointers. The access time via single indirect pointers will be larger as compared to direct pointers, due to one levelI of indirection. For large files, it will also use double indirect pointers, which have still larger access time**.**

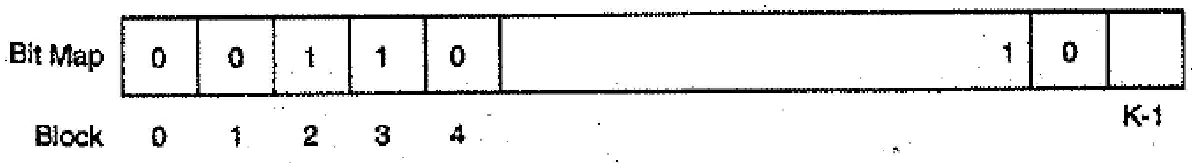
Advantages of Indexed Allocation

Indexed allocation supports direct access of allocated random blocks, without suffering from external fragmentation.



2.5 FREE DISK-SPACE MANAGEMENT

The OS maintains a pool of free disk blocks. Whenever, a new file is created, blocks are alloCated from the free blocks' pool. Whenever, a file is deleted, its allocated blocks are declared free and put in the free blocks pool. Free blocks pool may be implemented as: (a) Bit Map. The Bit Map is stored at a known location on the disk. The Bit Map has a bit associated with each block. It can be set to '1', if the block is free; else to '0' if the block is already allocated. For accessing the information, the Bit Map is loaded' into memory

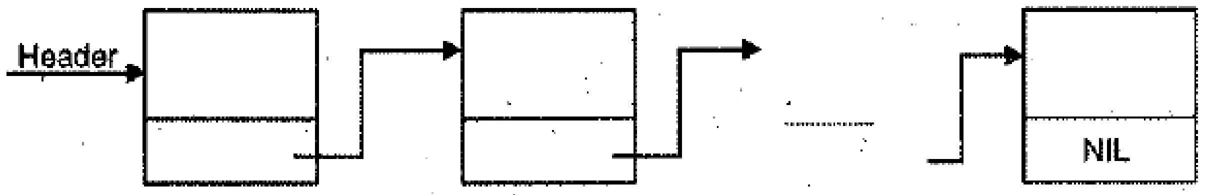


Assume 1 : Free

0 : Already Allocated

A System can use opposite convention also i.e., 0: Free, I: Allocate The advantage is that information about each block can be accommodat in a single bit. It requires special bit manipulation routines to maintai and access the bit map. (b) Free Blocks' Link List. This method maintains a linked list of at free blocks on the disk. For accessing the information in the linked list, it is loaded into physical memory. Whenever, an

allocation is to be done, blocks are removed from the head of th list and allocated. It occupies large space. Also, traversing the. link list will be time-consuming.

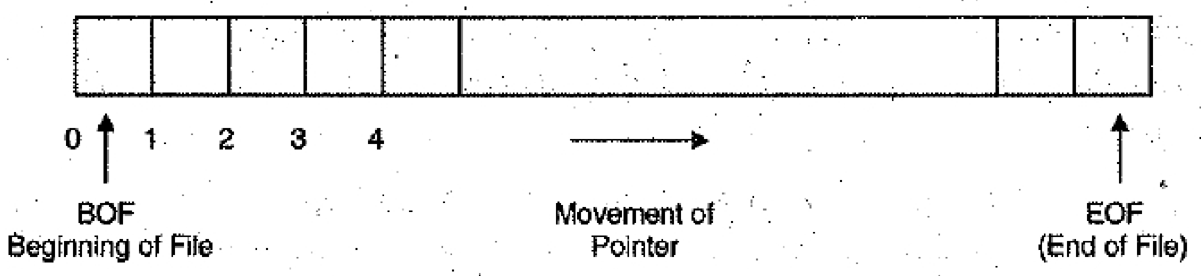


2.6 FILE ACCESS METHODS

File access methods depend on the underlying file organization. Commonly used file-access methods are:

Sequential Access

It is based on a tape-model of file. When a file is opened for operation, the File Pointer is positioned at the beginning of file. When a block has been read, the pointer is automatically shifted to the next record. When a file is opened for 'Write' operation, the pointer is positioned at the EOF. A 'Write' operation appends a record to the EOF and advances the pointer to the end of newly appended record, whichnow is EOF. Some systems may support skipping of some records' forward and backward. When File, Pointer is pointing to the EOF, it is sensed by the pointer and returned to the calling process.



Advantage

1.simple implementation

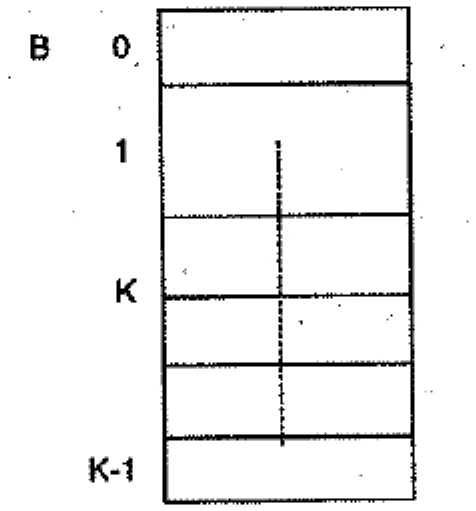
Disadvantage

1.The access being sequential is not efficient

2. Random access of a Record is not feasible. Average time to access a particular record will be equal to the time required to access half the file.

Direct Access or Random Access or Relative Access

This access is possible, only if a file is made up of fixed-length logical records. A read/write request will include a record number. The record numbers are kept sequential. Such files are indexed by record number and support random or direct accessing of records, jumping from one record to another forward or backward. It supports absolute or relative indexing of records. This access method is most suitable for DBMS files.



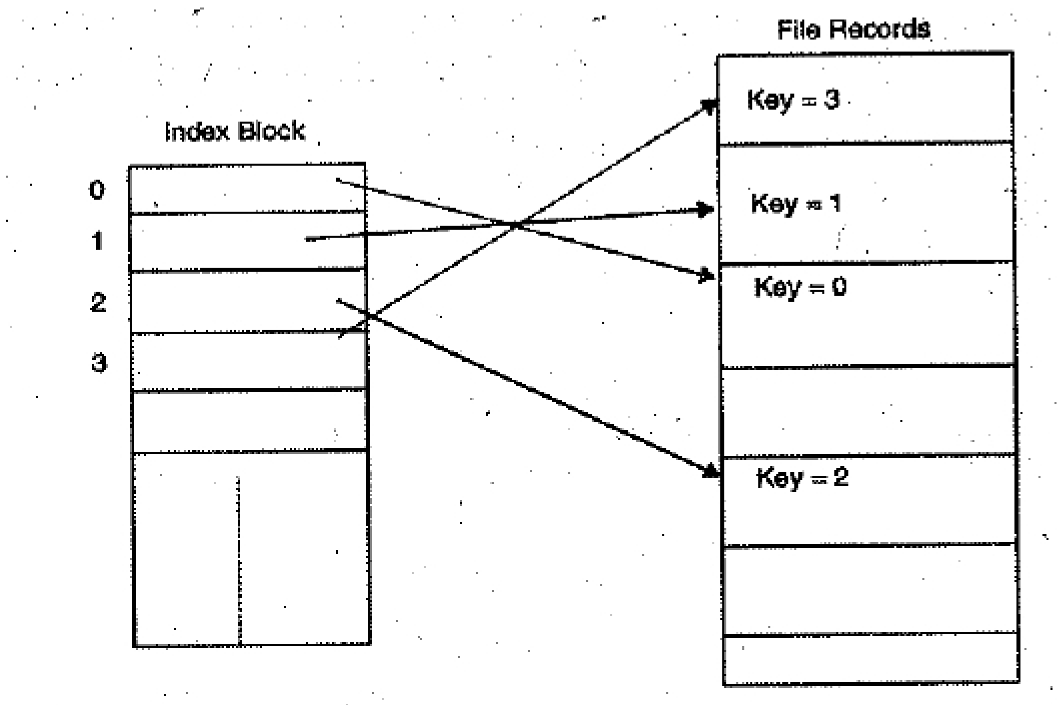
Suppose B = Base Address of a file, with records numbered 0.. N-1

S = Size of each record, the address

Then, Address of Record "K" = B + K * S

Indexed Access

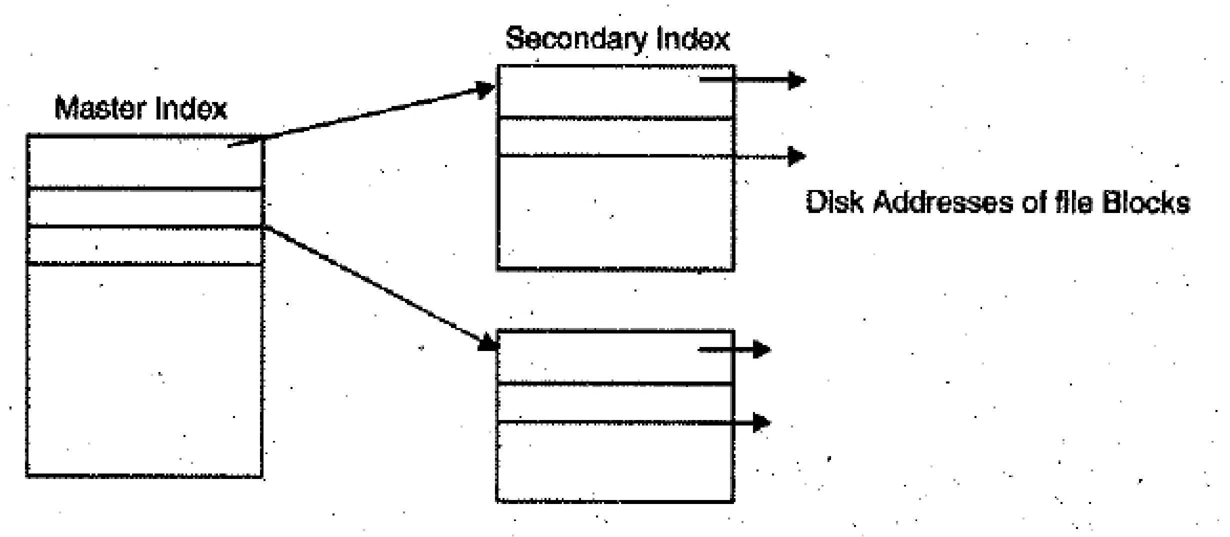
The • records are arranged in a logical sequence according to a key contained in each record. The system maintains an index containing the physical address of certain records. By using the key attribute, the records can be indexed directly. This is most commonly used method in DBMS files.



Indexed Sequential Access Method (ISAM)

It uses a small master index that points to disk blocks of a secondary index. The secondary disk blocks in turn point to the actual disk blocks allocated to a file. The file is kept sorted on a defined key. To find match for a key, we first make a binary search in the master index table; which provides a block number of the secondary index. The secondary block is read into memory and a binary is made in the secondary index to determine address of the file block containing the key value. Finally, the block is searched sequentially to determine the record containing the key value.

The pointers in the Master Index and Secondary Index Blocks are sorted as per the key values. Each File Block contains a number of records, whic h are also sorted as per the key value.



.

The FCB contains the disk address of Master Index Block So, while searching .for a key value, perform the following steps:

• First load the Master Index Block

Carry out binary .search for the key value in the Master Index Table and locate the disk address of desired Secondary Index Block.

• Then load the desired Secondary Index Block.

• Carry out binary search for the key value in the Secondary Index Table and get the disk address of desired File Block.

• Then load the desired file block.

Finally, carry out a sequential search with the key value in the file block to locate the desired record.

SHARING OF FILES

Sharing of Files by Multiple Users

The OS impleinents the concept file/directory owner (or user) and group.The owner is the user who can change the file/directory attributes, grant access rights to other users and has the maximum access rights over the file/directory: Group is a sub-set of users, who granted a sub-set of access rights over the file/directory. Other uers (other than the owner arid group users) may also have a small of access rights over the file/directory.

The access rights may be read, list, write, append, delete, execute, change attributes etc. Whenever, a user makes a call to access a file/directory,. Os will check access rights of that user before granting access.

file Sharing on Remote File Systems

'It is common in distributed systems. The OS may support a Distributed File System, in which remote directories and files are made visible to :the user via the underlying networking. Users may be granted access rights on remote files and directories.

Client Server Model

It is common in networked systems like world wide web (www). The 'servers declare the files which are available to the clients. A server :serves multiple clients. So, a file at a server may be accessed simultaneously by multiple clients. Clients are identified at the servers by IP Addresses. Additional authentication methods may be used to validate the clients.

2.8 FILE PROTECTION MECHANISM

Any real secure system must have some protection mechanism or mechanisms to enforce the access rights required 'by the system. In the case of Unix, we rely on the memory management unit to enforce access rights for memory segments, and we rely on the file system to enforce the access rights for files. In this section we will cover the techniques that are used in operating systems to protect file systems.

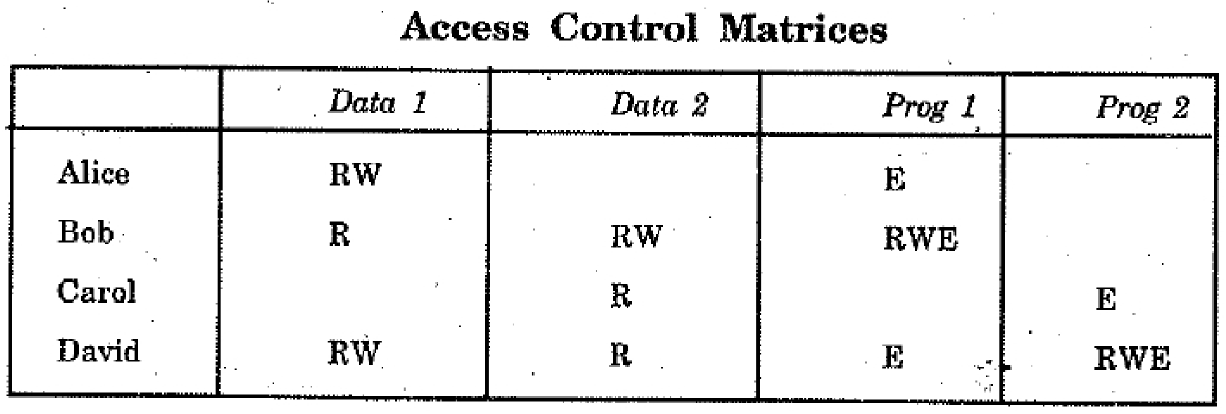
Protection Domains

A computer system may have objects that need protection. These objects may be hardware (as disk drives, memory segments, printers etc.) or software (as processes, semaphores, files, databases etc.). Each object has a unique name by which it can be identified and a set of operations that can be carried out on it. A Protection Domain is a set of (object, rights) pairs. At any given time this pair specifies an object and a set of operations, having rights to perform on it. It is possible for a single object to be in multiple domains at the same time and objects can also.switch from one domain to another during execution, which is highly system dependent. Let us take an example of UNIX. In UNIX , the domain of a process is defined by its uid and gid. For any combination, it is possible to make a complete lists of all objects that can be accessed, and whether they can be accessed for reading, writing, or executing. Two processes with the same combination will have access to exactly the same set of objects and with different combination will have access to a different set of objects, although overlapping may be possible in most of the cases.

Access Control Matrices

There are two practical ways to implement the access matrix, access control lists and capabilty lists. Each of these allows efficient storage of the information in the access matrix.

Access Control Matrices



Access Control Matrices

	Data 1	*Data 2*	*Prog 1*	*Prog 2*
Alice	RW		E	
Bob	R	RW	RWE	
Carol		R		E
David	RW	R	E	RWE

In a large system the matrix will be enormous in size and mostly sparse.

Access Control List

It is the column of access control matix

Advantage

Easy to determine who can access a given object..

Easy to revoke all access to an object.

Disadvantages

1. Difficult to know the access right of a given subject.

2. Difficult to revoke a user's right on all objects. Used by most mainstream operating systems.,

Capability List

It is the row of access control matrix.

A capability can be thought of as a pair (x, r) where x is the name of an object and r is a set of privileges or rights.

Advantages

1. Easy to know the access right of a given subject.

2. Easy to revoke a user's access right on all objects.

Disadvantages

1. Difficult to know who can access a given object. ,

2.Difficult to revoke all access right to an object. number of capability-based computer systems were developed, but have not proven to be commercially successful.

Protection Models

Protection matrices are not static. They frequently changes as new objects are created, old objects are destroyed and owners decide to increase or restrict the set of users for that object. At any time the matrix determines that what are the access rights of a process in a given domain.

SUMMARY

• The file system resides permanently on secondary storage, which is designed to hold a large amount of data permanently. The most common secondary storage device is disk. Physical disks may be segmented into partitions to allow multiple file systems per spindle and these file systems are mounted onto a logical file system to make them available for the further use.

• The various files can be allocated on the disk in three ways: through contiguous, linked or indexed allocation strategies..

• File access methods depend on underlying 'file organization. The most commonly used file-access methodS are: sequential access, Direct access and indexed sequential access methods.

REVIEW QUESTIONS

1. What are various file attributes?

2. Discuss several file system implementation strategies.

3. How does DMA increase system concurrency? How does it complicate the hardware design?

4. Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of allocation strategies (contiguous, linked and indexed) answer these questions:.

(a) How is logical to physical address mapping accomplished? For the indexed allocation, assume that a file is always less than 512 blocks long).

(b) If we are currently at block number 10 (i.e., last block accessed was 10) and want to access logical block 4, how many physical blocks must be read from the disk?

5. Could you simulate a multi-level directory structure using a single-leve0 directory structure, in which arbitrarily long names can be used? If your-. answer is yes, explain how. Contrast this scheme with the multi-leve4 scheme. How would your answer change, if file names are limited to seven characters.

6. Briefly explain the various disk scheduling policies.

7. What is the Interrupt Service mechanism? Explain the concept of Software

interrupt.

8. Explain various operations possible on File.

9. Discuss various File Allocation Strategies for disk space management.

10. A sequential access file has fixed size 15 byte records. Assuming the first-;;; record is record 1, the first byte *of record 5 will be at what logicallocation?

11. Which allocation scheme woul work best for a file system implemented on a device that can be accessed sequentially, a tape drive, for instance.

12. Consider a system that supports the strategies of contiguous, linked, and indexed allocation. What criteria should be used in deciding which strategy is best utilized for a particlar file?

13. How do caches help to improve performance? Why do systems" not use more or larger caches if they are so useful?

FURTHER READING

1. Operating Systems: A practical Approach: Rajiv Chopra, S. Chand Publisher, 2010 •

2. Operating Systems: PrincipleS and Design: Pabitra Pal Choudhury, PHI Learning
3. Operating Systemes and. Software Diagnostics: Ramesh Bangia and Balvir Singh, Laxmi Pub., 2007 .

4. Principles of Operating Systmes: Sri V. Ramesh, Laxmi Pub.; 2010

UNIT III

CPU SCHEDULING

* STRUCTURE *

Learning Objectives

Scheduling Concepts

Schedulers

Scheduling Algorithms

LEARNING OBJECTIVES ---

After reading unit, you will be able to:

• discus the use of CPU

• define the CPU seheduling

• define scheduling concepts of the CPU.

• know about schedules and its operating systems

• explain your idea of scheduling Algorithms.

• discuss the First cause first -science scheduling and chartist-Job. First Scheduling.

3.1 SCHEDULING CONCEPTS

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The idea is quite simple. A ;`;process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU would normally Sit idle while the process waited for the completion of the event. In a multiprogramming system, several processes are kept in memory at a time. When one process has to wait, the operating system takes the CPU away from that process and gives it to another process. This pattern continues. Every time one process has to wait, another process may . take over use of the CPU. .2 The benefits of Multiprogramming are increased CPU utilization and `Iligher throughput, which is the amount of work accomplished in a given :,tline interval.

Scheduling Queues

As processes enter the system, they are put into a job queue. This.: queue consists of all processes residing on mass storage awaiting:; allocation of main memory, The processes that are residing in main=i memory and are ready and waiting to execute are kept on a list calledi the ready qUeue. This BA is generally a linked-list: A ready-queuel 'header will contain pointers to the first and last process control block in the list. Each PCB has a pointer field that points to the next procesi:1 in the ready queue.

There are also other queues in the system. When a process is allocated:; the CPU, it executes for a while and eventually either quits or waits.for the occumence of a particular event, such as the completion of an :j I/O request. In the case of an I/O request, such a request may be to disk. Since there are many processes in the system, the disk may be busy with the I/O request of some other process. Thus, the process may have to wait for the disk. The list of processes waiting for a particular,;! I/O device is called a device queue. Each device has its own devicel queue. A common representation for a discussion of process scheduling is al queueing diagram, shown below. Each rectangular box represents a..i queue. Two types of queues are present : the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processess in the system.

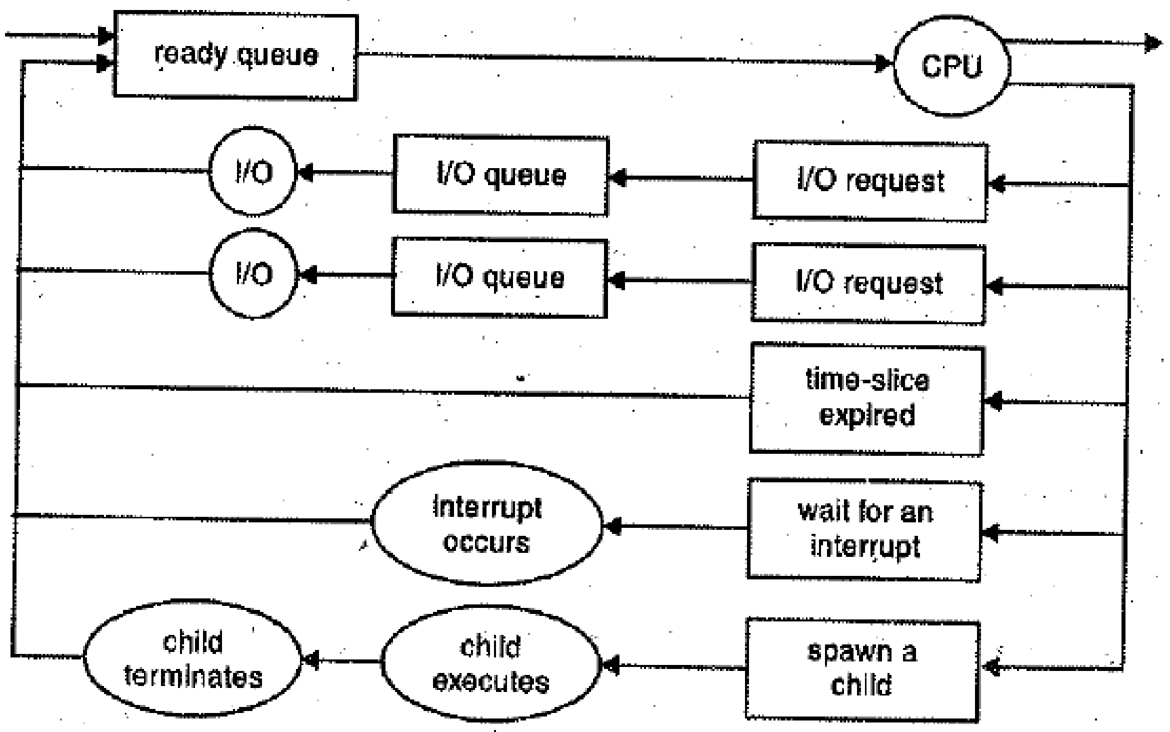


Figure 3.1

A new process is initially put into the ready queue. It waits in the ready qiieue until it is selected for execution and is given the CPU. Once the process is allocated the CPU and is executing, one of several

events could occur:

• The process could issue an I/O request, and then be placed in an I/O queue.

- The process could create a new process and wait for its' termination.
- The process could be forcibly removed from the CPU, as a result of an interrupt, and be put back into the ready queue.

In. the first two cases, the process eventually switches from the waiting :`:;state to the ready state and is then put back into the ready queue. A process continues this cycle until it terminates, at which time it exits '.:from the system.

3.2 SCHEDULERS

.A process migrates between the various scheduling queues throughout its life time. The operating system must select processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.

In a batch system, there are often more processes submitted than can be executed immediately. These processes are spooled to a mass storage device (typically a disk), where they are kept for later execution. The long-term scheduler (or job scheduler) selects processes from this pool and loads them into memory for execution. The short-term scheduler (or CPU scheduler) selects from among the processes that are ready to execute, and allocates the CPU to one of them.

The primary distinction between these two schedulers is the frequency of their execution. The short-term scheduler must select a new process for the CPU quite often. A process may execute for only a few millisecond's before waiting for an I/O request. Often, the short-term scheduler executes at least once every 10 milliseconds. Because of the short duration of time between executions, the short-term scheduler must be very fast. If it takes 1 millisecond to decide which process to execute for 10' milliseconds, then 1/ (10 + 1) = 9 per cent of the CPU is being used simply for scheduling the work(overhead).

The long-term scheduler, on the other hand, executes much less frequently. There may be minutes between creation of new processes in the system. The long-term scheduler controls the degree of multiprogramming (the number of processes in memory). If the degree of multiprogramming is to be stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. Thus, the long-term scheduler may need to be invoked only when a process leaves the system. Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.

On some systems, the long-term scheduler may be absent or minimal. Per example, time-sharing systems often have no long-term scheduler, . but simply put every new process in memory for the short-term

scheduler. The stability of these systems depend either on a physical limitation or on the self-adjusting nature of human users.

Some operating systems, such as time-sharing systems, may introduce;:i an additional, intermediate level of scheduling. This medium-term::: scheduler is diagrammed below. The key behind a medium-term• scheduler is that it sometimes it can be advantageous to remove::; processes • from memory (and from active contention of the CPU) and thus to reduce the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is often called swapping. The process is swapped out and swapped: in later by the medium-term scheduler. Swapping may be necessary to improve the process mix, or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

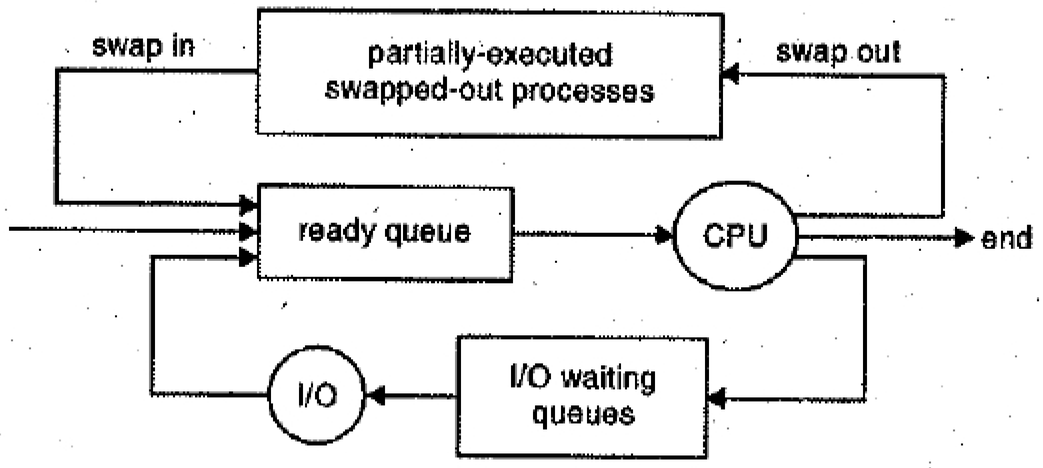


Figure 3.2

CPU SCHEDULING

Scheduling is a fundamental operating system function. AlmoSt all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating system design.

CPU-I/O Burst Cycle

The success of CPU scheduling depends on the following observed property of processes : Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states. Process execution begins with a CPU burst. That is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on.• Eventually, the last CPU burst will end with a system request to terminate execution, rather with another I/O burst.

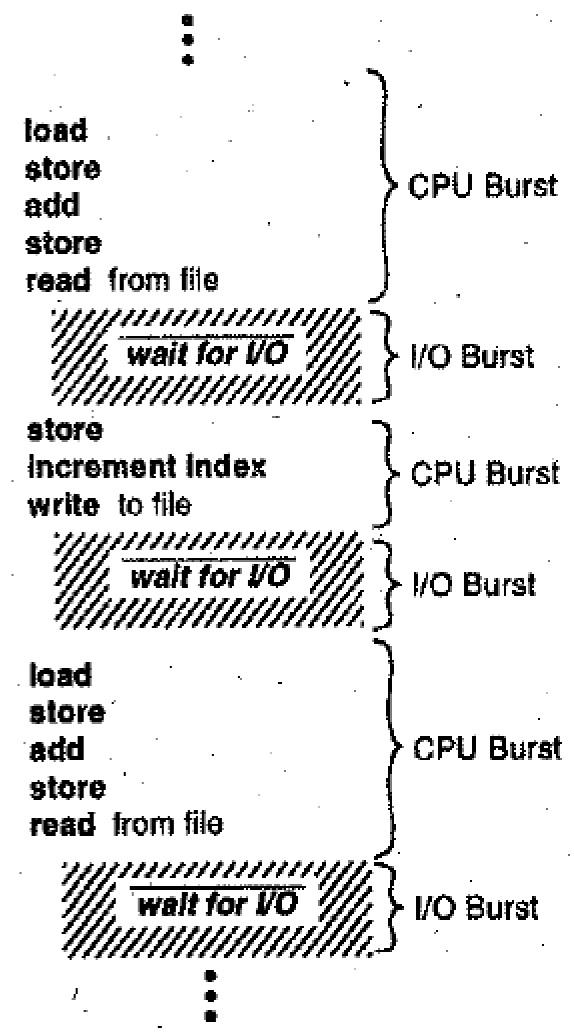


Figure 3.3

The durations of these CPU bursts have been measured. Although they vary greatly from process to process and computer to computer, they .tend to have a frequency curve similar to that of 'below.

The curve is generally characterized as exponential or hyperexponential , There is a very large number of very short CPU bursts, and there is a small number of very long CPU bursts. An 1/0 bound program would typically have many very short CPU bursts. A CPU bound program might have a few very long CPU bursts. This distribution can be quite important in selecting an appropriate CPU scheduling algorithm.

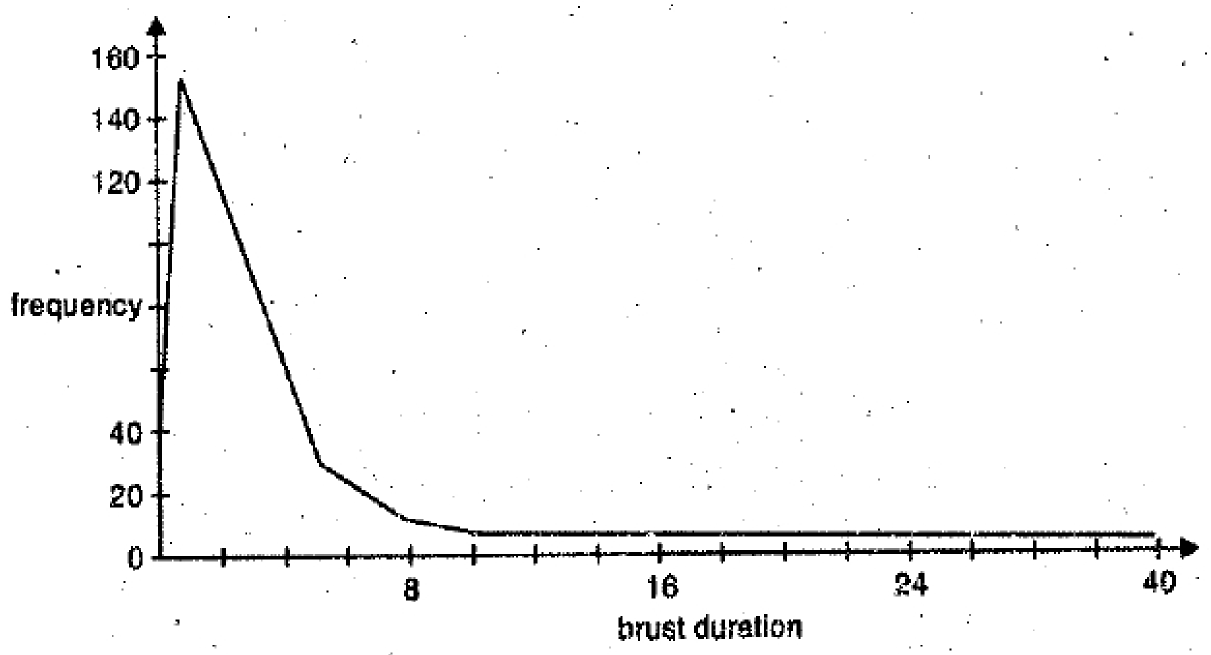


Figure 3.4

Whenever the CPU becomes idle, the operating system must select one :of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are area to execute, and allocates the CPU to one of them. Note that the ready queue is not necessarily a First-In-First-Out(FIFO queue. As we shall see when we consider the various scheduler algorithms, a ready queue may be implemented as a FIFO queue, priority queue, a tree, or simply an unordered linked-list. Conceptually, however, all the processes in the ready queue are lined up waiting for a chance to run on the CPU. The records in the queues are generally PCB's of the processes.

Scheduling Structure

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

1. When a process switches from the running state to the waiting, state (for example, I/O .request), invocation of wait for the termination of one of the child processes).

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).

3. When a process switches from the waiting state to the ready state, (for example, completion of I/O).

4. When a process terminates.

For circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. This, however, is not the case for circumstances 2 and When scheduling takes place only under cirumstances 1. and 4, we say the scheduling scheme is non-preemptive; otherwise, the scheduling scheme is preemptive. Under non-preemptive scheduling, once the CPU' has been allocated to a process, the process keeps the CPU until it: releases the CPU either by terminating or by switching to the waiting'; state.

Context-Switch

Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task' is knOwn as a context-switch. Context-switch time is pure overhead. IV: varies from machine to machine, depending on the memory speed, the number of registers, and the existence of special instructions (such a single instruction to load or store all registers). Typically, it ranges from 1 to 100 microseconds. Also, the more complex the operatingy system, the more work Must be done during a context-switch-

Dispatcher

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

1. Switching context
2. Switching to user. mode
3. Jumping to the proper location in the user program to restart that program.

Obviously, the dispatcher should be as fast as possible.

3.3 SCHEDULING ALGORITHMS

CPU scheduling deals with the problem of deciding which of the processes in 'the ready queue is to be allocated to the CPU. There are many different CPU scheduling algorithms. These algorithms have different properties and may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we, must consider the properties of the various algorithms.

Many criteria have been suggested for comparing CPU scheduling algorithms. Which characteristics are used for comparison can make a substantial difference in the determination of the best algorithm. Criteria that are used include the following:

• **CPU utilization.** We want to keep the CPU as busy as possible. CPU utilization may range from 0 to 100 per cent. In a real system, it should range. from 40 per cent (for a lightly loaded system) to 90 per Cent (for a heavily used system). ,

Throughput. If the CPU is busy, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput.

Turnaround time. From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission to the time of completion is called turnaround time. Turnaround time is the sum of all the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

Waiting Time, The CPU scheduling algorithm does not really affect the amount of time during which a process executes or does I/O. The algorithm affects only the amount of time that a process spends waiting in the ready queue. Thus, rather than looking at turnaround time, we might simply consider the waiting time for each process.

• **Response time**. In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are, being output to the user. Thus, another measure is the time from the submission of a request until the first response 184 produced. This measure, called response time, is the amount of timer it takes to start responding, but not the time it takes output that response. The turnaround time is generally limited by the speed of the output device.

It is desirable to maximize CPU utilization and throughput, and to.4 minimize turnaround time, waiting time, and response time In most 4 cases, we optimize the average measure. However, it may sometimes„; be desirable to optimize the minimum or maximum values, rather than the average.

First-Come First-Served Scheduling

By far the simplest CPU scheduling algorithm is the first-come first-::3 served (FCFS) algorithm. With this scheme, the process that requests .1 the CPU is allocated the CPU first. The implementation of the FCFS policy is easily managed with a First-In-First-Out

(FIFO) queue. When a process enters the ready queue, its PCB is linked onto the tail of the "". queue. When the CPU is free, it is allocated to the process at the head of the ready queue. The FCFS scheduling is simple to write and Z understand.

The FCFS scheduling algorithm is non preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it wants to release the CPU, either by terminating or by requesting I/0 The FCFS algorithm is particularly troublesome for time-sharing systems. Where it is important that each user get a share of the CPU at regular intervals. It would be disastrous to allow one process to keep the CPU for an extended period.

Shortest-Job-First Scheduling

A different approach to CPU scheduling is the Shortest-Job-First(StlF) algorithm.(The term shortest process first is not used because most people and textbooks refer to this type of scheduling discipline as shortest-job-first) This algorithm associates with each process the length of the next CPU burst. When the CPU is available, it is assigned to the process that; has the next smallest CPU burst. If two processes have the same length CPU burst, FCFS scheduling is used to break the tie,

The SJF algorithm is provably optimal, in that it gives the Minimum average waiting time for in given set of processes. Given two processes, with one having a longer execution time than the other, it can be shown that moving a short process before a long process decreases the Waiting time of the long process. Consequently, the average waiting time decreases.

The real difficulty with the SJF algorithm is knowing the length of the CPU request. For a long-term (job) scheduling in a batch system, can use the process time limit. Thus, users are motivated to estimate e process time limit accurately, since a lower value may mean faster 'response. (Too low a value will cause a "time-limit- excedded ” error and require resubmission.) SJF scheduling is used frequently in process scheduling

Although the SJF 'algorithm is optimal, that is, not other algorithm can -deliver better performance, it cannot be implemented at the level of short-term CPU scheduling. There is no way to know the length of the next CPU burst. One approach is to try to approximate SJF scheduling. We may not know the length of the next CPU burst; but we may be able .predict its value. We expect that the next CPU burst will be similar An length to the previous ones. Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the Shortest predicted CPU burst.

The SJF algorithm may be either preemptive or non-preemptive. The choice arises when a new process arrives at the ready queue while a previous process is executing. The new process may have a shorter next PU burst than what is left of the currently executing process. A preemptive SJF algorithm will preempt the currently executing process, .Whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst. Preemptive SJF scheduling is sometimes Called shortest-remaining-time-first scheduling.

Priority Scheduling. The SJF algorithm is a special case of the general priority scheduling algorithm. A priority is associated with each process, the CPU is allocated to the process with the highest priority. Equal- pority processes are scheduled in FCFS order.

An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst (t) p = l/t. The larger the CPU burst, the lower the priority and' vice versa.

Note that we dismiss scheduling in terms of high and 'low priority. Priorities are generally some fixed range of numbers, such. as 0 to 7, or .';0:;to 4095. However, there is no general agreement on whether 0 is the ':highest or lowest priority. Some systems use low numbers to represent ..1low priority, others use low numbers for high priority. This difference an lead to confusion. In the text, we assume that low numbers represent high priority.

Priority scheduling can be either preemptive or non-preemptive. When a process arrives at the ready" queue , its priority is compared with the priority of the currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newt arrived process is higher than the priority of the currently running, process. A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue. A major problem with priority scheduling algorithms is indefinite blocking or starvation;; A process that is ready to run but lacking the CPU can be considered blocked, waiting for the CPU. A priority scheduling algorithm can leave some low-priority processes waiting indefinitely for the CPU. In, al heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU. Generally, one of two things will happen. Either the process will eventually be run (At 2 A.M. Sunday, when the system is finally lightly;; loaded) or the computer system will eventually crash and lose all unfinished low-priority processes. A solution to the problem of indefinite blockage of low-priority processes is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example, if priorities range from 0 (high) to 127 (low), we could decrement a waiting:1 process's priority by 1 every minute. Eventually, even a process an initial priority of 127 would have the highest priority in the system and would be executed. In fact, it would take no more than 2 hours and 7 minutes for a priority 127 process to age to a priority 0 process.'

Round-Robin Scheduling

The round-robin(RR) scheduling algorithm is designed especially for time-sharing systems. A small unit of time, called a time quantum or time-slice is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. To implement RR scheduling, we keep the ready queue as a First-In, First-Out (FIFO) queue o processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process, sets a timer to interrupt after 1 time quantum, and dispatches the process. One of two things will then happen. The process may have a CPU burst 1 of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then' proceed to the next process in the ready queue. Otherwise, it the CPU burst of the currently. running process is greater than 1 time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process from the ready queue.

In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row. If a process's CPU burst exceeds 1 time quantum, that process os preempted and is put back in the ready vie 'The RR scheduling algorithm is inherently preemptive.

The performance of the RR algorithm depends heavily on the size of the quantum. At one extreme, if the time quantum is very large infinity); the RR policy is the same as the FCFS policy. If the time 4,0.tt.im, is very small, (say 10 milliseconds), the RR approach is called it-h6.esicI. sharing, and appears (in theory) to the users as though each processes has its own processor running at 1/n the speed of the real processor.

For operating systems, we need to consider the effect of context switching on the performance of RR scheduling. Let use assume that we have only recess of 10 time units. If the quantum is 12 time units, the process ,finishes in less than 1 time quantum, with no overhead. If the quantum time units, however, the process will require 2 quanta, resulting in context switch. If the time quantum is 1 time unit, then 9 context 'matches will occur, slowing the execution of the process accordingly. 's, we want the time quantum to be large with respect to the context twitch time. If the context switch time is approximately 5 per cent of time quantum, then about 5 per cent of the CPU time will be spent context switch.

summary of CPU Scheduling Implementations

FCFS	inherently non-preemptive
SJF	preemptive or non-preemptive
Priority	preemptive or non-preemptive
Round-Robin	inherently preemptive

Multilevel Queue Scheduling

Another class of scheduling algorithms has been created for situations 'which classes of processes are easily classified into different groups. Or example, a common division is made between foreground (interactive) recesses and background (batch) processes. These two types of processes have quite different response-time requirements, and so might have different scheduling needs, In addition, foreground processes may have priority (externally defined) over background processes.

Multilevel queue scheduling algorithm partitions the ready queue into separate queues as shown below. Processes are permanently assigned do one queue, generally based on some property of the process, such as memory size or process -type. Each queue has its own scheduling algorithm . For example, separate queues might, be used for foreground d background processes: The foreground process queue might be scheduled by a RR algorithm, while the background queue is scheduled by an FCFS algorithm. In addition, there must be scheduling between the queues. This is commonly a fixed-priority preemptive scheduling For example, the foreground queue may have an absolute priority over the background queue.

Multilevel Feedback Queue Scheduling Normally, in a multilevel queue scheduling algorithm; processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. If there are separate queues for foreground and background processes, for example, processes do not move front; one queue to the other, since 'professes do not change their foreground' or background nature. This set up has the advantage of low scheduling' overhead, but is inflexible.

Multilevel feedback queue scheduling, however, allows a process to move. between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower priority queue. This scheme leaves I/O-bound an interactive processes in the higher priority queues. Similarly; a process:, that waits too long in a lower-priority queue may be Moved to a higher-priority queue. This is a form of aging that would prevent starvation,

SUMMARY

• A common representation for a discussion of process scheduling is a queueing diagram.

• A process migrates between the various scheduling queues throughout its life time. The operating system must select processes from these. queues' in some fashion. The selection process is carried out by the appropriate scheduler.

• Scheduling is a fundamental operating system function. Almost computer resources are scheduled before use The CPU is, of course; one 's of the primary computer resources.: Thus, its scheduling is central. to operating system design.

• CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated to the CPU. There are many, different CPU scheduling algorithms.. These algorithms have different: properties and may favor one class of processes over another.

REVIEW QUESTION

1. What do you mean by term scheduling.
2. 2. Describe different types of scheduling algorithms.
3. Write short wotes on:

(i)Scheduling oneues

(ii) CUP-I/O Burst cyele

(iii) Scheduling

FURTHER READING

1. Operating Systems: A practical Approach: Rajiv Chopra, S. Chand Publisher, 2010
2. Operating Systems: Principles and Design: Pabitra Pal Choudhury, PHI Learning 3,
3. Operating Systemes and Software Diagnostics: Ramesh Bangia and Balvir Singh, Laxmi Pub., 2007
4. Principles of Operating Systmes: Sri V. Ramesh, Laxmi Pub., 2010

UNIT IV

MEMORY MANAGEMENT DEADLOCKS

STRUCTURE

4.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

• discuss memory management in operating system

• grasp the memory allocation and it's kinds

• describe non-contiguous memory allocation explain nirtual memory concept

• discuss locality of reference of a process

• define the deaolocl handling and conditions for deadlock to occur

• discuss how to detect on deadlock?

- explain methods of deadlock handling
- A explain deadlock preventing method
- understand recovery from deadlock
- know the deadlock avoidance

4.1 MEMORY MANAGEMENT

One Of the major functions of Operating System is Memory Management. e.bS controls the allocation/de-allocation of physical memory. It keeps ck. of memory occupancy, loading of programs into free memory space getting the memory, occupied by a process released, when the process terminates, dynamic allocation/de-allocation of memory to executing processes etc.

Logical Address Space and Physical Address Space of a Process

When, a process is executing, the CPU would generate addresses, called ':;magical Addresses. The corresponding addresses in the physical memory, as occupied by the executing process, are called Physical Addresses. ;But, the Memory Management Unit will recognize only the physical addresses

logical address space. This process refers to the set of all logical a dresses, that can be referenced by a process.

Physical address space. This process refers to the set of physical addresses, occupied by a process in the main memory - The mapping between logical and physical addresses, is done by the OS. For this, it makes use of some registers and tables.

MEMORY ALLOCATION

memory allocation is of two kinds:

I. Contiguous Memory Allocation.

2. Non-contiguous Memory Allocation.

Contiguous Memory Allocation

In contiguous memory allocation, a memory-resident program occupies a single contiguous block of physical memory. The memory is partitioned 'into blocks of different sizes, for accommodating the programs. The ,partitioning is of two kinds:

(a) Fixed Partitioaing . The memory is divided into a fixed number of partitions, of different sizes, which may suit the range of usually occurring program-sizes. Each partition can accommodate exactly one process. Thus, the degree of multi-programming is fixed. Whenever, a program needs to be loaded, a partition, big enough to accommodate the program, is allocated. Since, a program may not exactly fit the allocated partition, some space may be left unoccupied, after loading the program. This space is wasted and it is' termed as Internal Fragmentation.

The memory management is implemented using ,a table, call Partition Description Table (PDT). This table indicates tit base and size of each partition, along with its status (whethe F: Free Or A: Allocated).

Example:

Partition Description Table. (PDT)

Partition Id	Partition Base	Partition Status	Partition Status
0	0K	50K	A
1	50 K	50K	F
2	100 K	50K	A
3	150 K	100K	A
4	250 K	250K	F

Physical Memory Space

(0-50 K)

Operating System (0-45 K)

(Internal Fragmentation: 05 K)

(50-100 K)

Free

(100-150 K)

Process B (100-140 K)

(Internal Fragmentation : 10 K)

(150-250 K)

Process 'A' (150-230 K).

(Internal Fragmentation: 20 K)

(250-500 K)

Free

Advantages .

1.. The implementation is very simple.

. 2. The processing overheads are low.

Disadvantages

1. The degree of multiprogramming is fixed, since the "number of partitions is fixed.

2. Suffers from internal fragmentation.

(b) Variable Partitioning.

This scheme is free of the limitations encountered in the case of fixed-partitioning

Its functions are as follows:

Initially, the entire available memory is treated as a single partition.

The incoming programs, requesting memory allocation, are queued up.

A waiting program is loaded, only when a free partition, big enough to fit the program, is available. When a program is loaded, it is allocated a space, exactly equal to, its size. The balance unoccupied space, in the allocated partition, is treated as 'another free partition.

When a process terminates, it releases the partition occupied. If the released partition is contiguous to another free partition, then both the free partitions are clubbed together into a single free partition.

When a free partition is too small to accommodate any program, it is called External Fragmentation. This memory' is lost.

There may exist more than one fragment, lost due to External-Fragmentation. These fragments can be joined together by using Compaction. The space retrieved by compaction may form a partition, big enough to accommodate some more waiting programs.

• The memory management is implemented by using of a Partition Description Table (PDT), with the following information:

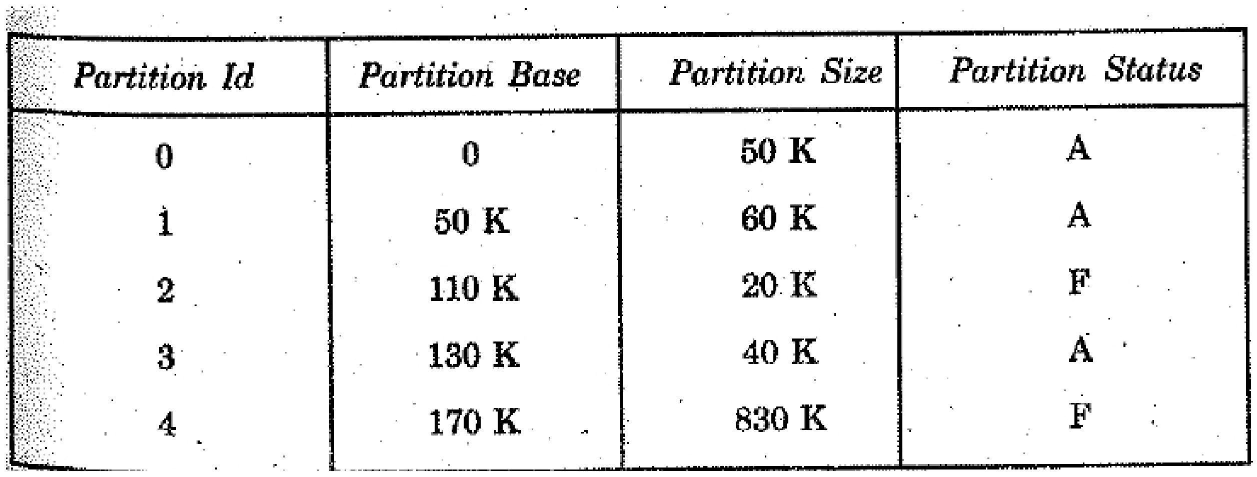
(a) Partition Id.

(b) Partition base.

(c) Partition size.

(d) Partition status (Whether F: Free or A: Allocated).

Example:



Partition Id	*Partition Base*	*Partition Size*	*Partition Status*
0	0	50 K	A
1	50 K	60 K	A
2	110 K	20 K	F
3	130 K	40 K	A
4	170 K	830 K	F

Physical Memory Space

(0-50 K)

Operating System (0-50 K)

(50-110 K)

Process 'A' (50-110 K)

(110-430 K)

Free (External Fragmentation: 20 K)

(150-250 K)

Process 'A' (150-230 K) (250-500 K).

Free

External Fragmentation and Compaction

External Fragmentation refers to the large number of small chunks free memory that may be scattered all over the physical memory al individually each of the chunks may not be big enough to accommodate` even a small program. However, if joined together, the total free spa thus resulted, may be big enough to accommodate some more programs that may be waiting for loading. Compaction is a technique, by whit the resident programs are re-located in such a way that the small chunks of free are made contiguous to each other and clubbed together• into single free partition, that may be big enough to accommodate sort more programs. But the Compaction has very high processing overheads

Different Strategies for Partition Allocation

There are three strategies:

(a) First Fit

This refers to the allocation of the first encountered partition that may be big enough to accommodate the program being loaded. This algorithm works both for fixed-partition and variable-partition allocation scheme

(b) **Advantage**. Search time is small.

Disadvantage The memory loss, on account of fragmentation, is likely to be high.

(c) Best Fit

It refers to the allocation of the smallest available free partition that may be big enough to accommodate the program being loaded. This' algorithm also works both for fixed-partition and variable-partition allocation schemes

Advantage.

The memory loss, on account of fragmentation, will be lesser:than in the case of First Fit.

Disadvantage. Search time will be. larger, as compared to First Fit.

(c) Worst Fit

It refers to the allocation of the largest partition out of the ones which are available, which may be bigger enough to accommodate the program spirit behind this scheme is that the balance space, left in the died partition, after loading the program, may be big enough to a :.another small program; and in that eventuality, memory loss due to fragmentation may be lower than First Fit. Search time in this scheme Uld be of the same order as in the case of Best Fit.

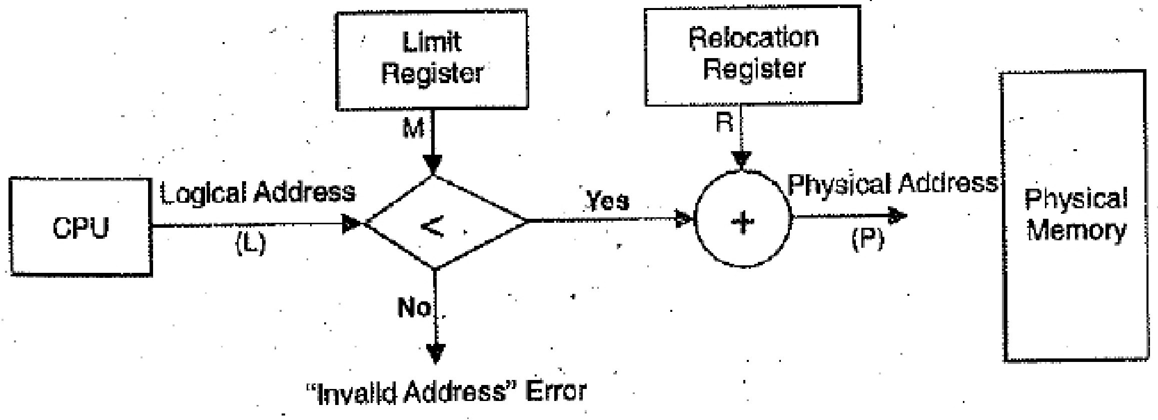
Advantage. It is expected to provide a smaller memory loss on account :fragmentation.

Disadvantages. .1. Search time is larger than First Fit. It is same as in the case of Best Fit.

2. This algorithm is designed only for variable-partition allocation. By mistake, if used along with fixed-partition, the results will be suicidal.

Memory Protection in Contiguous Memory Allocation

The limit Register is initialized to the size M of the currently executing program and the Relocation Register is initialized to the base address ;the executing program. When the program generates a logical address, the address is compared with the contents of the Limit Register "). If the logical address <= M, then it is a valid address; else it is an Valid address error'. If the logical address is valid, then the corresponding physical address is computed by adding the contents of Relocation Register. (B) to the logical address. In case of "Invalid Address `Error", the process is terminated.



(Logical address L < Limit register value M) physical address P = Logical address L + Relocation value R; else raise (valid address Exception".

Non-Contiguous Memory Allocation

It offers the following advantages over, contiguous memory allocation

(a) Permits sharing of code and data amongst processes.

(b) There is no external fragmentation of physical memory

(c) Supports virtual memory. concept.

However, non-contiguous memory allocation involves a complex, implementation and involves ,additional costs in terms of memory an processing.

Non-contiguous memory allocation can be implemented by the cone

of:

(a) paging;

(b) segmentation;

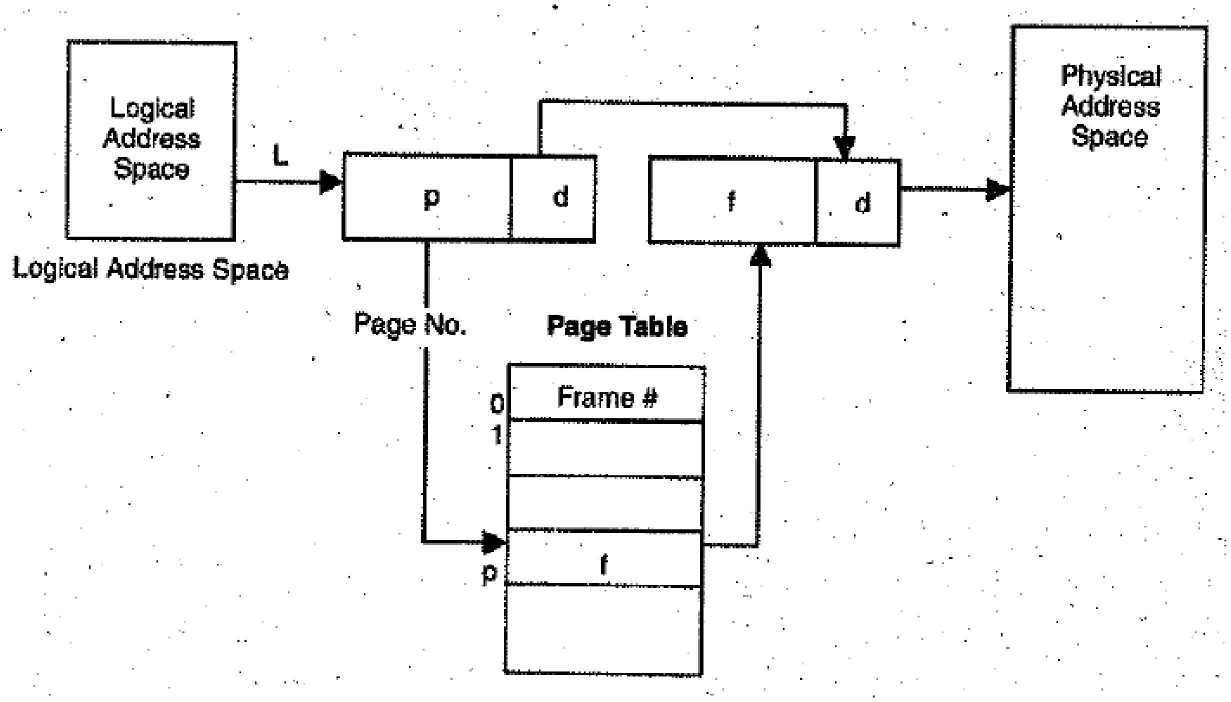
(c) segmentation with paging.

(A) Paging

It permits physical address space of a process to be non-contiguous. Th logical address space of a process is divided into blocks of fixed size called Pages. Also, the physical memory is divided into blocks of fixed; size called Flumes. In a system, the page and frame will be of same size: The size is of the order of 512 bytes to a few MB. Whenever, a proces44 is to be executed, its pages are moved from secondary storage (a fas0 disk) to the available frames in physical memory. The information abon0 frame number, in which a, page is resident, is entered in page table. Th page table is indexed by page number.

Implementation of Paging

The system makes use of paging table, to implement paging. When al process is to be loaded, its pages are moved to free frames in the physical memory.



the information about frame number, where a page is stored, is entered in the .page table. During the process execution, CPU generates a logical .address, that comprises of page number (p) and offset within the page (a The page number p is used to index into the page table and fetch .corresponding frame number (f). The physical address is obtained by combining the frame number (f with the offset (d).

Given a Logical Address L, How to Compute the corresponding Physical Address?

For a 'm' bit processor, the logical address will be m bits long.

Let the page size be 2" bytes.

Then, the lower order n bits of a logical' address I will represent page offset (d) and the higher order m-n bits will represent the page number (p).

Then, page number p = L/

And page offset d = L% 2

Let f be the frame number that holds the page 2nreferenced by logical address I. Then f can be obtained by indexing into page-table, by using page number

p as index i.e., f = page-table A;

Corresponding physical address = f * d

This physical address is fed to the Memory Management Unit (MMU) to access the memory location, referenced by logical address 1. The max, size of logical address space of a process can be r bytes i.e., up to 2" pages.

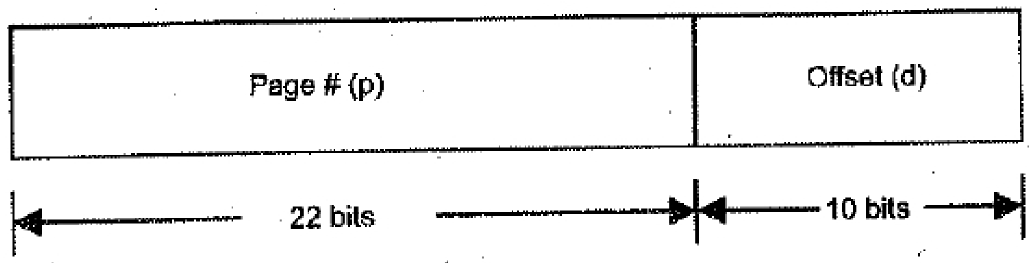
Internal Fragmentation in Paging Since, a program size may not be an exact multiple of the page-size, some space would remain unoccupied in the last page of a process. This results in internal fragmentation. The average memory loss, due to internal fragmentation, would be of the order of half page per process. So, larger -the page size, larger would be the loss of memory, by internal fragmentation. Suppose, a system supports a page-size of P bytes, Then, a program of size M bytes, will have an internal fragmentation = P — (M %'P) bytes.

Limitations of Basic Paging Scheme (Discussed Above) vis-a-vis Contiguous Memory Allocation

1. The effective memory access time increases, since for accessing of an operand, first its frame number has to be accessed. Since, the page table resides in the RAM itself, the effective access time to get an operand will be twice the RAM access time. So, if RAM access time is 100 ns, the effective access time would be 200 n,s.

2. The page table occupies a significant amount of memory.

Example:



For a 32 bit processor, with a page size of 1024 bytes,

Size of logical address = n = 32

Page size = 2m = 2i0 1024 bytes

Number of bits to represent page offset = m = 10

Number of bits to represent page number = n m = 22

• The low order 10 bits of a logical address will represent. page-offset and the higher order 22 bits will represent page number.

• Max size of logical address space = 232 bytes = 4 G Bytes.

• Max Number of pages in logical address space = 222 = 4 Million.

• So, the max length of page table of a process = 4 M entries, each entry 'being 4 bytes. So, a page table would occupy 16 M bytes in RAM.

• Suppose, the physical memory size is 256 MB, the page table of a process would occupy 1116th of the whole memory. (a substantial amount of memory, just to accommodate a process page table)

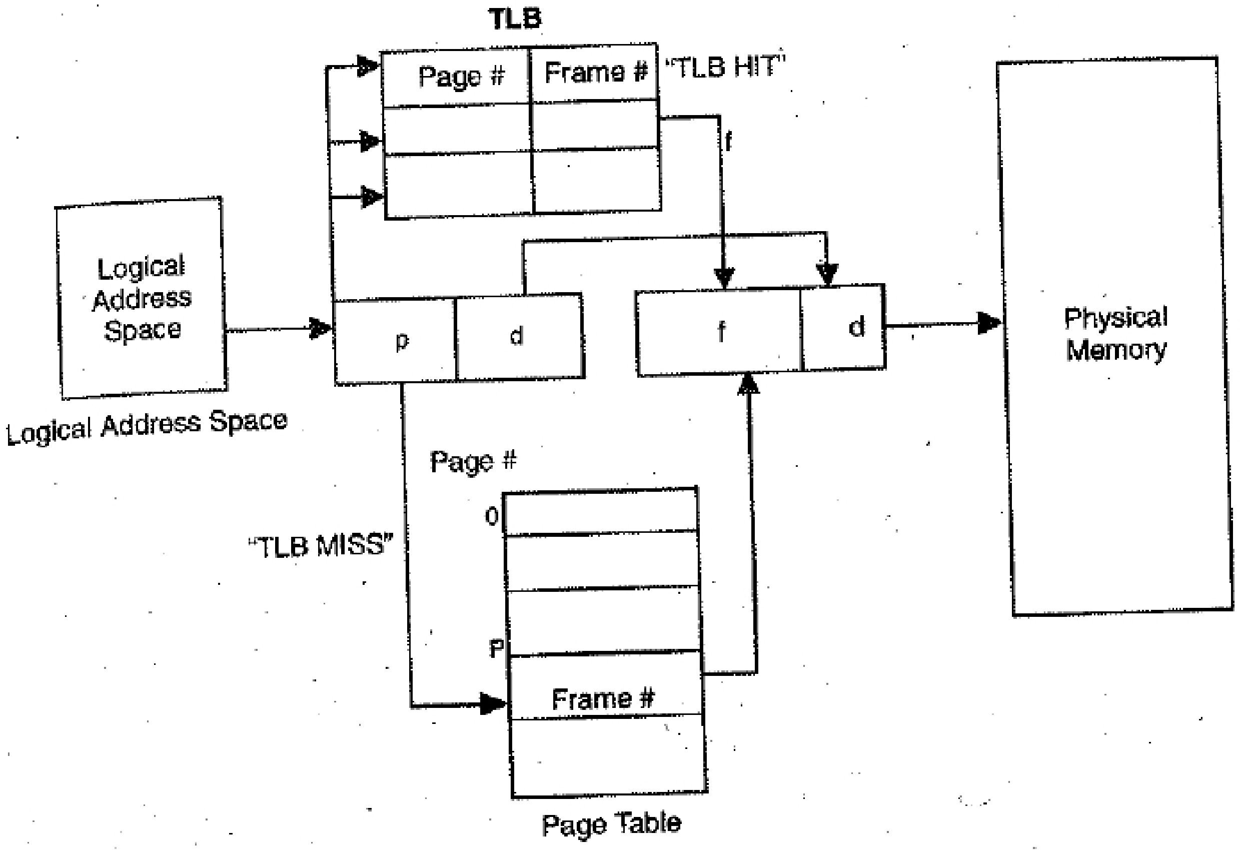
• The page table is per process. So, if 5 processes are memory resident simultaneously, the page tables would occupy 80 MB of RAM.

4. Since the page table is per process, the page tables would also need to 'be switched during context switching.

As, evident, the Basic Paging Scheme would need some enhancements to reduce its memory overheads.. These are discussed in the preceding paragraphs.

How to Reduce the Effective Memory-access-time in a. Paged System? Use of Translation Look-aside Buffer (TLB)

TLB is page-table cache, which is implemented in a fast associative memory. The associative memory is distinguished by its ability to search for a key, concurrently in all entries in a table. This property' makes the associative memory much faster than conventional RAM. But, it is much costlier also. Due to higher cost, it may not be cost-effective to have the entire page table in. TLB, but a subset of the page-table, that may• be currently active, can be moved to TLB. It is implemented as -follows:



Each entry of TLB would contain Page # of a page and the Frame ;where the page is stored in RAM

. • Whenever, a logical address is generated, the page number p of the logical address is fed as a key to the TLB.

• The key is searched in parallel in all the entries of TLB.

• If a match found for the page number p, it is termed as TLB Hit. The entry, with the matching' number contains -the Frame # f, -where the page is stored. The frame number is used to access the desired physical location in the RAM.

• If match not found for the page number, it indicates TLB Miss. Then, the frame number is accessed from the page table. Also, the page table entry is moved to TLB, so, that for further references to that page, its frame number can be accessed' from the TLB itself. If TLB is full, then some replacement algorithm can be. used to replace one of the existing entries in the TLB. The algorithm could be "replacement of the least recently used (LRU) entry".

This would improve the effective memory access time as illustrated by the following example:

Let TLB Hit Ratio = 0.9 (This is the probability that an intended frame number would be found in the TLB itself, with no need to look into the Page Table. Larger the TLB, higher would be the TLB Hit Ratio.)

Let RAM access time t , = 20 ns

And TLB access time T 100 ns

Effective memory. access (with TLB) =. H* (T + t) + (1 — H) (2T + t) = 0.9* 120 + 0.1* 220

= 108. + 22

= 130 ns

Effective memory access (without TLB)= 2 T = 200 ns

Reduction in effective access time = (200 — 130)* 100/200 = 35%

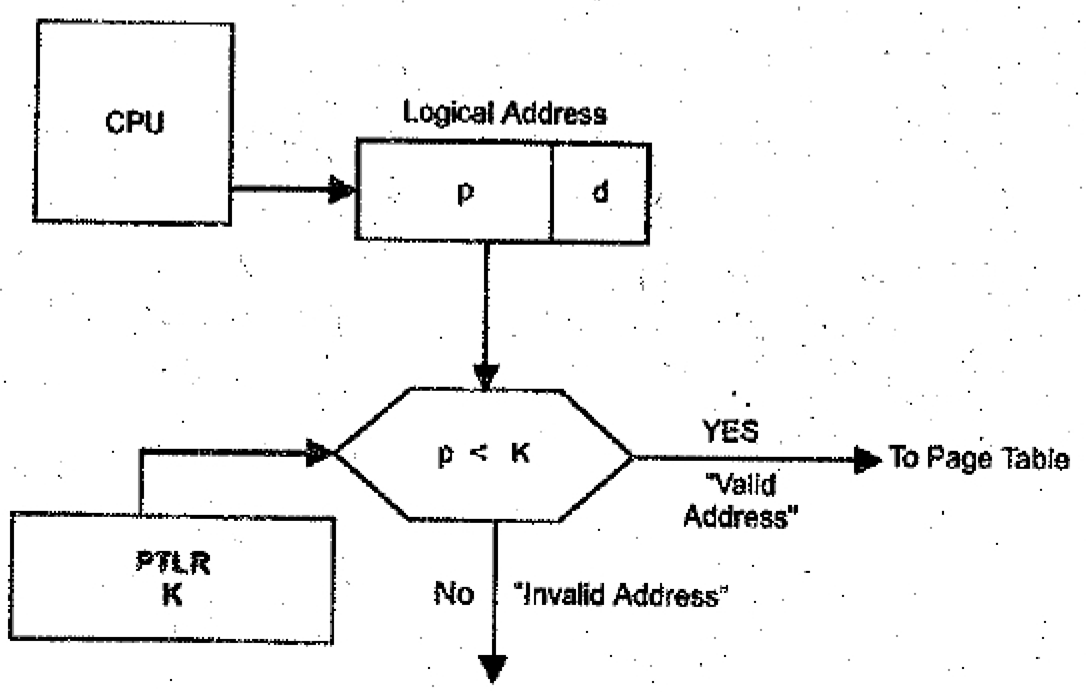
How to Reduce the MemorY-Overhead of 'a Paged System?

1. By increasing the page-size. This will correspondingly reduce the max number of pages in the logical address space, thus reducing the size of page table. But, increasing the page size would increase the loss of memory, due to internal fragmentation.

2. By creating a page table for the exact program size and using a Page 712ble Length Register (PTLR).

When a program is loaded into RAM for execution, its page Table is created to have, only as many entries, as the size of. the program in pages. The size of the program is stored in a register called Page Table Length Register. (PTLR). When a logical address is generated, its page number p is compared with the Page Table Length (say K) stored in PTLR. If p < K it is a valid address; else it is invalid. If the page number is valid, frame number is accessed, by indexing the page table with p as index.

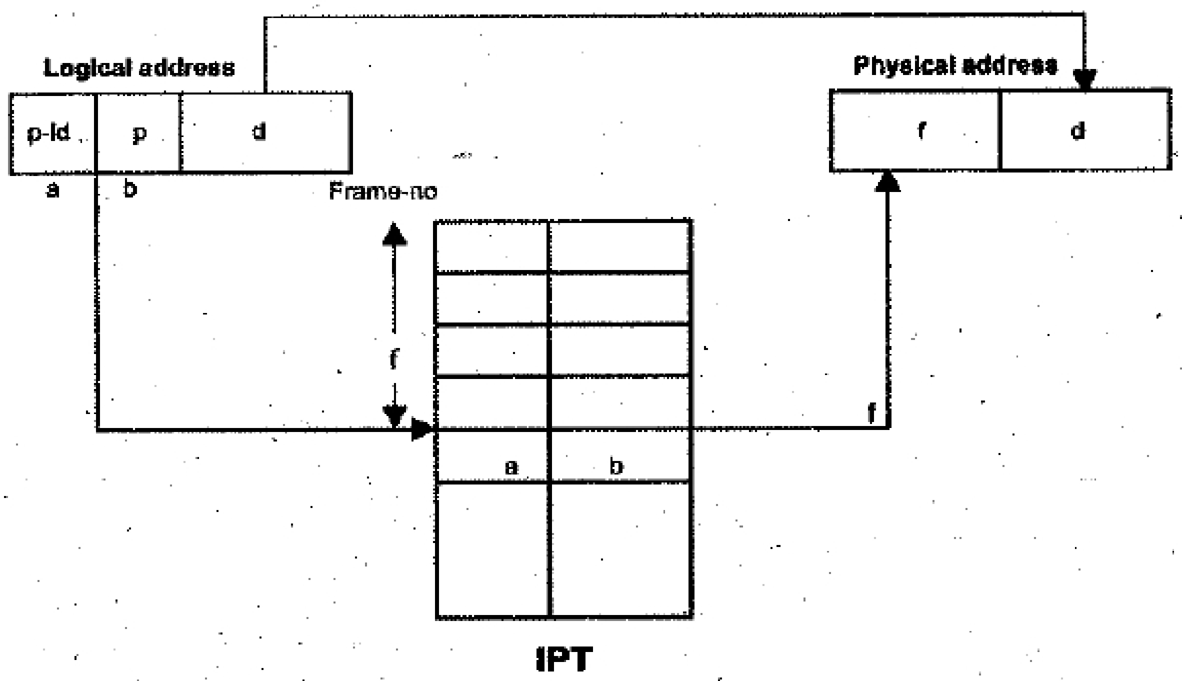
When a program is loaded for execution, PTLR is initialized to K, where K = (Program-Size/Page-Size) + 1



3.By using of inverted page table. The table is indexable by.frame number, instead of page number. Each entry, in the table, contains page number, corresponding to the indexed frame number.

Inverted Page Table

The size of the inverted page table size is related to the size of the physical memory, not the size of the logical address space. Since, each • me in the physical memory can have at most one entry in the inverted page table, the table can act as a 'system wide table, if each entry in the able contains process-id (of the process, to which the page belongs) with Page #. Since, information about process-id of each page is available in the IPT, a single IPT is sufficient for the entire system. IPT is not process-specific and it does not need switching during .context switching. A logical address generated by CPU contains process-id (p-id), page number ' (p). and page-offset (d). A search is carried out the IPT to find a match for the process-id and the Page #. The offset of the matching slot in the IPT, gives the Frame # f, where the desired page is residing. The Frame # f, combined with the offset d, gives the intended physical address.



Advantages of IPT Over Conventional Page Table

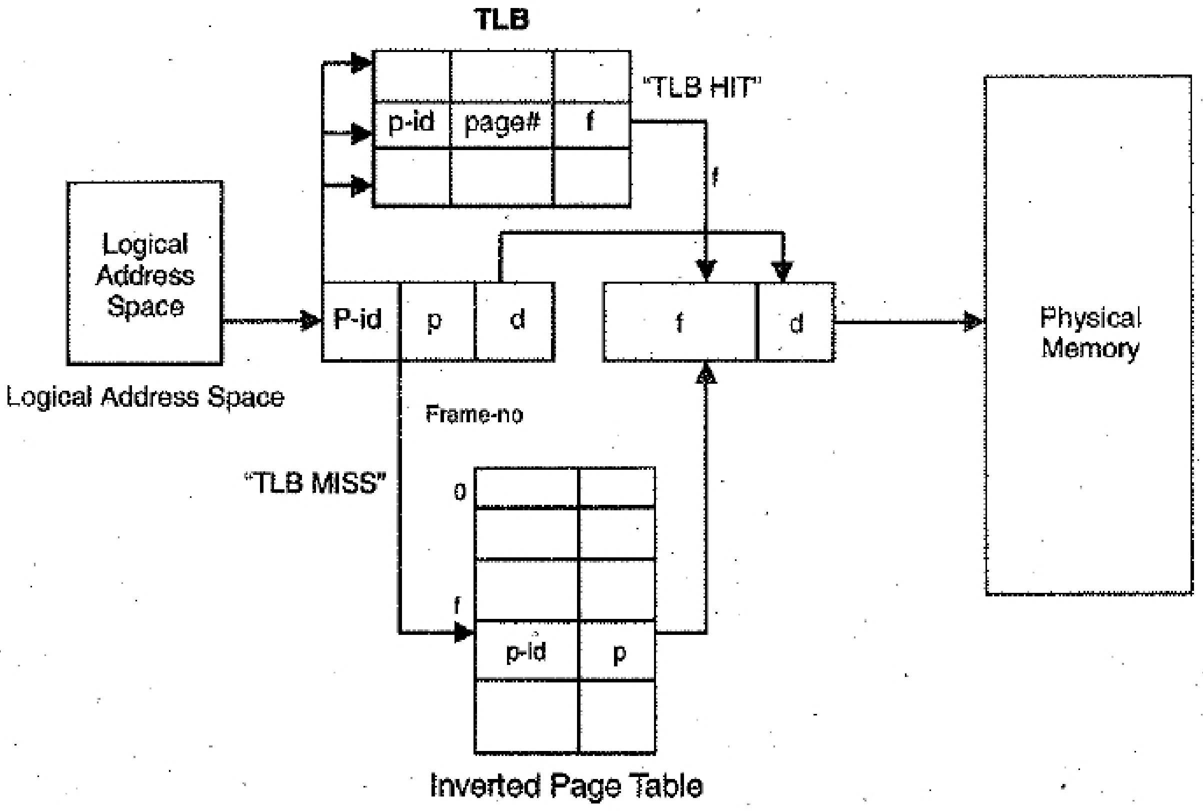
There is only one page table. for all processes, whereas conventional page table is per process. Its size will be related to the size of physical memory available.

Limitations of IPT

The search time to match an entry in the IPT is very large. If the search time in Page Table is considered to be one unit, then the average search time in case of IPT would be K/2 units.

How to Reduce Average Search Time of IPT?

The average search time of IPT can be reduced by combining the IPT with a TLB. Each entry in the TLB would comprise Page #, process-id and its corresponding frame number. The associative memory would permit matching of a (page number, process-id) pair, simultaneously with all the entries in the TLB. The desired frame number can then be obtained from the matching entry.



Some other Implementations of Page Table

Multi-level Paging

The page table is split into multiple levels. For example, in a two-level page table, a logical address would comprise of the following fields:

(a) Page Number p1, for indexing into the Outer Page Table. Each entry in the outer page table contains the base address of an inner page table: So, indexing the outer page table with p1, will select the intended inner page table. Base address of inner page table = Outer-page-table [pl.];

(b) Page number p2, for indexing into the inner page table, selected by page number pl. Each entry of a inner page table would contain a frame number that contains the intended page. So, indexing the selected inner page table with p2, makes available the base address of the frame that contains the intended page.

Frame number f = Inner-page-table [p2];

(c) Offset or displacement d, that is used to index into the selected I frame, to obtain the desired operand.

Physical address page-size + d;

The combined length of all the inner page tables would be same as the length of a single page table in the case of single-level-paging. 0

Example:

Suppose size of logical address = n = 32

Page size = 2rn = 21?

= 1024 bytes

Number of bits to represent page offset = m = 10

Number of bits to represent page number = m = 22

Length of page table (for single-level-paging) = 222 = 4 M entries

Suppose, for the two level paging, of the total 22 bits representing page number, 12 bits are used to represent the outer page number and 10 bits are used to represent the inner page number.

Then, Length of outer page table = 212 = 4 K entries

Number of outer page tables = 1

Length of each inner page table =210 = 1 K entries

Number of inner page tables = 212 = 4 K

Combined length of all inner page tables = 1 K X 4 K = 4 M entries

This is exactly equal to the page table length of single-level paging.

But, in the two level paging, all inner page tables need not be concurrently memory-resident. Suppose, the executing program has a logical addiess space of 32 MB. This would comprise 32 K pages. Each inner page table can address 1 K pages. So, only 32 inner page tables (out of the total 4 K inner page tables) would need to be memory-resident. Thus, it reduces the memory overhead of page table.

Advantage of Multi-level Paging

All the inner page tables would not be required to be memory resident simultaneously. Depending upon the size of executing program, only a small fraction of the set of inner page tables would need to be memory resident, thus reducing the memory overhead of page table. •

Disadvantage of Multi-level Paging

To access an operand, multi-level paging needs some extra memory accesses. For example, in the case of two-level paging, an additional memory access is required, that is, to get the base address of inner page table.

Hashed Page Table

• A page table is created of length M.

• Whenever, logical address is generated, a hashing function is applied to the page number p, to generate an index value i. i = p % M;

• The index value i is used to index into the page table. Each entry in the page table is a pointer to a link list. A node in the link list will provide mapping between page number p and the corresponding frame number f. Each node will contain the following information:

(a) Page-number (say p1), such that (p1 % M

(b) Frame-number fl, where the page number p1 resides.

(c) Pointer to the next node in the list.

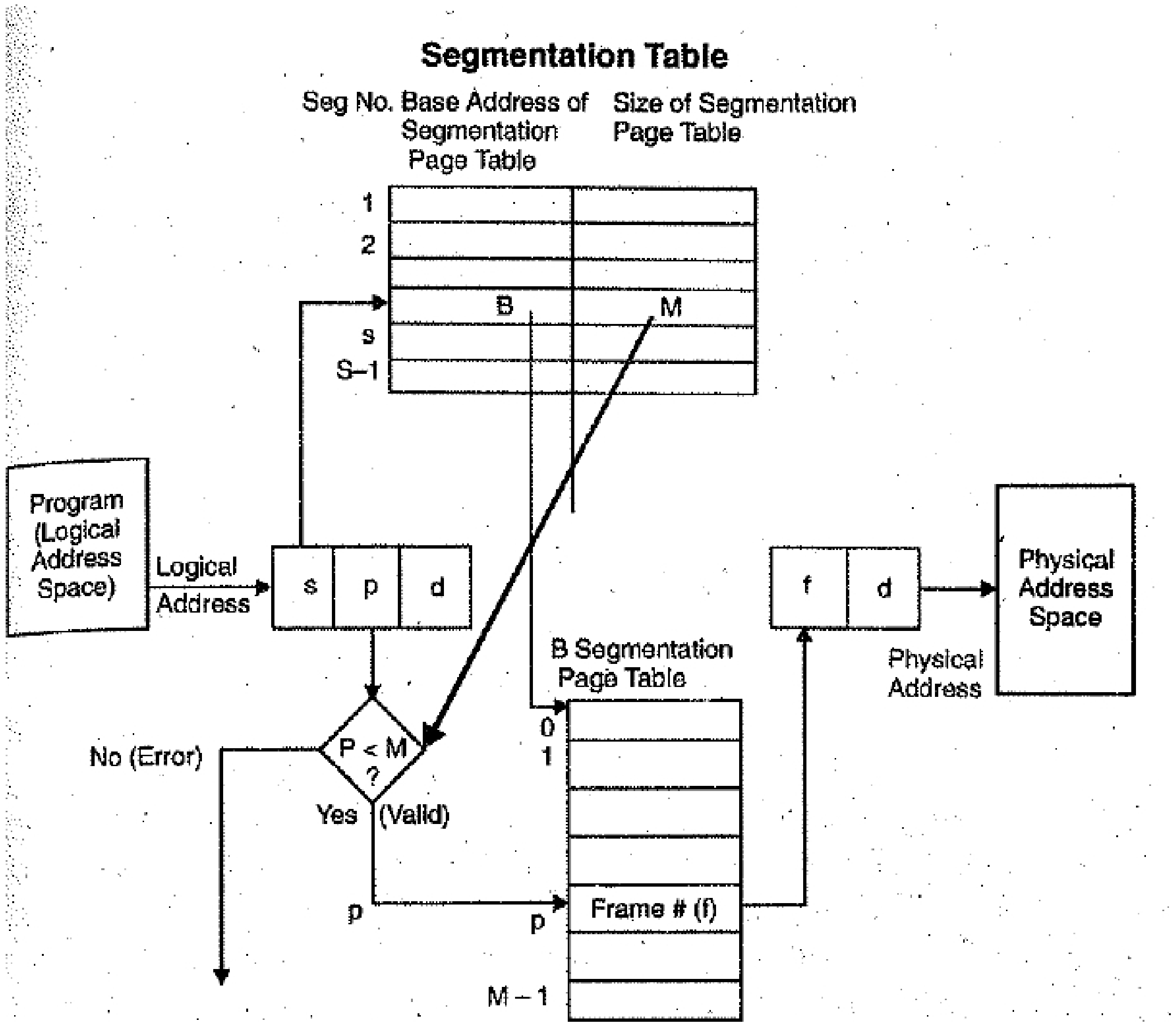
So, the link list, accessed through the index value i, will be traversed, till a match found for page number p and the corresponding frame number f is obtained.

Using Hashed table, we can have a page table of any length. Translation from logical address to physical address will involve indexing into the page table and then traversing the link list looking for a match of the page number. The link list, being sequential, is time-consuming. But, this will reduce the memory overhead of page table, since link list nodes will be created only for those pages, that will be memory resident.

(B) 'Segmentation

In segmentation, the physical memory is divided in segments of varying sizes. Each segment is assigned a unique segment number. The memory management is done through a segment table, which is indexed by segment number. For each segment, it has an entry that provides (a) base address of the Segment and (b) size of the segment. The _logical address

Segmentation table



contains segment number 's' and offset within the segment V'. Using the segment number, the system obtains base address of the segment. Then makes a check to determine whether offset is within the segment size r not. If yes then the offset d is valid and physical address is computed y adding the offset to the base address; else it is error.

C) Segmentation with Piging

In segmentation with paging, the logical space is divided into a number of segments of varying sizes and each segment is divided into a number of pages, each of a fixed size. The memory management is done through a segmentation table. Each segment has an entry in the table. An entry :contains base address of the segment page table and size of the segment page table. The logical address contains segment # `s', page number ''p' in a segment and offset `d' in the page. The segment number 's' is to access the segment entry in the segmentation table. The page table se address 'B' in the entry is used to access the segment page table. So, each segment will have a separate page table. The page number S used to access the frame # r in the page table, provide p < M (i.e., of the segment page table); else it is invalid page number. The 'frame number 1" is combined with the offset `d' to compute the physical address.

4.3 VIRTUAL MEMORY CONCEPT

Virtual memory is a technique that permits execution of processea. with their code only partially loaded into the physical memory. Thi& technique takes advantage of the fact that a program tends to hay locality of reference and a process code is never required in its entirety at the same time.

Advantages

1. Programs are not constrained by physical memory size.

2. Degree of multiprogramming can be varied over a large range

Implementation (Concept of Demand Paging)

Virtual memory technique is implemented by Demand Paging. When : process is swapped-in, the pager loads into the physical memory a sel of its pages, which may be predicted to be initially needed by the procesif The secondary memory (a high-speed disk) holds the remaining pageo of the process. Subsequently, when the process needs some pages that may not be memory resident, the pager will loaded those pages into thk, RAM. This technique is called Demand paging. For its implementation the OS needs some hardware support. The page-table is implemented to include a Valid-invalid Bit for each page entry. When a process swapped-in, this bit is set to invalid; for all the entries in the pagq table. When a page is loaded into RAM, its frame number is entered and page validity bit is set to Valid. Thus, if the bit is set to Valid, it indicates that the page is legally from the logical address space of 0k; process and is in currently in the RAM.. If it is set to invalid, it indicatef that either the page does not belong to the logical address space of the process or it is still not loaded into RAM. Whenever, a logical address (virtual address) is generated, the syste

operates as follows:

(a) Looks into the page table. If the page validity bit is set to it reads the, corresponding frame number, adds page offski and determines physiCal address.

(b) If the page table indicates page-invalid, then aystem reads the> page into a free frame. If no free frame is available, it usei:a; replacement algorithm to replace an occupied frame.

The implementation will need following system tables:

(a) Page Map Table (PMT). This table is per-process table.' indicates whether a page is memory resident or not If vali4, invalid bit is set to invalid (say 0), it indicates that the pa is not yet loaded into memory. If the.bit is set to valid (say it indicates that the page is memory resident. For memory'.

resident pages, it provides frame number of the frames, where the page resides in the memory

(b) Memory Map Table or.. Free Frames' List. This provides information about the availability of free frames, that can be allocated to accommodate new pages. This table is per-system.

(c) File Map Table (FMT). This table is per-process. It provides the disk 'addresses, where the process pages are stored on the secondary storage.

Global and Local Page Replacement

Whenever, a process experiences a page-fault, if no free frame are Available, then the OS gets a currently occupied frame released and allocates the released frame to the page-faulting process, for ":accommodating the new page. If the victim page belongi to the page-failting process itself, it is called local page replacement; else it is called global page replacement. Global page replacement does not focus an the performance of individual processes. It, rather, focuses on the .system performance.

Page Replacement Algorithms

1. FIFO

2. Least Recently Used (LRU)

3. Optimal Page Replacement

4. Clock Page Replacement ,

5. Least Frequently Used (LFU)

6. Most Frequently Used (MFU)

7. Page Buffering- Algorithm .

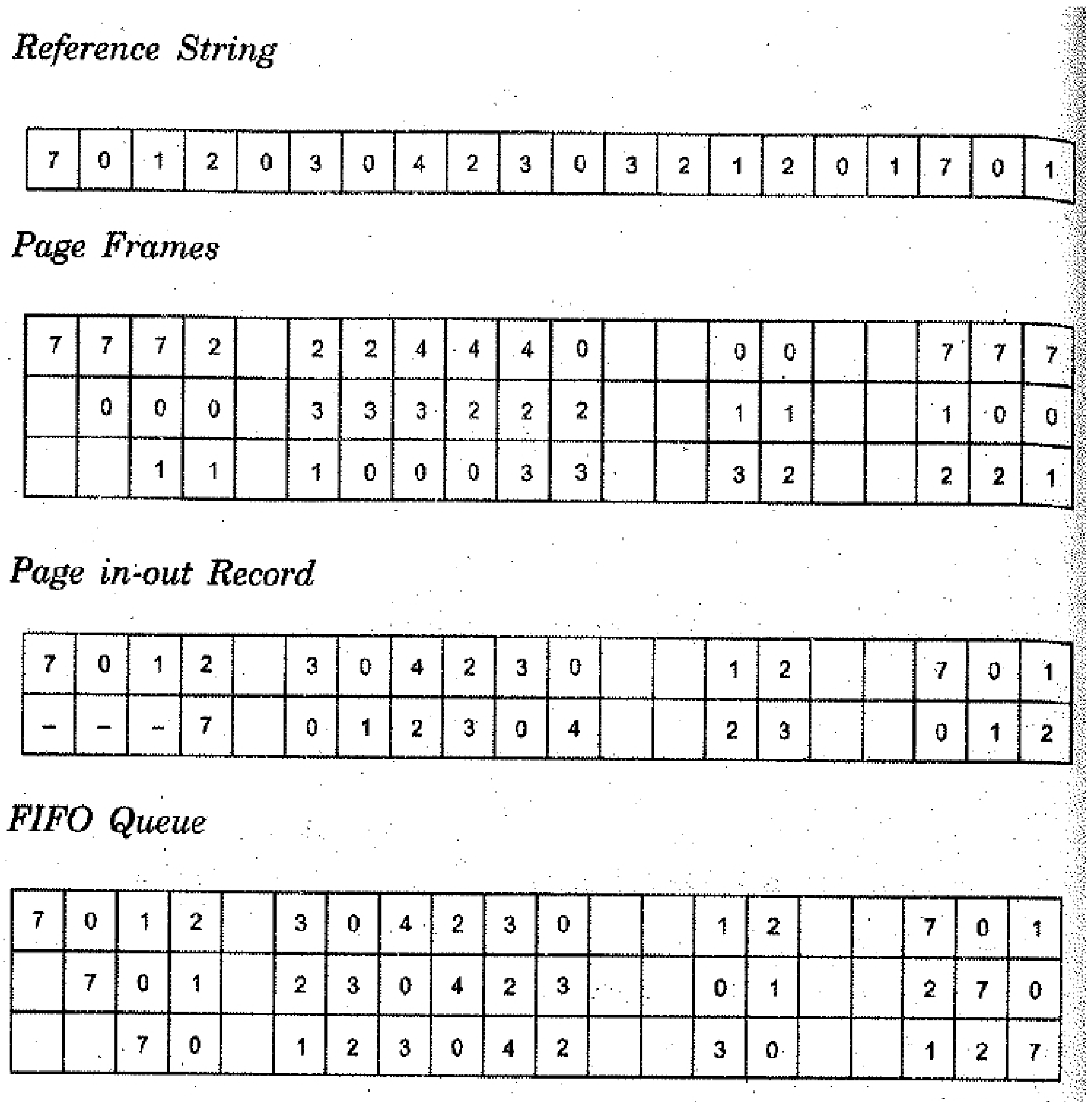
FIFO Page Replacement Algorithm

It replaces the page that has been in the memory longest.'One possible implementation is a FIFO queue of existing pages in the memory. The' oldest page will be at the head of the queue. Whenever, a page-fault occurs, the page at the top of the queue is made victim and the new page is put at the tail of the queue.

Reference String

A reference string refers to the sequence of page numbers referenced by a program during its execution. Page number =Quotient (Logical Address/page-size) Therefore, if logical address= 0745 and page size•= 100 bytes, then page number = Quotient (0745/100) = 7 :

,Assume a reference string : 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 and a set "of 3 frames available for allocation.



Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
	0	0	0		3	3	3	2	2	2			1	1			1	0	0
		1	1		1	0	0	0	3	3			3	2			2	2	1

Page in-out Record

7	0	1	2		3	0	4	2	3	0			1	2			7	0	1
–	–	–	7		0	1	2	3	0	4			2	3			0	1	2

FIFO Queue

7	0	1	2		3	0	4	2	3	0			1	2			7	0	1
	7	0	1		2	3	0	4	2	3			0	1			2	7	0
		7	0		1	2	3	0	4	2			3	0			1	2	7

Number of Page Faults in FIFO := 15

Plus points of FIFO Algorithm: Implementation is fairly simple:

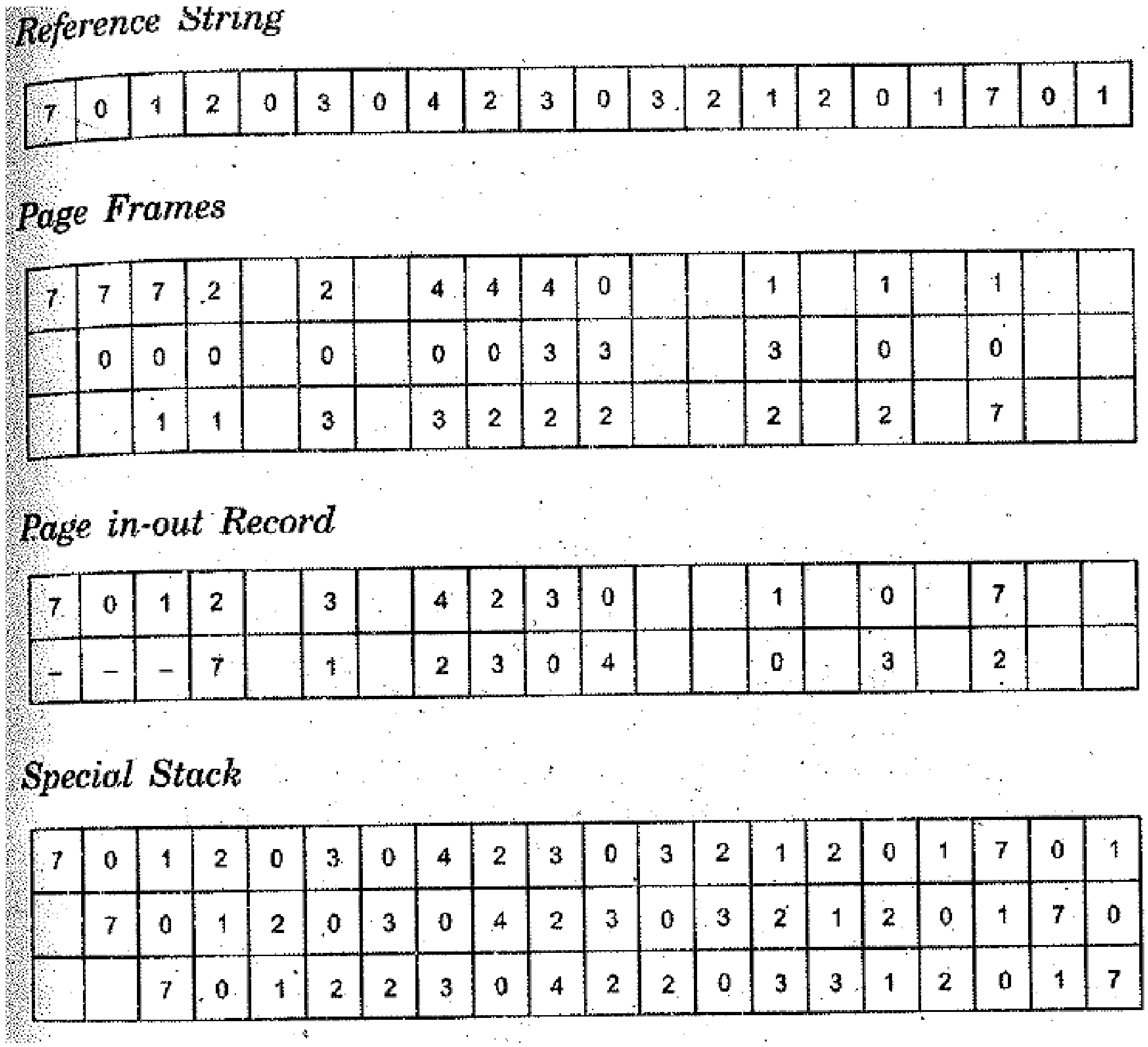
Limitations of FIFO Algorithm

(a) This algorithm does not take into account, the Current usage of the pages and may often eject some pages that may be currently active. Such pages would need to be moved-in again, in the near future.

(b) Also; if the system has global page replacement, then the program having largest number of allocated pages would have higher page fault rate, since the probability of oldest page belonging to this program, would be very high. This phenomena is called Belady's Anomaly and it defies intuition.

2. **Least. Recently. Used (LRU)**
   It replaces LRU page i.e., the page which has been used least recently:- So, while choosing a resident page for replacement, the algorithm takess into account its current usage. It is presumed that a page that has been used least recently, would be the one that would be least likely to be accessed in the near future. One implementation of this algorithm could, by using a stack. Whenever, a new page is brought in; it is placed the top of stack. Also, whenever, a resident page is accessed, it is removed from its current position and moved to the top of stack. whenever, a page is to be replaced, victim page is chosen from the w ttom of stack.

let us see the performance of this algorithm, for the above reference string.



Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

Page in-out Record

7	0	1	2		3		4	2	3	0			1		0		7		
–	–	–	7		1		2	3	0	4			0		3		2		

Special Stack

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0
		7	0	1	2	2	3	0	4	2	2	0	3	3	1	2	0	1	7

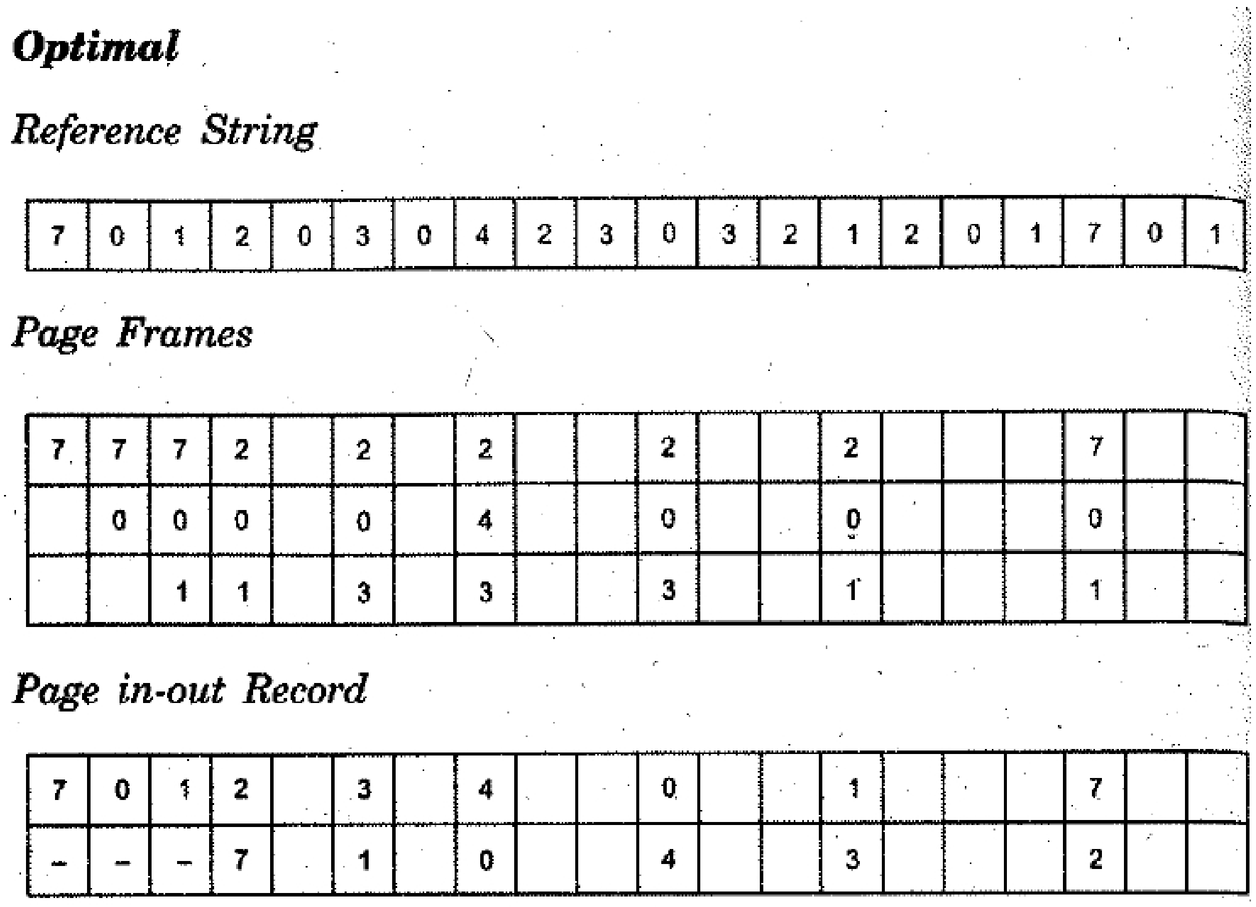
Number of Page Faults in LRU = 12

Plus points of LRU Algorithm While selecting a resident page for replacement, It~ takes into consideration the current usage of a page the algorithm is free from 13elacly'S Anomaly.

Limitations of LW Algorithm. The algorithm has a lot of processing-, overheads, which are needed to keep track of LRU page.

Optimal (OPT) Page Replacement

its ideal form, this algorithm should replace a page, which is to be referenced in the most distant future. Since, it requires knowledge of '1 the future, its ideal form is not practically realizable. The significance f this algorithm is only theoretical. It is used to compare performance ":`of a practically realizable algorithm with that of the optimal (though `I not a realizable algorithm, but optimal).



Optimal

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	7	7	2		2		2			2			2				7		
	0	0	0		0		4			0			0				0		
		1	1		3		3			3			1				1		

Page in-out Record

7	0	1	2		3		4			0			1				7		
–	–	–	7		1		0			4			3				2		

Number of page faults in Optimal = 09

Going by the number of page faults, the Optimal Algorithm appears to be the best, but it is not feasible to implement this algorithm, since the aliorithm requires knowledge of the future. Out of FIFO and LR algorithms, the LRU is better, since it takes into consideration the current usage of a page whenever a resident page is considered for replacement. The page, which has been least recently used, is chosen for replacement.. So, the algorithm is free of Belady's Anomaly.

4 Clock Algorithm

This algorithm combines the relative low overhead of FIFO algorithmt with the page-usage-consideration of the. LRU algorithm. This algorithm, is also sometimes referred to as Not Recently Used '(NRU). It replace • a resident page, which has not been accessed in the near past. It works ". as follows:

(a) The algorithm maintains a circular list of all resident page A referenced bit is associated with each page, which is se whenever, a page is reference&

(b) Either at regular intervals or whenever the number of free page-frames falls below a preset threshold, the algorithm sweeps the circular list While sweeping, it operates as follows:11

(i) It inspects the referenced bit of each page. If the referenced bit of a page is set, it implies that the page has been referenced in the near past i.e., subsequent to the last sweep. The algorithm clears the referenced bit and proceeds to look at the next. page.

(ii) If the referenced bit of a page is not set, it implies that the page has not been referenced subsequent to the last sweep. Now, if the modified bit of the page is not set then the page is declared non-resident and the frame is freed. However, if the modified bit is set then the page is scheduled for writing onto the disk.

(iii) The algorithm continues the sweep,. till the required number of frames has been freed.

(iv) Next time, it commences from the point where it had left last time (as indicated by the pointer).

Least Frequently Used (LFU)

this algorithm, a count is associated with each resident page, which is incremented by one, whenever a page is referenced. Whenever a replacement is necessary, a page with least count is replaced. The main drawback of this algorithm is that some pages may have a very high usage initially and may build a high count. Such pages, even if they have low usage subsequently, would remain memory-resident, due to. the high count. One solution to this problem could be that occasionally the count of each resident page could be shifted right by one (divide by two). Thus, the count would become "exponentially-decaying average-usage count".

. Most Frequently Used (MFU)

This algorithm replaces the page with the largest usage-count. It is based on the assumption that the pages with smaller count have been brought-in only recently and would need to be resident. This algorithm is very close to FIFO algorithm and has all the anomalies associated with FIFO.

4.4 LOCALITY OF REFERENCE OF A PROCESS

At any time, 'during its execution, a process will be accessing only a small subset of logical address space.. This subset of the logical address space is called its current locality of its reference. The current locality keeps shifting: It also keeps varying in size. Whenever, a page forming :part of the current locality is not found in the memory, it will cause a page-fault. So, when a process moves from one locality -to another, it will get the new pages into memory through page-faulting. . How to determine the Minimum number of frames, needed for accommodating the current locality of execution of a process?

Page Fault Frequency

A profess; having inadequate frames, will exhibit, a high PUge Fault Frequency. The system can specify a Critical Page Fault Frequency.

Let P (per millisecond) be the critical page fault frequency, whe P=1/T

T is the critical page-fault interval in milliseconds (i.e., the time elaps between two successive page faults).

The PFF Algorithm may be implemented as follows:

(a) Whenever, a process is having a page-fault, the OS compute the time elapsed, since its last page-fault. If the interval lower than the critical value T (it indicates a PFF higher the critical), then a new frame is allocated to the process.

(b) However, if the interval is higher than T, then a frame assign to the process, whose "referenced" bit and "written-into" bi are found to be clear, is replaced by the new page.

(c) The OS sweeps and frees those' page-frames that have no been referenced since last sweep. These frames are added to the pool of free pages. The "referenced" bits of the remaining?. resident pages are reset.

Working Set Model

This model is based on concept of current locality of reference of a, process. The working set principle states that: (

a) A program should' not be run, unless its working set is fully loaded in the memory

(b) No page of a process should be replaced, if it forms part of working set.

Ideally, the working set of a process should comprise its current local Precisely, For this, a Working-Set-Window (say 6) is set. The www.king set of a process comprises the set of pages referenced by it, during the most recent 8 references.

The Working Set of a process, at a time t is defined as :

W (t, 3) = (Set of pages i page i appears amongst r". 1,ra-s. . rt1 where rt denotes the' page, referenced at time t.

The working set window S is to be carefully chosen. If it is too small, then the working set will not encompass the current locality fully If, the window is too large, then the working set may encompass more than the current locality, thus affecting the degree of multiprogramming, adversely. The window size can be fine-tuned by observing page fault frequency of active processes. If the page fault frequency is found to be too high, then the. window size are increased; else if the PFF is found to be too low, then the window size is reduced.

Thus, if and when new pages are needed and no free frames are available, the OS may swap-out (move from RAM to Disk) some of the active processes. Conversely, when sufficient number of free frames are available after fulfilling the requirements of working sets of active processes, the system may swap in (move from disk to RAM) and activate e. more processes.

The working set principle provides a local replacement policy, to prevent 'Oohing, while maintaining a high degree of multi-programming.

The Concept of Thrashing

;A process is said to be Thrashing, if it is spending more time in paging tin in execution. This would occur, when a process is having insufficient ;umber of frames allocated to it, which may be inadequate to conarnodate all the pages, belonging to its current locality of reference. whenever, a new page is to be brought in, a page that may be currently free, has to be sent out. The victim page would need to be brought in gain,, shortly. This leads to sending in and out of same set of pages, in quick succession. Such a process would spend more time in paging an executing. The phenomena is known as Thrashing: This will result .a poor' throughput of the system.

Remedy of Thrashing

The OS has to swap out some of the active processes, thus releasing some occupied frames, that can be made available to the other active processes, till PFF is brought within acceptable limits. When some of the active ;processes have completed, the swapped-out processes can be swapped-in and their execution resumed.

4.5 DEADLOCK HANDLING

system Model

A Computer System supports a number of Resource Types like CPUs, Disk Drives and Magnetic Tapes etc. Each of the Resource Types may a number of identical copies, called its Instances. The resources ; are shared amongst' a number of concurrently executing processes.. Whenever, a process needs a resource during its execution, it requests he OS for assignment of that resource. When, an instance of the ':requested resource is available, it will be assigned to the requesting tocess. Till then, the process has to wait.

A Cycle of Accessing a Resource

1. Request. A process, needing a resource, will request the OS for assignment of the needed resource. Then the process waits, till OS assigns it an instance of the requested resource.

2. Assignment. The OS will assign to the requesting process an Instance of the requested: resource, whenever, it is available. Then, the process comes out of its waiting state.

3. Use. The process will use the assigned resource. In case, the resource is non-sharable, the process will have exclusive access to it

4. Release. After the process finished with the use of assigned resource, it will return the resource to the system pool. The released resource can now be assigned to another waiting process.

The `Request' and 'Release' are System Calls. If a resource is protected by a Semaphore, then the 'Request' would comprise a 'Wait' call on the Semaphore and the 'Release' would comprise a `Signal' call on the Semaphore.

Deadlock

It is a condition, wherein a set of processes are waiting forever for the resources, held by each other. None of them is able to proceed with its execution.

Example:

Semaphore Lock A, Lock_B; /* Semaphore counts initialized to Zero *

Process Po:

Wait (Lock_A);

Read (A);

A= A - 100;

Write (A);

```
 Wait (Lock B);
Read (B);
 B= B+ 100;
 Write (B);
 Signal (Lock_A);
 Signal (Lock_B);
```

Process P1:

```
Wait (Lock_B);
 Read (B); Wait (Lock_A);
 Read (A);
Signal (Lock_B);
 Signal (Lock..A.);
```

Suppose the execution of Po and P1 is overlapping in the following sequence:

```
Process PQ                    Process Pi
                             Wait (Lock_B); Read (B);
Wait (Lock_A);
 Read (A);
 A = - 100; Write
 (A); Wait (Lock_B);
 /* Bloated at Lock_B
                                 Wait (Lock A); /*
                                  Blocked at Lock_A
```

The Processes Po and P1 will remain blocked forever, Po waiting to access B, which. is held by P1 and P1 waiting to access Po which is currently held by Po. It is a deadlock condition.

4.6 NECESSARY CONDITIONS FOR DEADLOCK TO OCCUR

The following four conditions must hold simultaneously, for a deadlock to occur:

Mutual Exclusion

Some processes must hold some resources in a Non-Sharable Mode or Mutually Exclusive Mode. Like in the above example, Po is holding A and P1 is holding B, both in Mutually Exclusive Modes.

Hold and Wait

Some processes must be holding some resources in a non-sharable mode and at the same time must be waiting to acquire some more resources, which are currently held by other processes in a Non-Sharable Mode. Like in the above example Po is holding A in a non-sharable mode and at the same time trying to acquire B (by executing a Wait (Lock_B) call), which is currently held by P1 in a non-sharable mode. LikeWise, pi is holding B in a non-sharable mode and at the same time trying to acquire A (by executing a Wait (Lock,A) call), which is currently held by Po in a non-sharable mode.

No Preemption

The resources held by processes cannot .be preempted forcibly. A process releases the resource voluntarily when finished with it. Like in the above example, the processes release the resources A and B voluntarily by the execution of Signal Calls.

Cyclic Wait

A set of processes, say (P0, P1, PN.11 must wait in a cyclic fashion for the resources held by each other. That is,

P1 waiting for the resource held by Po.

P2 waiting for the resource held by P1.

PN-1 waiting for the resource held by

PN-2• P0 waiting for the resource held by PN.i.

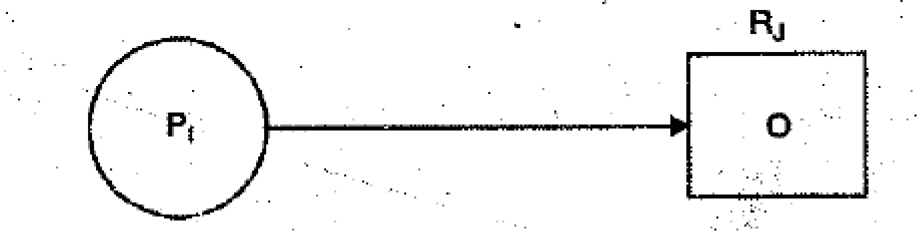
Like in the above example P1 is waiting for the resource held by P0 and P0 is waiting for the resource held by P1, thus satisfying the condition of cyclic wait.

4.7 HOW TO DETECT A DEADLOCK?

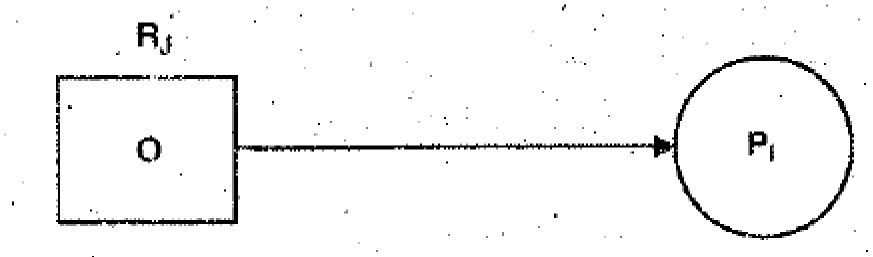
Resource Allocation Graph (RAG)

The Resource Allocation Graph is a directed graph, wherein the vertices represent . the Resources and Processes. A resource is represented by a Square, with dots inside the square representing different instances of that resource. A process is represented by a Circle, with Process Name indicated with a label inside the circle.

An Edge from process Pi to resource Rj represents a Request Edge.



An Edge from resource RR to process P/ represents an Assignment Edge.



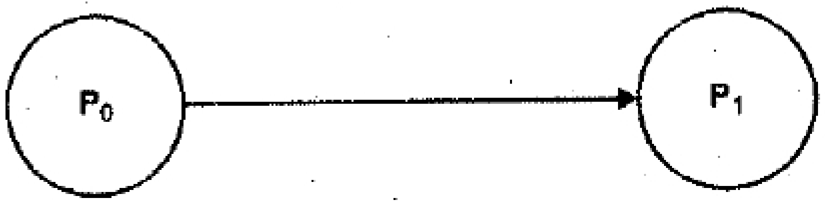
The Assignment Edge originates from the Instance within the resource. • symbol, to indicate which instance is assigned to the process. If there is no cycle in the Resource Allocation Graphs, it indicates that NO DEADLOCK EXISTS.

If there is a cycle detected in- the graph and the resources involved in the cycle have only single instance per resource, then it indicates that DEADLOCK EXISTS.

if there is a cycle detected in the graph and some of the resources involved in the cycle have multiple instances per resource, then there 0: AY BE A DEADLOCK.

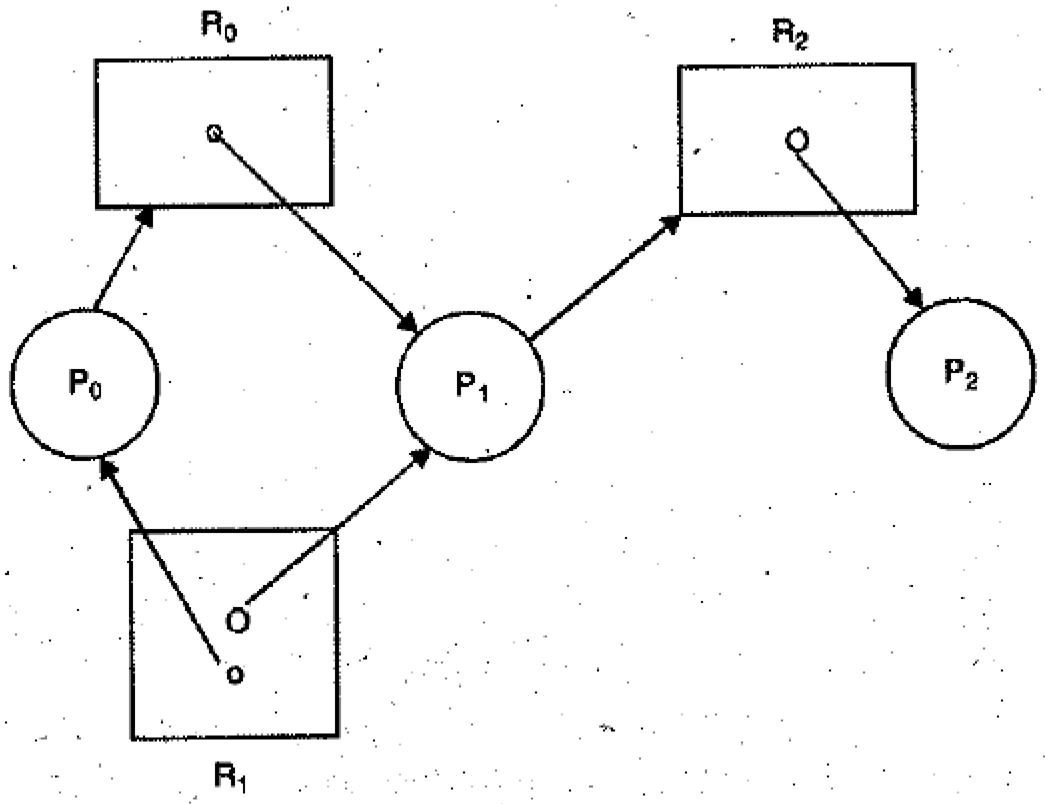
Process Wait For Graph (PWFG)

A Process Wait For Graph can be obtained by collapsing Resource 'Symbols in the Resource Allocation Graph. An edge from Process Po to P1 indicates that process Po is waiting for a resource, currently held by P1.

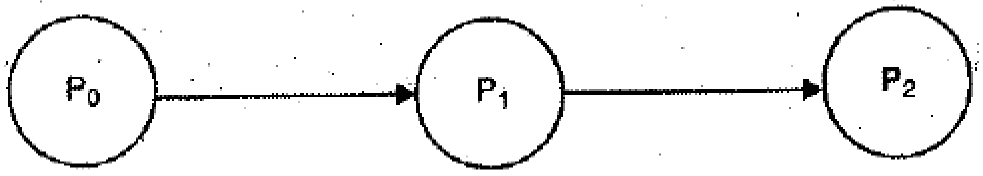


The above rules of detection of deadlock, as applicable to RAG, are also applicable to PWFG.

Example 1:



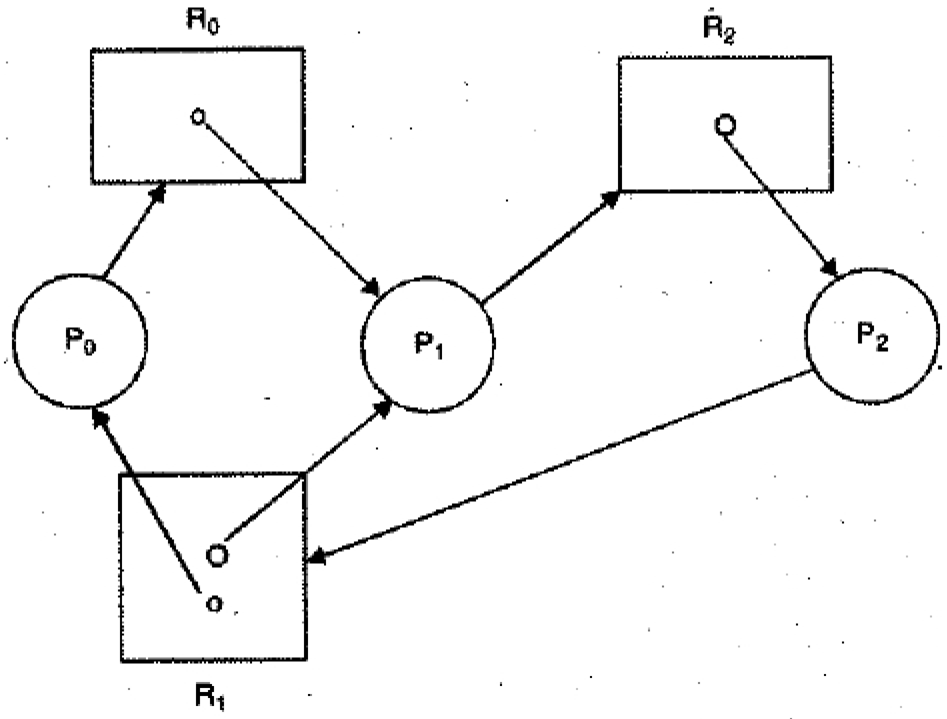
Equivalent Process wait for Graph (PWFG)



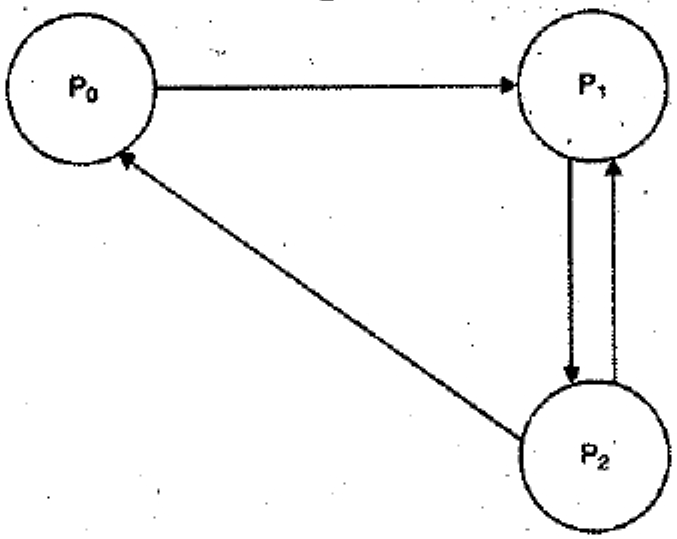
The RAG contains no cycle, thus indicating that NO DEADLOCK EXISTS, which is true, since P2 is not waiting for any resource and its execution can be completed. Once P2 completes, it will release one instance of Resource R2, for which P1 is waiting. Now, P1 can be assigned the released resource and P1 can complete its execution. Now, P1 will release an instance of Ro , for which P© is waiting. This will enable Po to proceed, thus all processes will be able to proceed. Thus, NO DEADLOCK exist

Likewise, there is no cycle in the equivalent PWFG, thus indicating NO DEADLOCK,

Example 2:

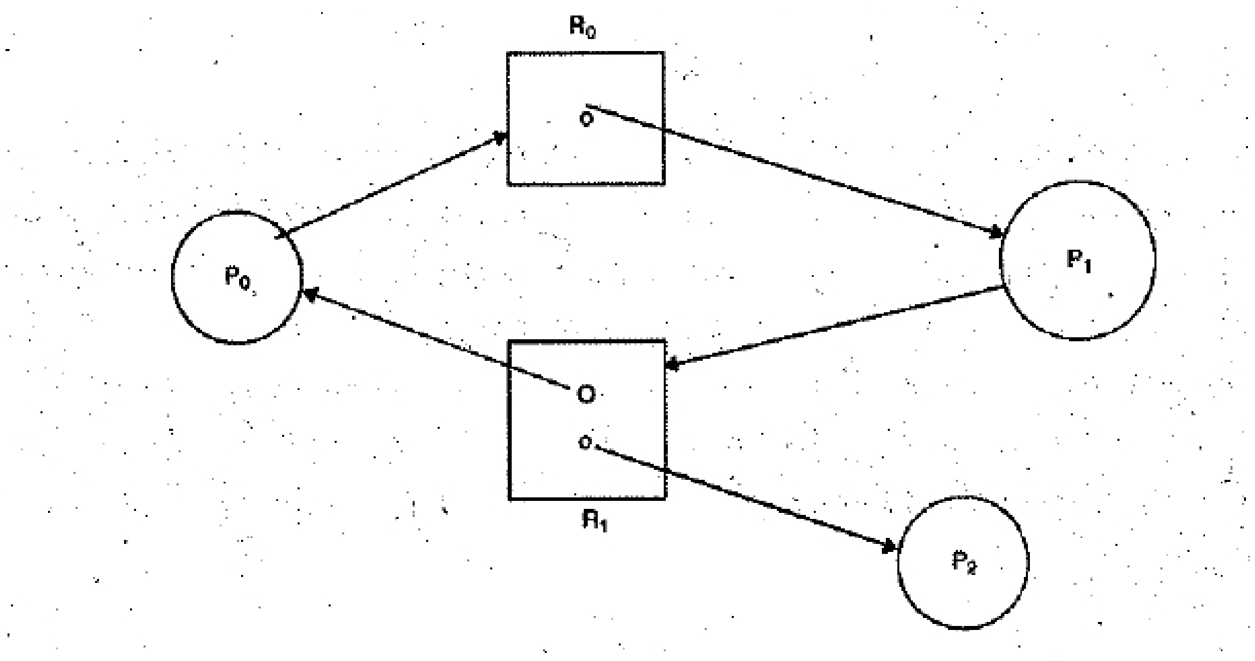


Equivalent Process Wait for Graph

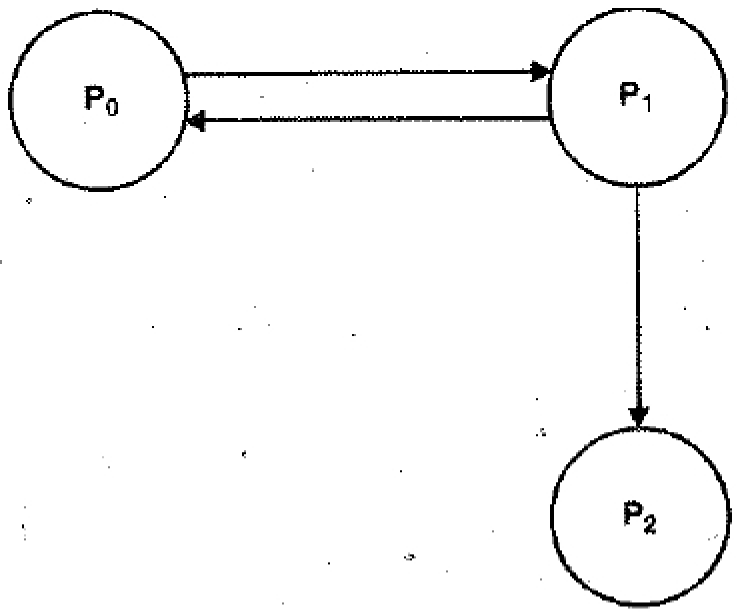


There is a cycle in the RAG i.e., Po Ro Pl -÷ R2 P2 —4 Ri Po. There is another cycle i.e., P1 —> R2 —4 P2 --> P1 One of the resources involved in the cycle i.e., R1 has two instances. So, there can be a deadlock. We can see that all resources are currently occupied, and all processes are waiting for some of the occupied resources. So, no process can proceed. Thus, there is a deadlock. Likewise, there are two cycles in the PWFG also.

Example 3:



Equivalent Process Wait for Graph



There is a cycle in the RAG i.e., Po Ro --->P1R1 -} Po. There is one resource i.e., R1 having two instances. So, there is a likelihood of Deadlock. PWFG also shows one cycle. Actually, there is no deadlock, since process P2 is not waiting for any resource and it can be completed. Once P2 releases resource R1, P1 can be completed. Once P1-releases Ro, Po can be completed. Thus, there is no Deadlock. This is the case •of RAG indicating a cycle, but NO DEADLOCK, since one 'of the participating resources has multiple instances and one of the instances is held by a process P2, which is not included in the cycle.

4.8 METHODS OF DEADLOCK HANDLING

Deadlocks can be handkd by any of the following methods:

Deadlock Prevention

Deadlocks can be prevented from occurring by preventing one 'of the necessary four conditions (i.e., Mutual Exclusion, Hold and Wait, No Prevention and Cyclic Wait) from holding. If one of the conditions can be prevented from holding then deadlock will not occur.

Deadlock Avoidance

Deadlocks can be avoided by maintaining the system always in Safe State. The system is said to be in safe state, if all pending processes can be successfully executed in some sequence, while satisfying their resource requirements fully. The sequence in which the pending processes can be executed is called a Safe Sequence. A process has to a priori declare its maximum need of resources, that may occur during its entire execution. A process is admitted for execution, only if its max need of resources is within the system capacity. During execution, whenever, a process requests additional resources (of course within its declared max needs), the OS will check whether system will remain in a safe state, after the 'allocation of requested resources. If YES, the resources are allocated immediately, else the process is made to wait till resources can be allocated while maintaining the system in a safe state.

Deadlock Detection and Recovery

The above approaches have considerable overheads. A less costly approach, when system requirement is not very critical, can be:

• Allow deadlock to occur

• Detect deadlock, when it occurs

• Recover from the deadlock

Allow Deadlock to Occur, Restart when System Stops

Functioning This is least costly approach, which can be employed when application is not at all critical, When deadlock occurs, system will finally stop functioning. Joist reboot the system.

4.9 DEADLOCK PREVENTION METHODS

Prevention of Mutual Exclusion

Whenever feasible technically, the concurrent processes may be permitted a shareable access to resources. This would prevent the condition of 'Mutual Exclusion' from occurring.

Hold and Wait

To prevent this condition, one of the following approaches can be adopted:

(a) 'If a process is holding a set. S1 of non-shareable resources *and requesting an additional set S2 of non-shareable resources, which cannot be allocated immediately, then., preempt the set of resources Si; which are currently held by the process. Now, the process is put to wait slate and it is waiting for the set {SI E 821 of resources. But, this approach may not be always feasible, since the process might have made some updates in the resources held by it So, preempting the resources in the middle of its execution may leave the resources in an inconsistent state, which may, not be acceptable,.

(b) Second, a relatively safer state is, that d9 not preempt any resources initially. But later, as process is in waiting state, preempt some resources, which may enable other processes to continue. This technique is used in swapping.. A suspended process can be swapped 'out. (i.e., main memory occupied by the process is preempted) and the freed memory is allocated to another process.

No Preemption

The techniques to prevent the condition of 'Hold and Wait' from occurring e same as discussed under no Preemption'.

cyclic Wait

the condition of 'Cyclic Wait' can be preempted by imposing a hierarchical ordering on the resources and making it mandatory for the :':processes to request resources, strictly in consistence with the hierarchical ‘ordering. This is explained below:

(a) The available resources are ordered with respect to each other.

(b) And it is imposed on the Processes that resources can be requested strictly in ascending order (or strictly in descending order) of their hierarchy

Suppose, the restriction is that resources can be ordered strictly in ascending order of their hierarchy.

,Then a process, which has already got a Resource R1 allocated to it, :-,Cannot request another resource 113 if ORDER (Rd) < ORDER (R,) request such resource "R3, the process has to first release all those ,xesources RK for which ORDER (RK) > ORDER (Rd) .

This would prevent the condition of 'Cyclic Wait' from holding, thus .:;;:obviating any potential deadlocks. :

A major limitation of this scheme is that processes may be forced to :`;block some resources much in advance, so as to follow the hierarchy of ",requesting the resources. This can be prevented by a more logical Ordering of resources, to match the most frequent order of their usage-::

4.10 RECOVERY FROM DEADLOCKS

Once a deadlock is detected, the next step is recovery from the deadlock. The recovery comprises rolling back and restating of some of the processes involved- in the deadlock, till the deadlock is resolved. .• = When a process is rolled back, the resources held by the process axe ..preempted, and the work already performed by the process, is undone. golling back of a process P would be feasible, only if the system has saved necessary information about its previous states. But, the saving :of this information involves a overheads. The two approaches used :.for rolling back are:

(a) Rolling back all the processes involved in deadlock to their initial states and then restart the processes.
(b) Roll back processes one by one, till deadlock is resolved. This approach is less costly as compared to previous approach. The choice of victims for rolling back is based on some criteria, like:

(i) How much a "process has progressed?

(ii) How much execution is it left with?.

(iii) The extent of resources held by the process? etc.

How to avoid process starvation, caused by rolling bac a process again and again?

To prevent a process from becoming victim of roll-back again and ag the system can znaintain a count of roll-back and the max number 0 times that a process can be rolled-back could be a system parameter.

4.11 DEADLOCK AVOIDANCE

Banker's Algorithm

It is a Deadlock-Avoidance Algorithm. It is used to avoid deadlocks, b maintaining the system in a safe state. The algorithm operates as?, follows:

(a) When a New Process is admitted into' the system, the process 1, has to pre-declare its maximum requirement of resources that may occur any time during its execution. 'The process will be admitted for execution only if its maximum requirement of resources is within the system capacity of resources. The processes admitted for execution can be safely executed in some sequence, called SAFE SEQUENCE.

(b) Whenever, a process requests allocation of resources (of cours(0 within its Maximum Requirement; already declared), the system'i proceeds as follows:

If the current availability of resources is sufficient t meet the request, the system proceeds to determine if the system would remain in a safe state subsequent t the grant of request. If YES, then the request is granted immediately; else the request is kept pending and they; requesting process is blocked, till the request can be granted safely.

Let N be the number of processes (P0, .Pr •••PPN-1 admitted' for

execution.

Let M be the number of types of resources f Ro, suPporte44.

by the system.

We will use following notation

: The i row of a NXM matrix will be treated as a vector of size N will be denoted as Ai. Suppose X, Y & Z are vectors of size M, then

(a) (X <= Y) will be true, only if X[k] < = Y [k] for all k 0 M-

(b) Z = X -Y would imply Z[k] =)(DO - Y (kJ for all k = 0...M-1

The algorithm makes use of the following data structures:

CAPACITY This is a vector of size M, indicating the system capacity each Type of resource. If CAPACITY [j] = k, it implies that system supports a total of k instances of resource R,

MAX This is N X M Matrix, which indicates the Maximum requirements System resources of each process. If MAX MIA = k, it indicates that rams Pi would need Max k instances of Resource ; during its entire execution.

ALLOCATED. This is N X M Matrax, which indicates the resources located to each Process. If ALLOCATED [i] = k , it indicates that recess Pi has got k instances of resource RJ currently allocated to it.

NEED. This is N. X M Matrix, which indicates the remaining need of resources of each process. If NEED MU] = k, it implies- that process Pi ay need additional k instances. of resource R. to complete ,its execution.

:NEED [i][j] = MAX [i][j].; ALLOC [i][j]

REQUEST. This is a matrix of dimension N X M, which indicates Pending request of resources' in respect of all processes. REQUEST = k implies that Process Pi has requested additional allocation of instances of Resource

REQUEST [i][j] will be valid request only if REQUEST [i][j] < NEED [i][j] else request' will be rejected as: invalid.

.AVAILABLE. This is a vector of size M, indicating the current availability of each Type of resource. If AVAILABLE lj] = k, it implies it that system supports a total of k instances of resource During system initialization AVAILABLE is assigned the value of system CAPACITY i.e., AVAILABLE = CAPACITY

Later, when some resources have been allocated to some processes: AVAILABLE = AVAILABLE - Z oi=14-1- ALLOCATED M

Algorithm

When a process Pi submits a REQUEST [I],

It is checked up for validity i.e., REQUEST[I] < NEED[I] 'should be true.

If YES, then system checks for availability of resources to meet the quest i.e., REQUEST[I] < AVAILABLE should be true; else the process is made to wait.

If both the above conditions are satisfied, then the REQUESTM tentatively granted and system tables are temporarily modified (w' the option of reverting back to the old values, as they existed prior tentative grant of .the request)

Step 1: Set AVAILABLE = AVAILABLE - REQUEST [I]

/* The request is granted tentatively */

ALLOCATED [I] = ALLOCATED [I] + REQUEST [I];

NEED [I] = NEED [I] - REQUEST [I];

Let COMPLETED be a vector of size N.

Set COMPLETED m = false, for all I = 0 .. N-1

Now, Banker's algorithm is invoked to determine whether system is in safe state. If YES, the grant of REQUEST[11 is firmed 4, else we revert to old values in the system tables and REQUESTA is kept pending, till it can be granted safely.

Step 2: Determine k such that

COMPLETEDfld == false && NEED [k] < TEMP

If no such k exists, then goto Step 4.

Step 3: /*The process k can be' completed*/

Set COMPLETED [k] = true;

. Set AVAILABLE = AVAILABLE + ALLOCATED [k]

/* After Process K is completed,• it will

release the resources allocated to it */

goto Step 2;

Step 4: If (COMPLETED fK1 = = true for each process K = 0 ... N-1)

/* Then each process can be completed in some sequeng which indicates that the system is in SAFE STATE. So firm-up the grant of request by maintaining updated tables as such*/

Set REQUEST [I] = 0; /* Already granted*/

Else

/* The System is in Unsafe State. -So, keep

REQUEST[Ii pending, till it can be granted safely */

/* Revert back to old values in system tables

*/ ALLOCATED [I] = ALLOCATED [I] .REQUEST

NEED [I]. = NEED [II. + REQUEST [I];

AVAILABLE = AVAILABLE + REQUEST [l]

A ‘C'-like algorithm, to simulate BANKER'S ALGORITHM

define true 1

define false 0

MAX[N](Ml; ALLOC (M EM, NEED [N] [M], REQUEST M[M); t AVAILABLE EMI, CAPACITY [M]

The CAPACITY table is initialized during System Configuration. It changes only when the System Configuration changes. Other tables are initialized, when system is restarted. ::'AVAILABLE = CAPACITY; other tables are initialized to O.' Whenever, a new process Pi is to created, it has to specify a priori its maximum requirement of resources i.e., MAXM that may occur anytime during its execution. A New Process is accepted for execution, only if .MAXM < = Capacity; else the process is deleted. If the process has -,:been accepted, its Max requirement is entered in the Max table.

whenever, a REQUESTW, for allocation of resources is received from *process Pi, it is processed as follows:

```
If (REQUEST[I] <= NEEDM ) /* "A valid request */

( if ( REQUESTM <= AVAILABLE ) /*Resources available to  grate,

                                              The request*/

   ( /* Grant the request Tentatively, as follows: */

 ALLOCATED [I] = ALLOCATED [I] 4- REQUEST

NEED = NEED - REQUEST [I];

 AVAILABLE = AVAILABLE - REQUEST [I];

/*Invoke BANKER'S ALGORITHM System Safe

 ( determine whether System remains in safe
the grant of request */
 If (System Safe( ))
( /* Firm-up the grant Of REQUEST[I1 */
 /* Leave system tables updated as such */
 REQUEST[I]  0/* Already granted



                                    ALLOCATED[I] = ALLOCATED[I]
REQUEST[I];
NEED[I] = NEED[I] + REQUEST[I];
 AVAILABLE = AVAILABLE + REQUEST [I]
int System_Safe ( )
int Completed[N], .Temp[N], Safe_Sequence [N];
 int Found, Count, j
; Temp = Available;
 Count = 0;
for (j=0; j <N; j++)
 /*unmark all processes */
 Completed[j] = false;
1
Found = false;
 j = 0; while ( Found && (j < N) )
{ if El Completed fj1 && (NEED[j] <= 'temp))
{ Completed [i] = true;
```

```
 Safe_Sequence [Count] =j;
 Count++;
 Temp = Temp. + ALLOCATED[j]
; Found —true;
else j++
; if (Found) goto 1;
 if (Count = = N) return 1; /* System Safe */
 else return 0; /*System unsafe */
```

source Code of running program, to simulate anker's Algorithm

```
******SIMULATION OF BANKER'S ALGORITHM
                          by P S Gill
#include <stdio.h>
*include <conio.h>
 Odefine N 05 /* Max number of processes. ids 0..N-1*/
 #define M 04 I* Max Types of Resources id 0..M-1 */. //
 Type Definitions
  typedef enum boolean {false, true};
 // Global Variables
 in max [N][M] =
{{0,0,1,2,}
{1,7,5,01,
{2,3,5,6},
 (0,6,5,2},
 {0,6,5,6}}
; int alloc EN] [M], need [NI [Ml, request [N] [M];
Int capacity[M] = (3,14,12,12);
 int available [M];
mt safe sequence[N];
mt choice;
char ch;
****.%*****
;void tentative_ rant (int p_id)
(- int j;
 for (j =0; j<M ; j++)
 { aline (p_idl(j1 = allodp_ic111,11 + request (p_idlij1;
 need {p_idlij1 = need (p:_id] ii] — request [p_id]
 available [j] = available [j] 7 request [Aid] [j];




/I
void revert back (int p_id)
 ( int j;
 for (j =0; j<M; j++)
allot alloc[p_id] [j] — request [p_id] ji;
 need [p_id] Ejl = nc,,edfp_idj[j] + request [p_id] ;
 available [j] = availableijj + request [p_id](}];


int need_met(int int j
; ' int flag= 1;
 j = 0;
while (flag && j<M) if (available Ii] < need(i]iji)
 flag = 0; else j++;
return flag;


int request_valid (int i)
 ( int j;
int flag= 1;
 j = 0;
while (flag && j<M)
 (if (need[i]0] < request [i][j]
 flag = 0;,
 else j++;
return flag; //
int res_available (int i)
int j;
int flag= 1;
j = 0;
while (flag && j<M)
(if (available [i] < request [i] [i]
flag = 0;
```

```
else j++;
}
return flag;
}
//----------------------------------------------------
int system_safe ()
{ int ij, count;
int completed[N];
int found;
int save_available[M];
for (j=0; j<N; j++)
save_available [j] = available [j];
for (i=0; i<N; i++)
completed [i] = 0;
count=0;
loop: found = 0;'
i=0;
while (Ifound && (i <N))
( if (lcompleted[i] && need_met(i))
[ completed [i] = 1;
found = 1;
safe_sequence [count] = i;
count++;
for (j=0; j<M; j++)
available [j] = available[j] +alloc [i][j];
}
else i++;
if (found) goto loop;
else ( /*unsave available*/
for (j=0; j<M; j++)
available [j] = save_available [j];
if (count == N) /*system safe */
return 1;
else return 0; /*system not safe */
}
//---------------------------------------------
//---------------------------------------------
void display_state ()
{ int i,j;
cIrscr();
printf (‘’n\n”);
printf(" MAX ALLOC NEED AVAILABLE\ n");
printf("  \n");
for (i=0;i<N;i++.)
printf ");
for (j=0;j<M;j++)
printf("%2d  ‘’,max [i] [j])
printir ");
for (j=0;j<Mj++)
printf("%2d ‘’, alloc [i][j]);
```

```
printr ");

for (j-0j<M;j++)

printf("%2d ",need[i][j];

printg" ");

if (i=0)

for (j=0<jM;j++)

printg"%2d ",available[j]);

printr\n");

printf ("\n\n");

if (choice == 2)

{   if (system_safe()

printf ("Safe Sequence —> ");

for (i=0; i<N; i++)

printf (" %d ", safe_sequence[i]);

printf (" \ n \n");

}

else printef system NOT in. safe state");


}

//------------------------------------------------------------------------------

void main ()

{int ij, p_id;

char ch; .

for (j=0; j<M; j++)

available[j] =capacity[j];
```

```
for (i=0;i<N;i++)

for (j=0j<M;j++)

{ need [i] [j]=max[i][j];

alloc [i]=0;

}

choice=0;

do    { display_state();

         printf(" 1: Enter next request \n");

         printf" .2: Display Safe Sequence n");

         printf"  9: Quit \n");

         printf("\n\n Enter Choice:");

         scanf"%d",&choice);

         if (choice == 1)

         [ printf("Enter Process Id (0..4): ");

         scanf"%d", &p_id);

         printr(‘’Enter qty of resources requested \n");

         for (j=0; j<M; j++)

         [ printf (" Resource[%d]: "j);

            scanf ("%d",&request[p_id][j];

            }

           if (request_valid(p_id) && res_available(p_id))

           { tentative_grant(p_id);

           for (i=0; i<N; ++)

           safe_sequence[i] = -1;

           if (!system_safe())
```

```
            revert_back(p_id);

            else for (j=0; j<M; j++)

             request [p_id][i] =0;

             }

             }

             {  while (choice !=9);

}

//---------------------------------------------------------------------------//
```

Problem

Two concurrent processes P1 and P2 want to use two processes RI and R2 in a mutually exclusive manner. Initially R1 and R2 are free. The programs executed by the two processes are given below:

```
S1: While (R1 is Busy) do no-op;

S2: Set R1 <— Busy;

S3: While (R2 is Busy) do no-op;

S5: Use R1 and R2;

S6: Set R1 -- Free;

S7: Set R2 -- Free;
```

P2:

```
Q1: While (R2 is Busy) do no-op;

Q2: Set R2 -- Busy;

Q3: While (R1 is Busy) do no-op;

Q4: Set R1 -- Busy;

Q5: Use R1 and R2;

Q6: Set R2 - Free;
```

Q7: Set R1 -- Free;

(i) Is mutual exclusion guaranteed for R1 and R2? If not, show a possible interleaving of the statements of P1 and P2 such that mutual exclusion is violated (i.e, both P1 and P2 use R1 or R2 at the same time).

Answer: Yes, Mutual Exclusion is guaranteed in all cases.

(ii) Can deadlock occur? If yes, show a possible interleaving of the statements of P1 and P2 leading to deadlock.

Answer: Yes, deadlock can occur, if statements are interleaved as:

S1, S2, Q1,Q2, S3, Q3

or

Q1, Q2, S1,S2, Q3, S3

or

S1, S2, Q1,Q2, Q3, S3

or

QI, Q2, SI,S2, S3, Q3

SUMMARY

- In this chapter we have seen the concept of Memory-management algorithms for multi programmed operating systems that may range from simple single user system approach to paged segmentation approach. For the simplest systems once a program is loaded into memory, it remains there until it finishes. Some operating systems allow only one process at a time in memory, while some other supports multiprogramming.

- When swapping is used, the system can handle more processes than it has space for in memory, Processes for which there is no space in memory, are swapped out to the disk.

- The memory management algorithms as paging, segmentation, combination of paging and segmentation consider various parameters as hardware support, performance, fragmentation, relocation, swapping, sharing and protection.

- More advanced computers often have some form of virtual memory. Virtual memory is a technique to allow a large logical address space to be mapped onto a' smaller physical address space. It allows large processes to be run, and also allows the degree of multiprogramming to be raised, increasing the CPU utilization.

- If memory requirements exceed the physical memory then page replacement is required to free the frames for new pages. Various page replacement algorithms are available for this purpose. FIFO page replacement is easy to use but suffers from Belady's anomaly. Optimal page replacement requires future knowledge. LRU replacement is an approximation of optimal but it is difficult to implement:

- In practical paging systems, other aspects than the paging algorithms are important. It includes the choice of working set model versus demand paging, local versus global allocation, page size, and many implementation issues.

- A deadlock state occurs when two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.
  There are three methods for dealing with deadlocks:

  (a) Use some protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.
  (b) Allow the system to enter deadlock state, detect it and then recover.
  (c) Ostrich algorithm: Ignore the problem all together, and pretend that deadlocks never occur in the system.

- A deadlock condition may occur if and only if the following four conditions hold simultaneously in the system:

(a) Mutual Exclusion	(b) Hold and Wait
(c) No Preemption	(d) Circular Wait

- Another method for avoiding deadlocks is to have a priori information on how each process will be utilizing the resources. The banker's algorithm, for example needs to know the maximum number of each resource class that may be requested by each process.

- If a system does not employ a protocol to ensure that deadlocks will never occur, then detection scheme .must be invoked to determine whether a deadlock has-

Occurred. If a deadlock is detected, the system must recover either by terminating some of the deadlocked processes, or by preempting resources from some of the deadlocked processes.

REVIEW QUESTIONS

1. Given memory partitions of 100 KB, 500 KB, 200 KB, 300 KB and 600 KB (in that order), how would the First-Fit, Best-Fit and Worst-Fit algorithms place the processes A: 212 KB, B: 417 KB, C: 112 KB and D: 426 KB (in that order). Which algorithm makes the most optimal use of the memory?

2. Consider a logical address space of 12 pages of 1024 bytes each, mapped onto a physical memory of 64 frames. What would be minimum number of bits in the
   (a) Logical Address (b) Physical Address
   and what would be the size of physical memory.

3. Explain the difference between internal fragmentation and external fragmentation. Which one occurs in paging systems? Which one occurs in systems using pure segmentation?

4. A 16 bit computer has a page size of 4096 bytes. The page table of a process A is:

Page number	Frame number
0	7
1	2
2	5
3	1
4	12
5	6
6	0

   For the following logical addresses, determine the corresponding physical addresses:
   (a) 3720 (b) 7512 (c) 22340 (d) 17510 (e) 11225

5. A 16 bit computer is implementing segmentation. Higher order 4 bits are used to specify segment number and lower order 12 bits specify offset with the segment. The segmentation table of a process A is as follows:

Sagment #	Sagment Limit	Sagment Base
0	1500	500
1	4000	8500
2	3520	2500

3	540	7000

For the following logical addresses, determine the corresponding physical addresses:
(a) 1400 (b) 11226 (C) 7012 (d) 12324 (e) 6092

6. A computer with a 32-bit address uses a two level page table. Virtual addresses are split into a 9 bit top-level page table field, an 11-bit second-level page table field, and an offset. How large are the pages and how many are there in the virtual address space?

7. Assume a process has a reference string of length p, with n distinct page numbers. if the number of page frames is m (initially all empty), what will be lower bound and upper bound of page faults?

8. A demand-paged virtual memory system has following parameters:
(a) Time to service a page fault
    (i) When a new frame is allocated): 12 ins
    (ii) When a modified page is replaced): 20 ms

(b) Memory access time: 200 ns

(c) While servicing a page fault,
    (i) Probability of allocating new frame: 30%
    (ii) Probability of replacing a modified page: 70%

Find the max page fault rate such that degradation in memory access time is not more than 20%.

9. In a virtual memory system, with page size of 200 bytes, consider the array Int A (100] [100]; which is stored starting from address 200. Find the number of page faults for LRU Algorithm, in respect of the following two programs, if the program is stored in page 0 (between address 0 and 199).

```
(a) for    ( i = 0; i < 100; i++ )
    for    (j = 0; j < 100; j++)
            A [i] [j] = 0;

(b) for    ( j = 0; i < 100; i++ )
```

```
for         (i. = 0; i < 100; i++)
              A [i] [j] = 0;
```

10. Consider the following page reference string:- 1, 2, 3, 4, 2, 1,5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

    How many page faults will occur for the following page replacement algorithms? Assume a set of three page-frames (initially all empty).
    (a) FIFO (b) LRU (c) Optimal

11. Prove the Belady's Anomaly in FIFO Page-Replacement- Algorithm, assuming that the number of frames is increased from 3 to 4; for the page reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
    Assume all frames are empty initially.
12. What is the concept of Contiguous Memory Allocation? What are its advantages and disadvantages?

13. Explain the difference between internal and external fragmentation.

14. Describe the' following Partition Allocation Strategies:
    (a) First Fit (b) Best Fit (c) Worst Fit

15. Consider a swapping system in which memory consists of the following hole sizes in memory order: 10 k, 4 k, 20 k, 18k, 7 k, 9 k, 12 k, and 15 k. Which hole is taken for successive segment requests of
    (a) 12k (b) 10 k (c) 9 k

    for First fit? Now repeat the question for Best fit and Worst fit.

16. A Worst fit allocator always splits the largest free memory area while making an allocation. Compare its advantages and drawbacks with the Best fit and first fit allocators.

17. Discuss the strategy for protecting the operating system from the user processes, and protecting user processes from one another?

18. Explain the concept of Non Contiguous Memory Allocation. How can it be implemented?

19. Whit are the possible ways to reduce the Memory-overhead of a Paged System?

20. Why are Segmentation and Paging sometimes combined into one scheme?

21. What are the Physical addresses for the following logical addresses?
(a) 0430 (b) 110 (c) 2500
(d) 3400 (e) 4112

Consider the following segment table:

Sagment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

22. How can we reduce the Effective Memory-Access-Time in a Paged system?

23. Consider a paging system with the page table stored in memory.
(a) If a memory reference takes 100 nanoseconds, how long does a paged memory reference take?

(b) If we add TLBs, and 75 per cent of all page-table references are found in TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs take zero time, if the entry is there.)

24. Why are page sizes powers of two? Why is this necessary?

25. What are the common techniques used for structuring the page table?
26. What is Deadlock and what are necessary condition to hold it? Discuss Deadlock Detection and Recovery scheme.
27. Consider the following snap-shot of a system:

Current State

Process	Allocated			Max			Available			
	R1	R2	R3	R1	R2	R3	R4	R1	R2	R3
P1	2	0	0	1	1	0	2	0	0	0
P2	3	1	0	2	0	0	0			
P3	1	3	0	6	0	0	1			
P4	0	1	1	4	0	1	0			

(a) Draw the Resource Allocation Graph.

(b) Draw the Process Wait For Graph.

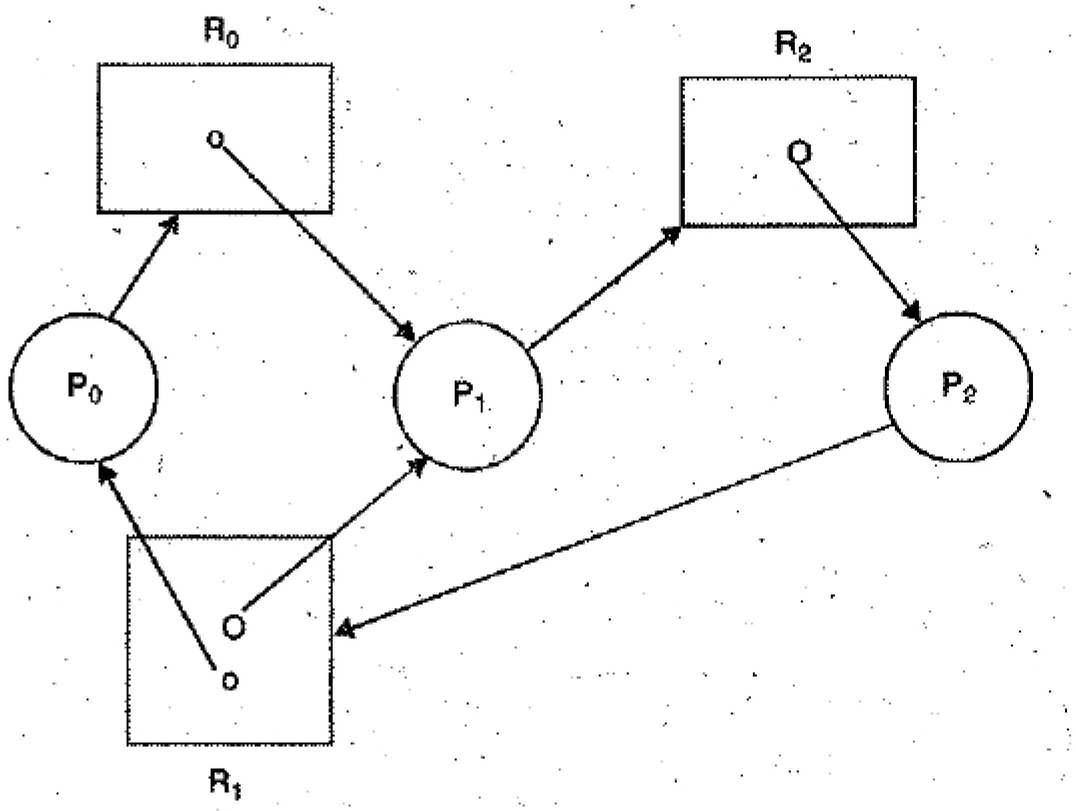
(c) Is the system deadlocked?

28. What is the minimum number of resources needed to be available for the state given below to be safe:

Current State

Process	Allocated	Max
	R1	R1
P1	1	3
P2	1	2
P3	3	9
P4	2	7

29. Consider the following Resource Allocation Graph:



(a) Is the System in a Safe State?
(b) Draw equivalent Process Wait For Graph (PWPG).

30. Explain the System Model of Deadlock. What are the necessary conditions for Deadlock to occur?

31. Explain the use of Resource Allocation Graph (RAG) and Process Wait For Graph (PWAG).

32. What are the methods of Deadlock Handling?

33. Explain the Deadlock Prevention Methods in brief.

34. How to avoid process starvation, caused by rolling back a process again and again?

35. Explain Deadlock Detection with Example. How Deadlocks are recovered?

FURTHER READING

1. Operating Systems: A practical Approach: Rajiv Chopra, S. Chand Publisher, 2010

2. Operating Systems: Principles and Design: Pabitra Pal Choudhury, PHI Learning

3. Operating Systemes and Software Diagnostics: Ramesh Bangia and Balvir Singh, Laxmi Pub., 2007

4. Principles of Operating Systmes: Sri V. Ramesh, Laxmi Pub., 2010

UNIT V

RESOURCE PROTECTIONS

STRUCTURE

5.0 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- discuss the racecourse protections in Computer system
- discuss goals and domain of protection
- explain the Access Matrix Model
- define security problem authertication
- explain about password and it's methods
- know about system threats and program threats
- discuss about the Encryption

- define computer security classification.

5.1 INTRODUCTION

Security and protection deal with the control of unauthorized use and access to the hardware and software resources of computer system. Business organizations and government agencies heavily use computers to store information to which unauthorized access must be prevented.

For example, in business organizations, this information includes financial or personal records, monetary transactions, legal contracts payrolls, product information future planning and strategies etc., clearly, an unauthorized use of company's confidential information can have Catastrophic financial consequences and unauthorized use of secret information of the government can have serious implications for the security of that nation. Therefore, with the widespread use of computers in business and government organizations, the security and protection of computer systems have become extremely important factors. Computer system security can be divided into two parts.

1. **External Security:** External security as also called physical security It deals with regulating access to the premises of the computer system, which includes the physical machine (hardware, disks, tapes, power supply, air conditioning), terminals, computer console etc. External security can be enforced by placing a guard at door, by giving a key or secret code to authorized person etc.

2. **Internal Security:** Internal security deals with the access and the use of computer hardware and 'software information stored in computer system. Aside from external and internal securities, there is an issue of authentication by which a user "logs into" the computer system to access the hardware and the software resources.

Protection Vs Security

Hydra [39] designers make a distinction between protection and security According to them, protection is a mechanism and security is a policy. Protection deals with mechanisms to build secure systems and security deals with policy issues that use protection mechanisms to build secure systems.

5.2 GOALS OF PROTECTION

The definitions of protection goals can be divided into two groups ;
Group I: protection against everybody else (including communication partners).

Group II: protection against third parties.
In table, definitions of the protection goals unobservability and anonymity in relation to the definitions of the Common Criteria .

Table 5.1. Definitions of Protection Goals

Unobservability	Definition
CC	Ensures that a user may use a resource or service without others, especially third parties, being able to observe that the resource or service is being used
I. against all	Ensures that a user may use a resource without communication partners others being able to observe that the resource or service is being used,
II. against 3rd parties	Ensures that a user may use a resource or service without third parties being able to observe that the resource or service is being used.
Anonymity CC	Ensures that a user may use a resource or service without disclosing the user's identity
1. against all	Ensures that a user may use a resource or service without disclosing the user's identity to anybody else.
II. against 3rd parties	Ensures that a user may use a resource or service without disclosing the user's identity to third parties but possibly to the communication partners.

Based on the above definitions, it is now possible to set up a system of implications between the protection goals (See figure 5.1).

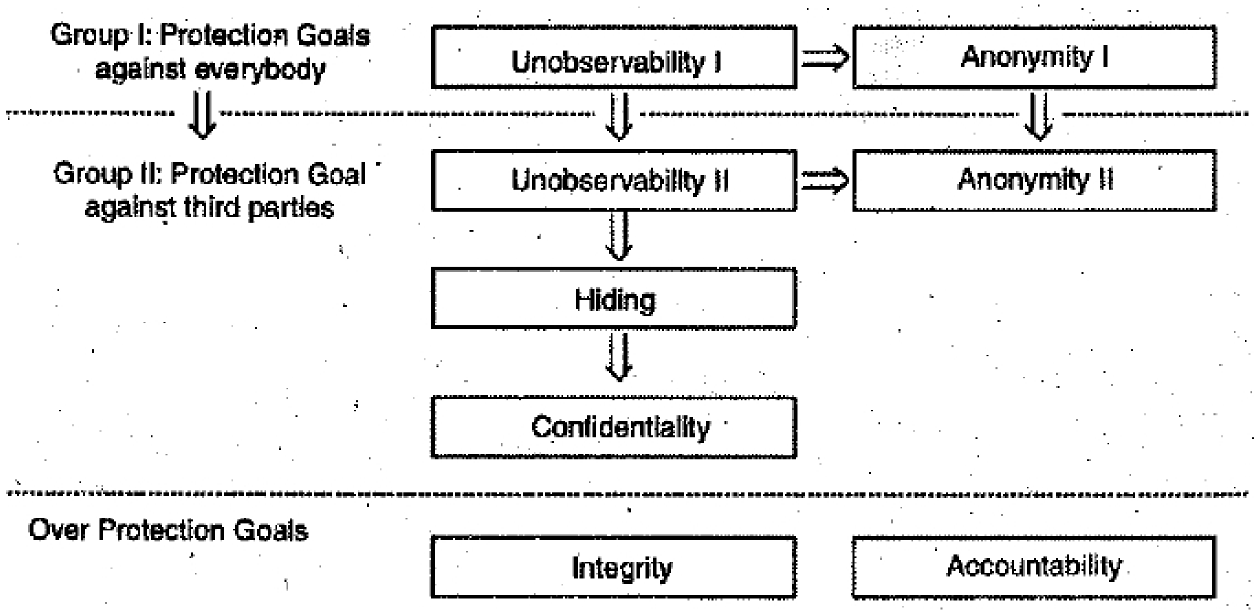


Figure 5.1.Implication.

5.3 DOMAIN OF PROTECTION

The protection domain of a process specifies the resources that it can access and the types of operations that the process can perform on the resources.

A protection domain is an execution environment shared by a collection of processes: it contains a set of < resource, rights > pairs, listing- the resources that can be accessed by all processes executing within the domain and specifying, the operations permitted on each resource.

Domain

It is defined as a set of (object, rights) pairs. Each pair specifies an object and one or more operations that can be performed on the object. Each one of the allowed operations is called a right.

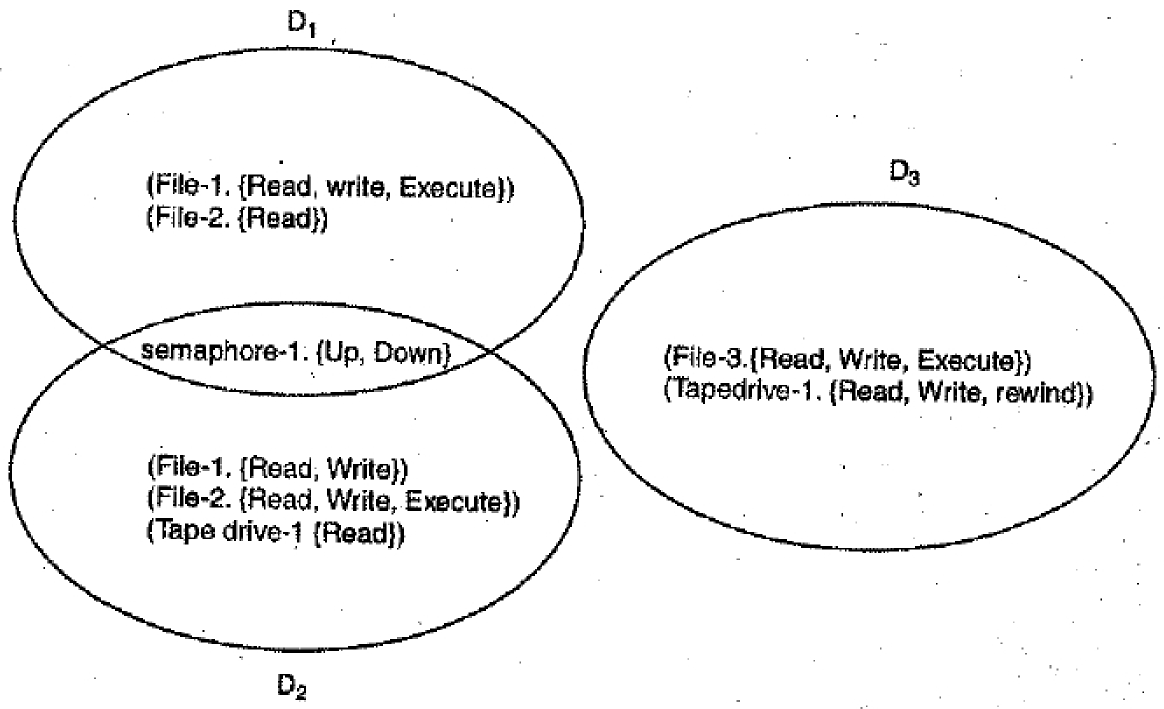
A security system based on the concept of the domain defines a set of domains with each domain having its own set of (object, rights) pairs; At any instance of time, each process executes in one of the protection domains of the system. Therefore, at a particular instance of time, the access rights of a process are equal to the access rights defined' in the domain in which it is at that time. A process can also switch from one domain to another during execution.

Figure 5.2 shows an example of a system having three protection Domains D1, D2, D3. The following points may be observed from this example:

1. Domains need not be disjoint. That is, the same access rights may simultaneously exist in two or more domains. For instance, access right (semaphore-1, (up, down)) is in both domains D1 and D2. This implies that a subject either of the two domains can perform up and down operations on semaphore-1.

2. The same object can exist in multiple domains with different right in each domain. For instance, rights for File-1 and File-2 are different in D1 and D2, and rights for Tape Drive-1 are different in D2 and D3. Considering File-1, a subject in domain D1 can read, write and execute, it, but a subject in domain D2 can only perform read and write operations on it. Therefore, in order to execute File-1., a subject must be in domain D1.

3. If an object exists only in a single domain, it can be accessed only by the subjects in that domain. For instance, File-3 can only be accessed by the subject in domain D3.



D_1
(File-1. {Read, write, Execute})
(File-2. {Read})
semaphore-1. {Up, Down}
(File-1. {Read, Write})
(File-2. {Read, Write, Execute})
(Tape drive-1 {Read})
D_2
D_3
(File-3.{Read, Write, Execute})
(Tapedrive-1. {Read, Write, rewind})

Figure 5.2. A system having three protection domain.

Since the domain is an abstract concept, its realization and the rule for domain, switching are highly system dependent for instance. Some ways for realizing the domain may be the following:

Each user as a domain: In this case, processes are assigned to domain according to the identity of the user on whose behalf they are executed. A domain switching occur when a user logs out and another user logs in.

Each process as a domain: In this case, the access rights of a process are limited to the access right of its own domain. A domain switching: occurs when the process sends a message to another process and then waits for a response.

Each processor as a domain: In this case, each procedure has its own set of access right and a domain switching occurs when a. procedure calls another procedure during the course of its execution.

<u>5.4 THE ACCESS MATRIX MODEL</u>

The access matrix model was first proposed by Lampson. It was further enhanced and refined by Graham, and Denning and Harrison. This model consists of the following three components:

Current objects: Current objects are a finite set of entities that access current is to be controlled. The set is denoted by 'O'. A typical example of an object is a file.

Current subjects: Currents subjects are a finite set of entities that access current objects. The set is denoted by 'S'. A typical example of a subject is a process. Note that S < O. That is, subjects can be treated as objects and can be accessed like an object by other subjects.

Generic Rights: A finite set of generic rights. R = {r1, r2, r3,   rm,) gives various access rights that subjects can have to objects. Typical examples of such rights are read, write, execute,' own, delete, etc.

The Projection State of a System

The projection state of a system is represented by a triplet (S, O, P), where S is the set of current subjects, O is the set of current objects, and P is a matrix, called the access matrix, with a row for every current subject and a column for every current object. A schematic diagram of an access matrix shown in figure 5.3

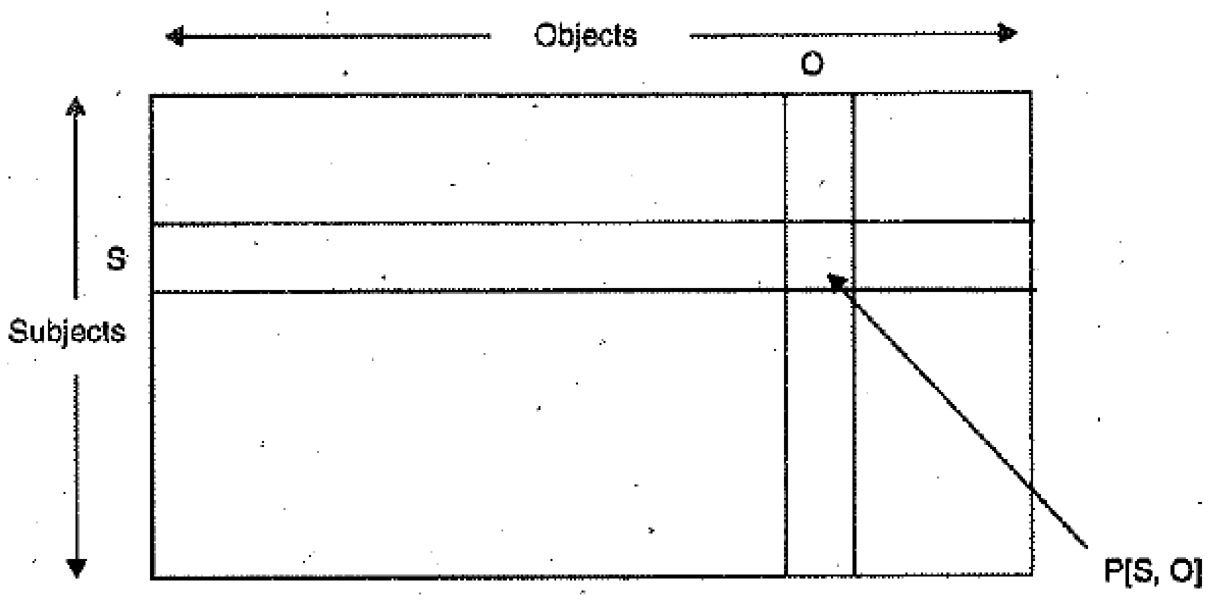


Figure 5.3 A schematic of an access matrix.

Note that the access matrix P itself is a protected object. Let variables S and 0 denote a subject and an object, respectively. Entry PCS, O] is a subset of R, the generic rights, and denotes the access rights which subject S has to object O.

Enforcing a Security Policy

A security policy is enforced by validating every user access for appropriate access rights. Every object has a monitor that validates all accesses- to that object in the following manner.

1. A subject S requests an "access α to object O.

2. The protection system presents triplet (S, α, O) to the monitor of O.

3. The monitor looks into the access rights of S to O. If α e P[S, 0], then the access is permitted; else it is denied.

The access matrix model of a protection system is very popular because of its simplicity, elegant structure, and amenability to various implementations.

Example 1. Figure 5.4 illustrates an access matrix that represents the protection state of a system with three subjects, S1, S2, S3, and five objects, O1, O2, S1, S2, S3. In this protection state, subject S1 can read and write object O1, delete O2, sendmail to S2, and the received mail from S3. Subject S3 owns PO1 and can read and write O2.

	O1	O2	S1	S2	S3
S1	Read, write	Own, delete	Own	Sendmail	Recmail
S2	Execute	Copy	Recmail	Own	Block, wakeup
S3	Own	Read, write	Send mail	Block, wakeup	own

Figure 5.4 An access matrix representing a protection state.

Implementation of the Access Matrix

Direct implementation of the access matrix for access control is likely to be very storage inefficient. In this section we study three implementations of the access matrix model.

Capability Based Method

The efficiency can be improved by decomposing the access matrix into rows and assigning the access rights contained in rows to their respective subjects. Note that a row denotes access rights that the corresponding subject has to objects. A row can be collapsed by deleting null entries for efficiency. This approach is called the capability based method.

Access Control List

An orthogonal approach is to decompose the access control matrix by columns and assign the columns to their respective objects. Note that a column denotes access rights of various subjects to the objects. A column can be collapsed by deleting null entries for higher efficiency. This technique is called the access control list method.

Lock Key

The third approach, called the lock-key method, is a combination of the first two approaches.

Capabilities

The capability based method corresponds to the row-wise decomposition of the access matrix. Each subject S is assigned a list of tuples (O; PES, O]) for all objects O that it is allowed to access. The tuples are referred to as capabilities. If subject S. possesses a. capability (O, P[S, PO]), then it is authorized to access object O in manners specified in P[S, O].

A schematic view of a capability is shown in Figure 5.5. A capability has two fields. First, an object descriptor, which is an identifier for the object and Second, access rights, which indicate the allowed access rights to the object. The object descriptor can very well be the address of the corresponding objects and therefore, aside from Providing protection, capabilities can also be used as an addressing mechanism by the system.

Object descriptor	Access rights
	Read, write, execute, etc

Figure 5.5. A schematic view of a capability

Advantage

The main advantage of using a capability as an addressing mechanism is that it provides an address that is context independent. That is, it provides an absolute address. However, when a capability is used as an addressing mechanism, the system must allow the embedding of capabilities in user programs and data structures, as a capability will be a part of the address.

Capability Based Addressing

Capability based addressing is illustrated in figure 5.6.

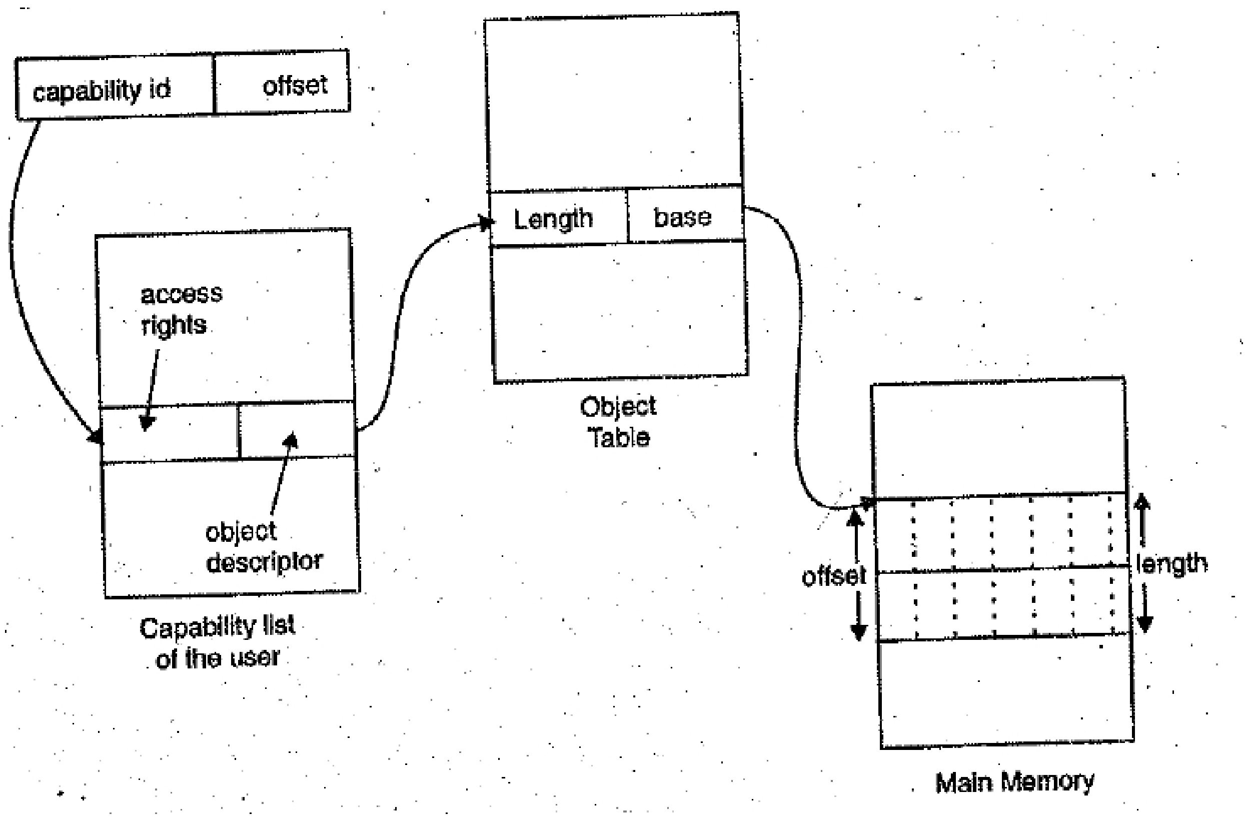


Figure 5.6. An illustration of capability-based addressing

A user program issue a request to access a word within an object. The address of the request contains the capability ID of the objects (which tells what object in the main memory is to be accessed) and an offset within the object (which gives the relative location of the word in the object to be accessed) The system uses the capability ID to search the capability list of the user to locate the capability that contains the allowed access rights and an object descriptor the system checks if the requested access is permitted by checking the access right in capability.

The object descriptor is used to search the object table to locate the entry for the object. The entry consists of the base address of the object in main memory and the length of object. The system adds the base address to the offset in the request to determine the exact memory location of the accessed word.

Features of Capability-Based Address

Capability based addressing has two salient features.

1. Relocability 2. Sharing

1. **Relocability:** An object can be relocated anywhere in the main memory. Without making any change to the capabilities that refer to it (for every relocation, only the base field of object needs be changed in object table).

2. **Sharing:** Sharing is made easy as several programs can share the same object (program or data with different names (object descriptor) for object.

Note: This type of sharing and relocability is achieved by introducing a level of indirection (via the object table) in addressing the tables (objects). The object descriptor in a capability contains the address of the object.

If there is a separate object table for each process or subject, then the resolution of the object descriptor is done in the context of process. If there is a global object table, then the resolution of an object descriptor is done in single global context.

Advantages of Capabilities

The capability based protection system has three. main advantages efficiency, simplicity and flexibility. It is efficient because the validation of an access can be easily tested; an access by a subject is implicity valid if it has the capability. It is simple due to the natural correspondence between the structural preperties of capabilities and the semantic properties of addressing variables. It is flexible because a capability system allows users to define certain parameters. For example, a users can decide which of his addresses contain capabilities. Also, a user can define any data structure with an arbitrary pattern of access authorization.

Drawbacks of Capabilities

Control propagation: When a subject passes a copy of a capability for an object to another subject, the second subject can pass copies of the capabilities to many other subjects without the first subject’s knowledge. The propagation of a capability can be controlled by adding bit called the copy bit, in a capability that indicates whether the holder of the capability has permission to copy (and distribute) the capability. The propagation of a capability can be prevented by the setting this bit to OFF when providing a copy of capability to the other users.

Review: Another fundamental problem with capabilities is that the determination of all the subjects who have access to an object (called the review of access) is difficult. This is because the determination of who all access to an object involves searching all the programs and data structures for the copies of corresponding capabilities. This requires a substantial amount of processing. Note, however that the review of access become simpler in the systems with the partitioned approach because now one needs to search only the segments that store capabilities (search space may be substantially reduced).

Revocation of access rights: Revocation of access rights is difficult because once a subject X has given a capability for an object to some other. subject Y. Subject Y can store the capability in a place not known to X, or Y itself may make copies of-the capabilities and pass it to its friend without any knowledge of X. 'lb revoke access rights from some subjects, either 'X must review all the accesses to that object and delete the undesired, ones or delete the object and create another copy of the object and give permission to only desired subjects. The simplest way to revoke accesses is to destroy the object, which will prevent all the undesired subjects from acessing it.

Garbage collection: When all the capabilities for an object disappear from the system, the object is left inaccesible to user and becomes garbage. This is called the garbage collection or the lost object problem. One solution to this problem is to have the creator of an object or the system keep a count of the number of copies to each capability and recover the space taken by an object when its capabilities count become zero.

The Access Control List Method

The access control list method corresponds to the column wise decomposition of the access matrix. Each object O is assigned a list of pairs (S, P[S, O]) for all subjects s that are allowed to access the object O. The access list assigned to object O corresponds to all access rights contained in the column for object O in the access matrix. A schematic diagram of an access control list is shown in figure 5.7.

Kritanshu	Read, write, execute
Shatakshi	Read
Ashish	Write
Aakash	Execute
|	|
|	|
|	|
Mandit	Read, write

Figure 5.7. The schematic diagram of an access control list

When a subject S requests access α to object O, it is executed in the following manner.

- The system searches the access control list of o to find out if an entry (S, Φ)) exist for subject S.

- If an entry (S, Φ)) exists for subjects S, then the system checks to see if the requested access is permitted (i.e., α e Φ))

- If requested access is permitted then the request is executed. Otherwise, an appropriate exception is raised.

Clearly, the execution efficiency of the access control list method is poor because an access control list must be searched for every access to a protected object.

Major -feature of the access control list method include. Easy Revocation: Revocation of access rights from a subject is very simple, fast and efficient. It can be achieved by simply removing the subject's entry from the objects access control list.

Easy Review of an Access: It can be easily determined what subjects have accessed rights to an object by directly examining the access control list of that object. However, it is difficult to determine what objects a - subject has access to.

The Lock-Key Method

In the lock-key method, every subject has a capability list that contains tuples of the form (O, K), indicating that the subject can access object O using key K. Every object has an access control list that contains tuples of the form (/, ¥), called a lock entry, indicating that any subject which can open the lock / can access this object in modes contained in the set ¥.

When a subject makes the request to access object 'o' in mode α; the system executes it in the following manner:

• The system locates the tuple (O, K) in the capability list of the subject. If no such tuple is found, the access is not permitted.

• Otherwise, the access is permitted only if there exists a lock entry (/, ¥) in the access control list of the object O. such that K=/ & α e ¥.

In this method, the revocation of access rights is easy. To revoke the access rights of a subjects to an object, simply delete the lock entry corresponding to the key of the subject. Their is no major advantage obtained from the use of capabilities except that capability-based addressing can be used. The access control list of the object must still be searched for every access. For the revocation of access rights of a subject to an object. The lock corresponding to the subject must be known. Thus, the correspondence between locks and, subjects must be known.

The IBM/360's storage keys protection method is similar to the lock-key method. The ASAP file system uses the lock-key method for protection.

5.5 SECURITY PROBLEM AUTHENTICATION

Authentication deals with the problem of verifying the identity of a user before permitting access to the requested resource. That is, an authentication mechanism prohibits the use of the system by unauthorized .users by verifying the identity of a user making a request.

Authentication basically involves identification and verification. Identification is the process of claiming a certain identity by a user, while verification is the process of verifying the user's claimed identity. Thus the correctness of an authentication process relies heavily on the verification procedure employed.

In distributed systems, authentication means verifying the identity of communicating entities to each other".

The main types of authentication normally needed in a distributed system are as follows:

1. User login authentication: It deals with verifying the identity of a User by the system at the time of login.
2. One-way authentication of communicating entities:. It deals with verifying the identity of one of the two communicating entities by the other entity.
3. Two-way authentication of communicating entities: It deals with mutual authentication, whereby both communicating entities verify each others identity.

5.6 PASSWORD

The password is the most commonly used scheme which is also easy to implement. The operating system associates a password along with the user-name of each user, and stores it after encryption in a system file (e.g.,/etc.,/ password file 'in UNIX). When a user wants to logon to the system the operating system demands that the user keys in both his username and password. The operating system then encrypts this keyed in password using the same encryption technique and then matches it with the one stored in the system file. It allows to login only after they tally. This technique is quite popular as it does not require any extra hardware. But then it provides only limited protection.

The password scheme is very simple to implement, but it is just as easy to break. All you need to do is to know somebody else's password. In order to counter this thread, the designers of the password systems make use of a number of technique. Some of these are listed as follows:

Choke of the Password

Three methods can be used in choosing a password. Either the operating system itself selects the password for the user or the system administrator decides the password for each user or allow a user to select it.

(i) The system selected password may not be easy to guess for an intruder, but then the problem is that the user himself may not remember it. As the password is not chosen by the user, it may not have any significance for him, even remotely.

(ii) System administrator choosing the password is not a particularly good idea as more than one person would know about each password thereby making the system more vulnerable.

(iii) If user selects a password, the user can remember it easily. While selecting a password most users make use of names, family names, names of cities or some important word directly or with reverse spellings, or they may use some other simple algorithm. Hence, knowing the user whose files are to be attacked, an intruder can guess the possible password easily.

Password Length

The password length plays an important role in the effectiveness of the password scheme, if the password is short, it is easy to remember. But then a short password is easy to break. If a password is too long, it is difficult to penetrate, but it is also difficult to remember by legitimate users. Therefore a tradeoff is necessary, it is normally kept 6-8 character long.

Force Password Change

In this scheme, the operating system forces the user to change the password at a regular frequency. This is done so that even if an intruder has found out a password, it would not valid for a long time, thus reducing the window of damage.

One Time Password

An extreme case of changing the password at a regular frequency is to force the user: to use ,a different password each time. For each user, a list of password can be prepared by the operating system or the system administrator, and stored in the system. User keeps a same copy with him. For the first time, the first password in the list has to be keyed in for a successful login. The user is forced to key in the next password from the list each time he logs onto the system. The operating system as well as the, user have to keep track of the' next password that is' valid, when the list is exhausted, a new list can be generated or one could start from the beginning (wrap-around). This scheme has only one major drawback. It is not exactly a very safe strategy to lose this list.

Disable Users

Many operating systems allow a user to try a few guesses (typically 3). After these unsuccessful attempts, the operating system logs the user off. Some operating systems go to the extent of disabling the user from the system itself.

This means that no more login is allowed. The user then has to contact the system administrator to reinstate the system access.

The drawback of this scheme is that it can be misused. If a person knows all the usernames, he can try out all of them, including that of the system administrator, one by one, to knock the whole system out. It is for this reason that some operating systems disable the user account only if it is not the last account-belonging to the system administrator.

<u>5.7 PROGRAM THREATS</u>

Trojan Horse

This is a program which oppears to be harmless, but has a piece of code which is very harmful. Its name is derived from Greek mythology. As is well known, there was a war between the Greeks and Trojans. The Greek army appeared to be defeated in a bloody war due to the superior Trojan tactics. Finally, in a bid to enter troy clandestinely, the Greeks created a large hellow wooden, horse and parked it at the gates of troy. The Trojans interpreted it as a symbol of peace and allowed it inside their city. At night, the Greek soldiers hiding inside the horse got out and set troy on fire.

The 'Trojan Horse', here, is meant to fool a common user. Hence, all the rogue software delivered to the users normally is in the form of the `Trojan Horse'.

A Trojan Horse program is a program that consists of clandestine code to do nasty things in addition to its usual function but appears to be benign. It' is often offered, as a gift or sometimes for a nominal price to present suspicion. A user normally accepts it into his or her system because of the useful function performed' by' it. However, once inside the user's computer, code hidden in the program becomes active and either executes malicious acts or creates a way of subcurting system Security so that unauthorized personeel can, gain access to the system resources. Note that a Trojan Horse attack may either be passive or active -depending on the activities performed by the clandestine code. For example, if the clandes time code simply steals information, then it is of the passive type. But if it does something more harmful like destroying/corrupting files, then it is of active type.

Purpose of Trojan Horse

The main purpose of the Trojan Horse is to reveal confidential ';.information about a computer or network to an attacker. When the user enters the user id and password, the Trojan Horse captures these details and sends there information to the attacker.

The user who had entered the login id and password does not have any knowledge of this attack.

Trap Doors

"Trap Door (in programs of computers) an international hole in the software".

Sometimes, software designers, may 'want to modify their programs after their installation or even after they have gone in production. To assist them in this task, the programmers leave some secret entry points which do not require authorization to access, certain objects.

Essentially they bypass certain validation checks. Only the software designers know how to make use of these shortcuts. These are called trap doors.

At times, such shortcuts may be necessary for coping with emergency situations, but then these trap doors can also be abused by some others to penetrate into the system.

5.8 SYSTEM THREATS

Worms

A computer worm is a full program by itself. It spreads to other computers over a network, bit while doing this it consumes the network resources to a very large extent. In fact, it can in fact, potentially bring the entire network to a grinding halt.

This worm program has two types of code

• Boot strap code

• Code forming the main body of the worm.

The boot strap code was compiled and executed on the system under attack. When executed, the boot strap code established a communications link between its new host and the machine from which it came, copied the main body of the worm to the new host, and executed it.

Once installed on a new host, 'the worm's first few actions were to hide its existence.

For this, it unlinked the binary version of itself; killed its 'parent process, read its files into memory and encrypted them, and deleted the file created during its entry into the system. After finishing its hiding operations, the worm's next job was to look into its host's routing tables to collect information about other hosts to which its current host was connected.

Thus, a worm is similar to a virus in concept, but its implementation is different. A worm does not modify a program, besides its replicates Itself again and again. This replication grows very fast. The performance computer and the network on which the worm resides becomes very slow, and finally it becomes in a halt condition. Thus a worm does not perform any destructive operation. It only consumes system resources to bring it down.

Safeguards against worms

There are two major safeguards against computer worms:

(i) Prevent its creation: This can be achieved by having very strong security and protection policies and mechanism.

(ii) Prevent its spreading: This can be done by introducing various checkpoints in the communications system. You can disallow the transfer of executable film over a network. But then, this may prove to be an operational hindrance. An option could be to force the, user or an operator to "sanction" such a transfer. Many corporate gateways to public networks employ this technique.

Viruses

Virus is a piece of program code that attaches itself to a legitimate program code. It runs when the legitimate program code. It runs when the legitimate program runs. Virus can affect other programs in that computer or in other computers but in the same network. For example, a virus could automatically execute at a particular date or at a particular time. The damage to the computer caused by virus can be repaired by using good backup procedure.

Working of Virus

A typical virus works as follows:

The intruder writes a new program that perform some interesting or useful function and attaches the virus to it in such a way that when the program executed the viral code also gets executed. The .intruder how uploads this infected program to a public bulletin board system or ,sends it by mail to other users of the system or offers it for free or for a nominal charge on floppy disks. Now if anyone uses the infected program, its viral code gets executed when the viral code of the infected program executes, it randomly selects and executable file on the hard disk and checks to see if it is already infected. Most viruses include a string of, characters that acts as a marker showing that the program has been infected. If the selected file is already infected, the virus selects another executable file. When an uninfected program is found, the virus infects it by attaching a copy of itself to the end of that program and replacing the first instruction of the program with a jump to the viral code. When the viral code is finished executing, it executes the instruction that has previously been first and then jump to the second instruction so now the program performs its intended function.

Infection Methods

There are five well known methods by which a virus can in feet other programs. These are discussed below:

1. **Append:** In this method, the viral code appends itself to the unaffected program.
2. **Replace:** In this case the viral code replaces the original executable program completely or partially.
3. **Insert:** In this case viral code is inserted in the body of an executable program to carry out some funny (!) or undesirable actions.
4. **Delete:** In this case, the viral code deletes some code from executable program.
5. **Redirect:** This is an advanced approach employed by the authors of sophisticated viruses. The normal control flow of a program is changed to executed some other (normally viral) code which could exist as an appended portion of an otherwise normal program. This mode is quite common, and therefore needs to be studied in details.

Virus prevention:

The best way to tackle virus problem is to prevent it, as there is no good cure available after the infection:

(a) One of the safest ways is to buy official, legal copies of software from reliable stars or sources (although even such copies have been reported to contain viruses at times).
(b) Refusing to accept software in unsealed packages or from untrusted sources.
(c) Avoiding borrowing program from someone whose security standards are less rigrous than one's own.
(d) Avoiding uploading of free software froth public domain, bulletien boards, and program send by e-mail.

Although viruses and worms both replicate and spread themselves, the two differ in the following aspects:

1 A virus is a program fragment whereas a worm is a complete', Program in itself.
2 Since virus is a program fragment, it does not exit independently. It resides in a host program, runs only when the host program runs and depends on the host program for its existence. On the other hand, a Worm can, exist and execute independently.
3 A virus spreads from one program to another whereas a worm spreads from one computer to another in a network.

Venial of Service (DOS)

Denial of Service prevents legitimate users from accessing some services for which they are eligible, for example, an unauthorized user might send many login requests to the server one after the other very quickly to flood the network.

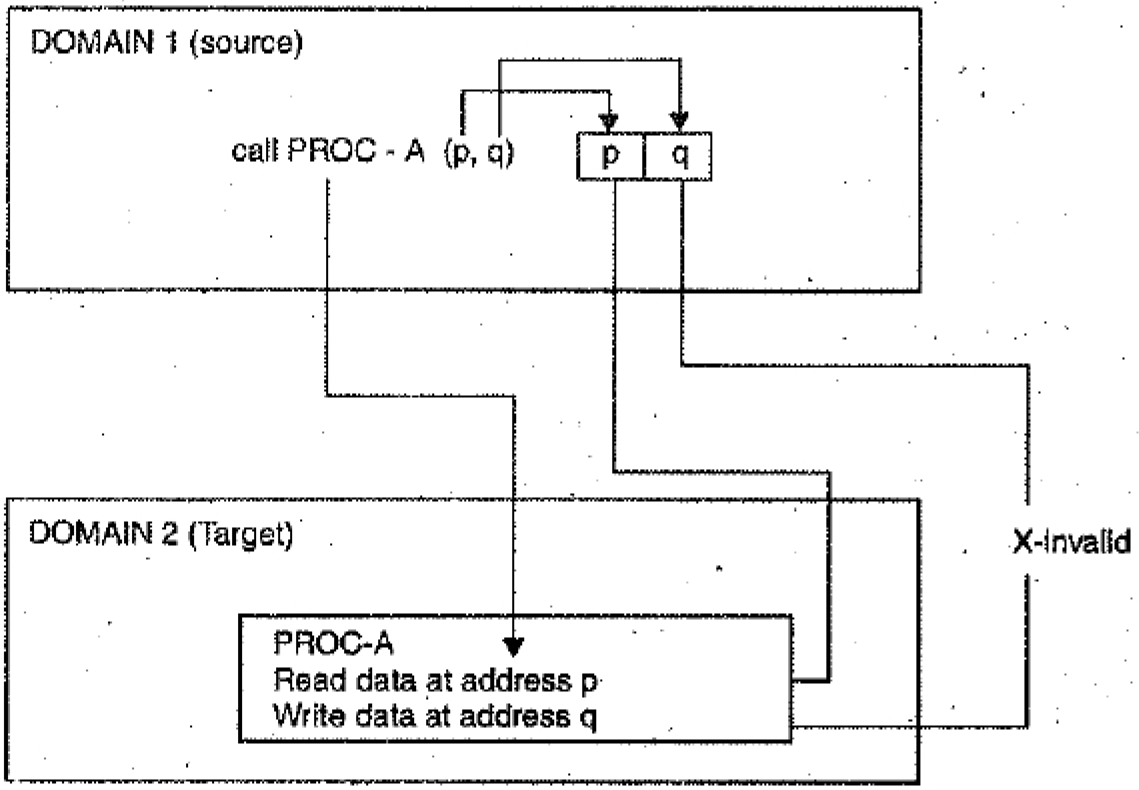
Now the server denies other legitimate users to access to the network. The parameters can be passed either directly (by value) or indirectly reference). In call by value method, a user program passes the parameter values themselves. For example, it can load certain CPU 4egisters with these values which are then picked up by the called the routine. If the called routine does not have access to those registers, it may result in the denial of this service. In call by reference, the actual values of the parameters may not be passed but the pointers or addresses of the locations where the actual values of the parameters 'are stored may be passed. The denial of service can place in this case too.

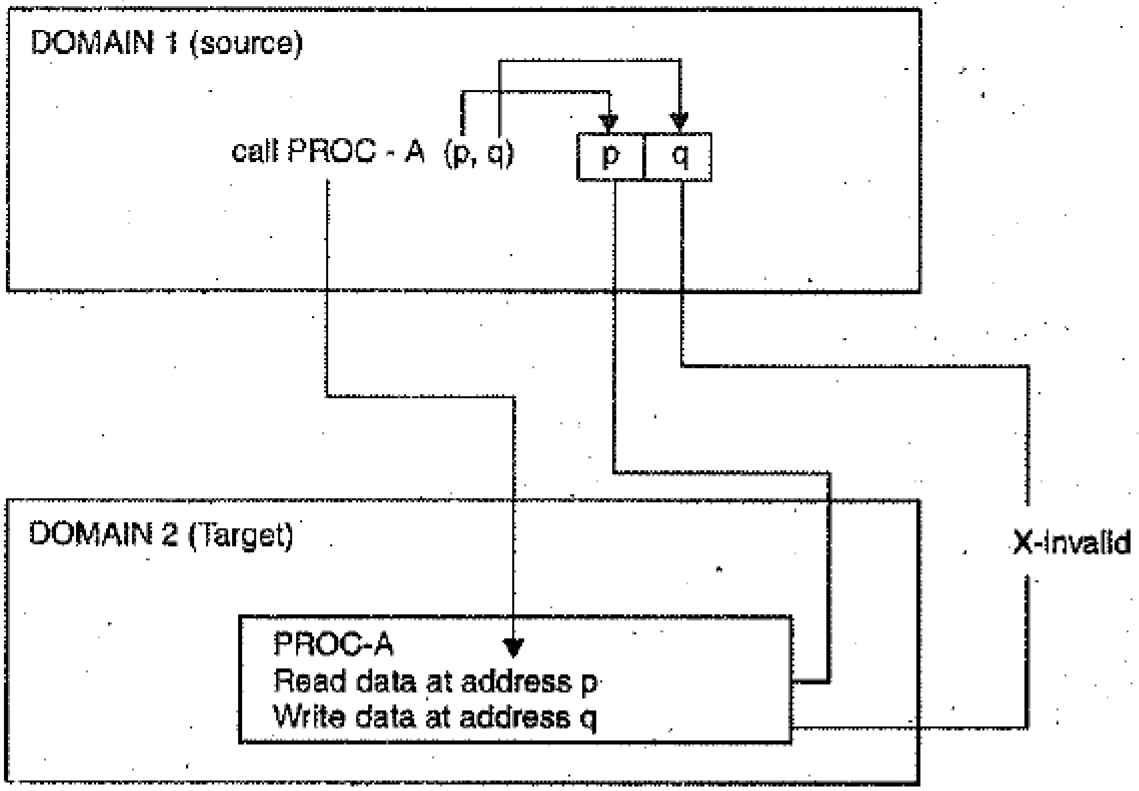
For instance, a call may pass parameters, referencing objects that cannot be accessed by the called routine (called 'target domain'), through the calling routine (called, 'source domain') has the access rights to those. Figure 5:8 depicts this scenario.

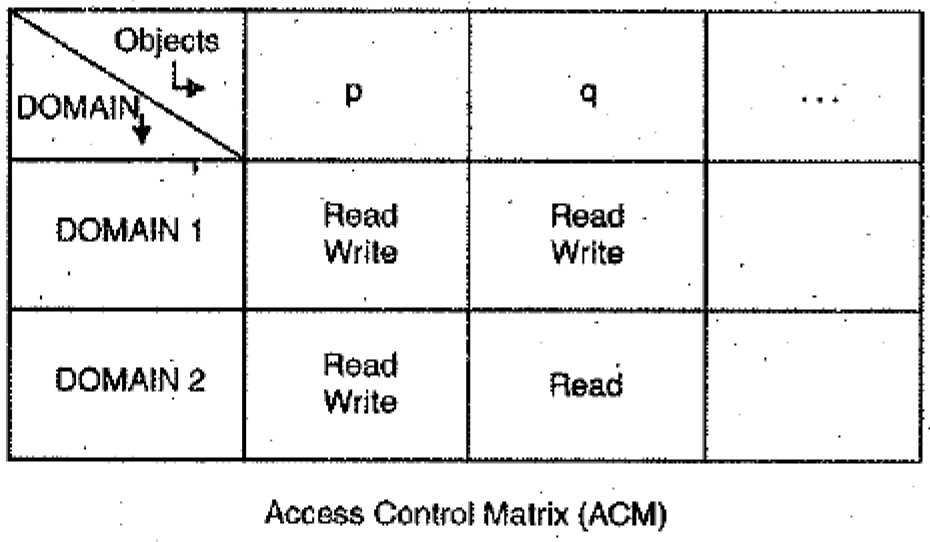
The figure 5.8 shows the source and target domain: DOMAIN 1 and DOMAIN 2 respectively. DOMAIN 1 contains a call to the procedure PROC-A which has parameters p and q which are only the addresses. DOMAIN 2 is supposed to execute this procedure by reading the data at an address p and then writing it at an address q. Note that what are passed are not the data items but the pointers to the data items.

The figure also shows the 'Access Control Matrix (ACM), for these two domains. It shows that DOMAIN 1 has both Read and Write accesses to both the addresses p and q. However, DOMAIN 2 has Read and Write access only to p. It has only Read access to q. What will happen this case?

A call will be made as usual, but at a time of execution, when the data read at address p has to be written at address q, the access will be denied. This is because DOMAIN 2 does not have a write access for q. Thus, the denial of the service results.


DOMAIN 1 (source)
call PROC - A (p, q)
p
q
DOMAIN 2 (Target)
PROC-A
Read data at address p
Write data at address q
X-Invalid


DOMAIN 1 (source)
call PROC - A (p, q)
p
q
DOMAIN 2 (Target)
PROC-A
Read data at address p
Write data at address q
X-Invalid



Objects → / DOMAIN ↓	p	q	...
DOMAIN 1	Read Write	Read Write	
DOMAIN 2	Read Write	Read	

Access Control Matrix (ACM)

Figure 5.8. Denial of Service

5.9 ENCRYPTION

Encryption is one of the most important tools in protection, security; and authentication. The process involves two things: Encryption which. means changing the original data to some other form so that nobody can make out anything about it, and Decryption which. means recovering the data in the original form. Figure 5.9 depicts this process. The figure 'a' shows A and B as the source and destination nodes.

The data by encryption is called plaintext and the data after encryption is called ciphertext.

Encryption Algorithm and the Key

Various algorithms are used for encryption and decryption. There are two basic methods (ciphers) used for encryption. These are :

• Transposition Ciphers

• Substitution Ciphers

We will illustrate the difference between two methods by taking an example of a message consisting of text full of characters or alphabets.

In the transposition ciphers, the letters in the message are not changed but their order is changed.

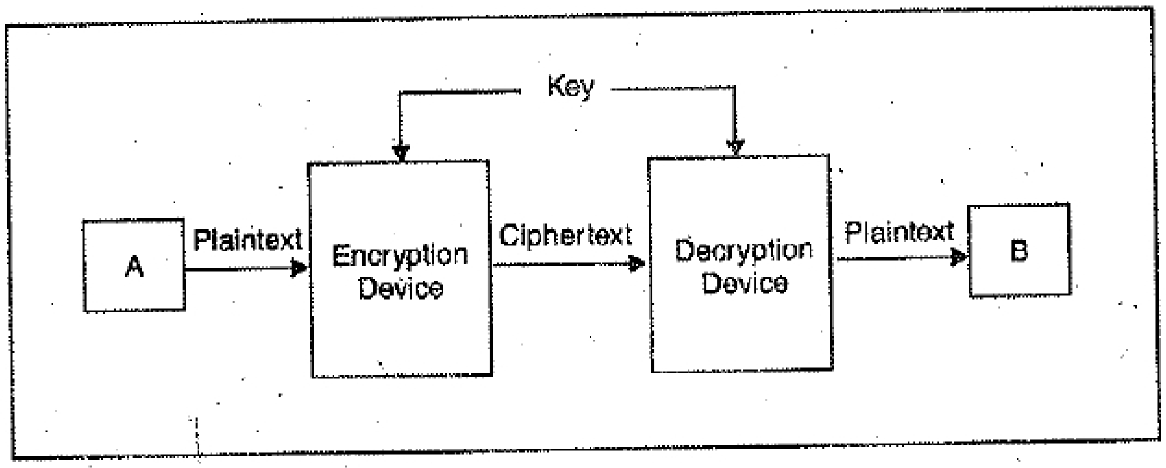


Figure 5.0 Conventional encryption

A simple example of this type could be a message sent in the reverse order. "I am fine" would become "enif ma I".

In the Substitution Ciphers' work by sending a set of characters different from the original ones. For instance, you could send the next alphabet. "I am fine" would become "J bn gj of".

Encryption would involve adding, say "00011001" to every 8 bits in the plaintext, and decryption would mean subtracting the same from the cyphertext. This Encryption/decryption can be achieved by the adder hardware circuits. This bits stream "00011001" would then become a key.

Figure 5.9 depicts conventional encryption. In this case, there is only one key and it is known to both A and B. It is not known to anybody else (though the algorithm may very well be known).

In the conventional encryption scheme, both parties A and B have to agree upon a key Somebody (A, B or a third party) has to decide upon this common key, get the concurrence of all concerned parties, and then initiate the communication. This process is' called key distribution. In the conventional encryption method depicted in figure 'a', each pair of nodes needs a unique key. Thus, if there are n. nodes, you will need n (n-1)/2 keys.

For 100 nodes this .number is 100 x 99/2 = 4950. An alternative method is called Public Key Encryption. This is based on a. principle that the keys used for encryption and decryption should not be the same. In fact, there are possible algorithms and hardware, in which one key K1' is used for encryption arid another key K2 is used for decrypting.: In fact, only K2 can decrypt the message and NOT K1 This is important because one of these keys. is publicly known (hence, the name) and another one is known only to the person who does decryption thereby ensuring that the information cannot leak out. There is an interesting property of the public key encryption system. Even if you interchange the keys, the encryption/decryption is still possible. For instance, you can use K2 for encryption and K1 for decryption. The scheme will still work.

Therefore; in this scheme, for each user, two keys are defined. One is called 'private key' and the other 'public key'. Each user node keep its private key secret, but publishes the public key to the central key database. This database maintains various public keys for different users at different nodes. Figure 5.10 depicts how this scheme works.

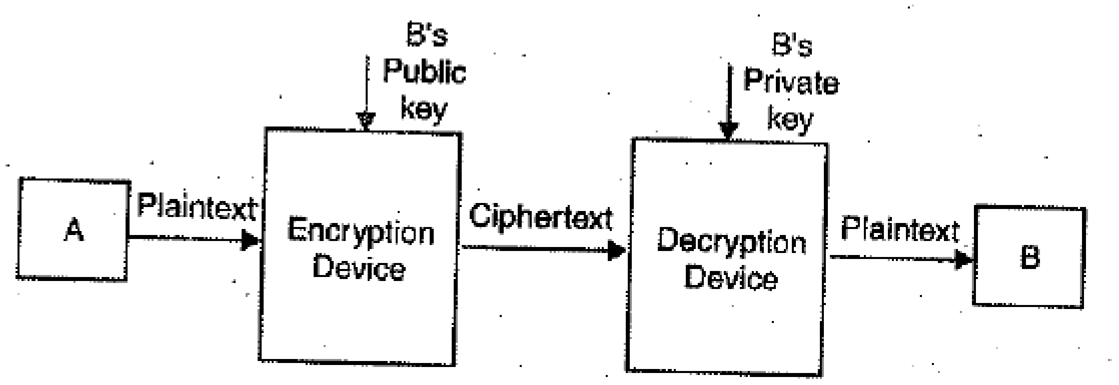


Figure 5.10 Public key encryption

The following steps are taken to achieve the encryption/decryption:

(i) When A wants to send the data to B, A consults the database for public keys and accesses B's public key. Because it is a public key, it is accessible to A even if it is B's public key.
(ii) A encrypts the message using B's public key and produces the ciphertext.
(iii) The ciphertext now traverses to node B.
(iv) Node B decrypts the message using B's private key, which is known to it anyway.

The protection, in this scheme is very good, because only B can decrypt that ciphertext message and nobody else can, even if the message were tapped by an intruder. However, in this scheme, node B could not confirm that the message has originated only from A i.e., no. authentication is possible in this scheme. B's public key is known to everyone. Anybody could have been sent that message instead of A. Hence, B has no guarantee that only A has 'sent the message.

5.10 COMPUTER SECURITY CLASSIFICATION

Security is a broad topic and covers a multitude of sins. In its simplest form, it is concerned with making- sure that nosy people cannot read, or worse yet modify messages intended for other recipients. It is concerned with people trying to access remote services that they are not authorized to use. It also deals with how to tell whether that message purportedly from the IRS saying: "Pay by Friday or else" is really from the IRS or from the Mafia. Security also deals with the problems of legitimate messages being captured and replayed and with people trying to deny that they sent certain messages.

Most security problems are intentionally caused by malicious people trying to gain some benefit or harm someone. A few of the most common perpetrators are listed in Table 5.2. It should be clear from this list that making a network secure involves a lot more than just keeping it free of programming errors. It involves outsmarting often intelligent, dedicated and sometimes well-funded adversaries. It should also be clear that measures that will stop casual adversaries will have little impact on the serious ones.

Table 5.2. Some People who Cause Security Problems and why

Adversary	Goal
Student	To have fun snooping on people's e-mail
Hacker	To test out someone's security system; steal data
Sales rep	To claim to represent all of Europe, not just Andorra
Businessman	To discover a competitor's strategic marketing plan
Ex-employee	To get revenge for being fired
Accountant	To embezzle money from a company
Stockbroker	To deny a promise made to a customer by e-mail
Conman	To steal credit card numbers for sale
Spy	To learn an enemy's military strength
Terrorist	To steal germ warfare secrets

Network security problems can be divided roughly into four intertwined areas: secrecy, authentication, non-repudiation, and integrity control. Secrecy has to do with keeping information out of the hands of unauthorized users. This is what usually comes to mind when people think about network. Security Authentication deals with determining whom you are talking to before revealing sensitive information or entering into a business deal Non repudiation deals with signatures: How do you prove that your customer- really placed an electronic order for ten million left-handed doohickeys at 89 cents each when he later claims the price was 69 cents? Finally, how can you be sure that message you received was really the one sent and not something that malicious adversary modified in transit or concocted?

All these issues (secrecy, authentication, non repudiation, and integrity control) occur in traditional systems, too, but with some significant differences. Secrecy and integrity are achieved by using registered mail and locking documents up. Robbing the mail train is harder than .it was Jesse James' day.

Also, people can usually, tell the difference between an original paper document and a photocopy, and if often matters to them. As a test make a photocopy of a valid check, Try cashing the original check at your bank on Monday. Now try cashing the photocopy of the check Tuesday Observe the difference in the bank's behaviour. With electronic checks, the original and the copy are indistinguishable. It may take while for banks to get used to this.

People authenticate other people by recognizing their faces, voices, and handwriting. Proof of signing is handled by signatures on letterhead paper, raised seals, and so on. Tampering can usually be detected by handwriting, paper, and ink experts. None of these options are available electronically. Clearly, other solutions are needed.

Before getting into the solutions themselves, it is worth spending a few Moments considering wherein the protocol stack network security belongs. There is probably no one single place. Every layer has something to contribute. In the physical layer, wiretapping can be foiled by enclosing transmission lines in sealed tubes containing argon gas at high pressure. Any attempt to drill into a tube will release some gas, reducing the pressure and triggering an alarm. Some military systems use this technique.

In the data link layer, packets on a point-to-point line can be encoded. as they leave one machine and decoded as they enter another. All details can be handled in the data link layer, with higher layers oblivious to what is going on. This solution breaks down when packets have to transverse multiple routers, however, because packets have to be decrypted at each router, leaving them vulnerable to attacks from within the router. Also, it does not allow some sessions to be protected (e.g., those involving online purchases by credit card) and others not. Nevertheless, link encryption, as this method is called, can be added to any network easily and is often useful.

In the network layer, .firewalls can be installed to keep packets in or : keep packets out In the transport layer, entire connections can be 'encrypted, end to end, that is, process to process.. Although these solutions help with secrecy issues and many people are working hard to improve them, none of them solve the authentication or non repudiation problem in a sufficiently general way. To tackle these problems, the solutions must be in the' application layer.

Security Attacks

"Security attack is any action that compromises the security of "information" owned by an organization."

Attack: An attempted cryptanalysis is termed as an attack.

Classification of Security Attacks

The security attacks are classified into two categories:

1. Active attacks

2. .Passive attacks

1. Active Attacks

An active attack attempts to alter the system resources or affect their operation.

Or

Active attack involves some modification of the data stream or the creation of a false. stream.

Active attacks are subdivided into four categories:

1. **Masquerade:** A masquerade attack takes place when one entity pretends to be a different entity. It usually includes one of the other forms of active attacks.

2. **Replay:** It involves the passive capture of the data unit and its subsequent retransmission to produce an unauthorized effect.

3. **Modification of messages:** It simply means, that some portion of the original message is altered, or that messages are delayed or reordered to .produce an unauthorized effect.

4. **Denial of Services:** This prevents or inhibits the normal use or management of communication facilities. This attack may have a specific target: an entity may suppress all messages directed to a particular destination.

2. Passive Attacks

"A passive attack, attempts to learn or make use 'of information from the systent but does not affect system resources."

Passive attacks are subdivided into two categories.

1. **Release of message contents:** The release of message contents is easily understood. A telephone conversation, an e-mail message and a transferred file may contain sensitive or confidential information.

2. **Traffic analysis:** Suppose we had a way of masking the contents of a. message or other information traffic so that opponents, even if they captured the message, could not extract the information from the message. The common technique for masking contents is encryption.

   Passive attacks are very difficult to detect because they do not involve any alteration of the data, these can be prevented by means of encryption. Active attacks are quite difficult to prevent absolutely because to do so requires physical protection of all communication facilities and paths at all times.

Security Services

"Security services are the services that enhance the security of the data processing system and the information transfers of an organization". The services are intended to counter the security attacks and they make use of one or more security mechanisms to provide the service. Classification of Security Services

1. Authentication

If the message sent by the sender is accurately received by the receiver then it is termed as authentication. Authentication is managed on the network by differ ways:

(i) By sending their PIN number,
(ii) By using their ATM card, electronic key etc.
(iii) By sending finger prints, DNA patterns etc.

Two specific authentication services are defined in the standard:

(a) Peer entity authentication: Used in association with a logical connection to provide confidence in the identity of the entities connected.

(b) Data-origin authentication: In a connectionless transfer, provides assurance that the source of received data is as claimed.

2. Data Integrity

If a received communication is exactly what the sender party has send (i.e., contain no modification, insertion, deletion or replay etc.) then it, is said to have integrity. Therefore, the parties must be able to assure themselves that the message they received is exactly what the other party has sent.

(a) Connection Integrity with Recovery: Provides for the integrity of all user data on a connection and detects any modification, insertion, deletion or replay etc., of any data - within an entire data sequence, with recovery attempted.

(b) Connection Integrity without Recovery: As connection integrity with recovery but provides only detection without recovery.

3. Non-repudiation

It prevents either sender or receiver from denying a transmitted message. Thus when a message is sent, the receiver can prove that the message was' in fact sent by the alleged sender. Similarly when a' message is received, the sender can prove that the message was in fact, received by the alleged receiver.

(a) Nora-repudiation, origin: Proof that the message was sent by the specified party.

(b) Non-repudiation, destination: Proof that the message was received by the specified party.

Example:

Suppose customer places the purchase order be a company for buying shares. If the price of a shares goes down on the same day, the sender makes a dispute that he has never placed such a purchase order. Similarly, if the price of a share goes up during the same day, the company may rise a dispute that it has never received such a purchase order.

4. Access Control

Access control is the prevention of unauthorized use of a resource i.e., this service controls who can have access to a resource, under what conditions access can occur and what those accessing the resource are allowed to do.

Or

Access control is the ability to limit, and control the access to host systems and applications via communication links.

5. Data Confidentiality

Data confidentiality is a service, which provides protection of data from unauthorized disclosure. Confidentiality is the protection of transmitted data from passive attacks.

Four categories of data confidentiality are:

(a) Connection confidentiality: The protection of all user data on a connection.

(b) Connection confidentiality: The protection of all user data in a single data block.

(c) Selective-field confidentiality: The confidentiality of selected fields within the user data on a connection or in a single data block.

(d) Traffic flow confidentiality: The protection of the information that might be derived from observation of traffic flows, i.e., the attacker should not be able to observe the source and destination frequency, length or other characteristics of the traffic.

Security Mechanisms

Security mechanisms are that which are designed to detect, prevent or recover from a security attack:

Mechanisms are divided into those that are implemented in a specific protocol layer and to those that are .not" specific to any particular protocol layer or security service

(a) Specific Security Mechanisms

May be incorporated into the appropriate protocol layer in order to provide some of the OSI security services

1. Encipherment: The use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of the data depend on an algorithm and zero or more encryption keys.
2. Digital Signature: Data appended to, or a cryptography transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (for example, by the recipient).
3. Access Control: A variety of mechanisms that enforce access rights to resources.
4. Data Integrity: A variety of mechanisms used to assure the integrity of a data unit or stream of data units.
5. Authentication Exchange: A mechanism intended to ensure the identity of an entity by means of 'information exchange.
6. Traffic Padding: The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.
7. Routing Control: Enables selection of particular physically secure routes for certain data and allows routing changes, especially when a breach of security is suspected.
8. Notarization: The use of a 'trusted third party to assure certain properties of a data exchange.

(b) Pervasive Security Mechanisms:

Mechanisms that are not specific to any particular OSI security service or protocol layer.

1. Trusted Functionality: That which, is perceived to be correct with respect to some criteria. (for example; as established by a security policy)
2. Security Label: The marking bound to a. resource. (which may be a data unit) that names of designates the security attributes of that resource.
3. Event Detection: Detection of security—relevant events.
4. Security Audit Trail: Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.
5. Security Recovery: Deal with requests from mechanisms, such as event handling and management functions, and takes recovery actions.

5.11 MODEL OF WINDOWS-NT

Windows-NT

Windows-NT is a. full-fledged operating system. Windows-NT is portable operating system like UNIX and will be available on other hardware platforms. The hardware currently supported by Windows-NT includes Intil's x 86 family, silicon Graphics MIPS Rs. 4000 RISC and Digital equipment Corp's Alpha Processor.

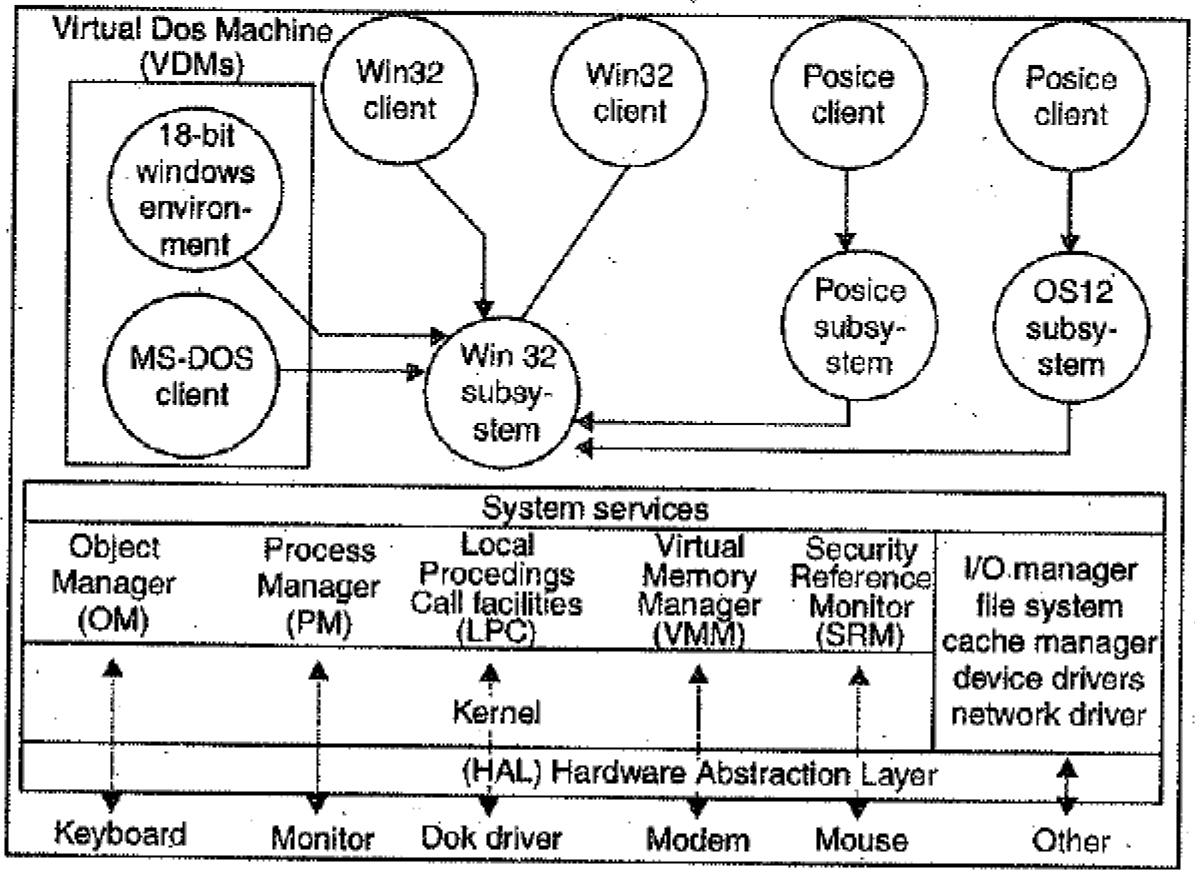
Windows-NT is a multiuser and Multitasking operating system. It has all the features of modern operating system. These include virtual memory management, threads, preemptive multitasking, symmetric multiprocessing and built in networking, in a modular and portable design.

Diagrammatic representation of organizational structure of WINDOWS-NT is shown in figure 5.11.

Windows-NT is a multiuser, multitasking and multithreading operating system. To a MS-Windows user, this feature Will be truly visible fast response he can, get in his applications, when multiple applications are running.

Windows-NT multitasking is preemptive like UNIX. Tasks are scheduled by the operating system, giving a time slice to each task and when its time slice is over, the next one is started. The previously running task is resumed when its turn come again. Applications can't avoid getting preempted and thus, they cannot tie up the CPU for a very long time. This is not the case in MS-Windows.

Multi threading is 'another new feature which differentiates Windows-NT from MS-Windows. A task can be subdivided into smaller un-its called threads. Each thread has an independent flow of control wit-hin a task. Different threads can be scheduled and executed independently.



Virtual Dos Machine (VDMs)
Win32 client
Win32 client
Posice client
Posice client
18-bit windows environ-ment
MS-DOS client
Win 32 subsy-stem
Posice subsy-stem
OS12 subsy-stem
System services
Object Manager (OM)
Process Manager (PM)
Local Procedings Call facilities (LPC)
Virtual Memory Manager (VMM)
Security Reference Monitor (SRM)
I/O manager file system cache manager device drivers network driver
Kernel
(HAL) Hardware Abstraction Layer
Keyboard
Monitor
Dok driver
Modem
Mouse
Other

Figure 5.11 Architecture of Windows-NT

Windows-NT supports file transfer, e-mail services and resource sharing on a network. Windows NT can interact with all existing networking system like NOVELL's netware. Sun Microsystem's NFS,. etc. Other communication facilities include support for sockets, named pipes, messages and remote procedure calls.

Windows-NT doesn't use the File Allocation Table (FAT) file system of MS-Dos. It has a high new performance file system called New Technology File System (NTFS). In NTFS, file names can be upto 256 character long. NTFS also implements fault tolerance, security and has support for very large files. Disk access is speeded up because of NTFS. However, to keep backward capabilities with MS-Dos, Windows- NT can support the FAT file system in a separate partition. Floppy drives will continues to have the FAT file system..

Windows NT has a modular architecture, divided into different subsystem and layers. At the lowest layer is the Hardware Abstraction Layer (HAL). This layer provides all the

Hardware specific functions. This is the layer which will be different on different hardware platforms. This layer handles interrupts, Direct Memory Access (DMA) memory management at the lowest level and multiprocessor synchronization. On top of the HAL layer is the Windows-NT Kernel. The Kernel provides the basic operation system services like scheduling of threads and tasks. It supplies the basic set of objects like files, memory and processes for use by the higher layers and also scheduling of multiprocessors. On top of ' the kernel, there are six modules which together form an executive. These have a, well defined interface and one module can be replaced by another to handle the same function when upgrading the operating system and when the different functionality is needed, without affecting the other modules.

The six subsystems and their functions are listed below:

1. **Object Manager:** It creates, manages and deletes object which are processes, threads, named pipes, files, etc.
2. **Process Manager:** If creates, terminates, suspends and resume processes, tasks and threads.

Virtual Memory Manager: It manages the memory for processes. It allocates the free memory. This subsystem also handles paging and memory protection from other processes. Each process can access up to 2 GB of virtual memory.

Security Reference Monitor: If enforces security policy and keeps tracks of file access rights based on the ownership and permissions of the user. This subsystem can also be used to create alarms for any security violations and also to keep an audit trail.

Local Procedure Call Facility: It is used to pass messages between client systems and subsystems on one system. I/O Subsystem: The I/O subsystem handles the device drivers and passes data to and receive data from the devices of all the subsystems. The file system is also handled by I/O subsystem.

On top of these subsystems is the Win 32 layer which handles all the user interactions and decides the correct subsystem for each service requested by the user application.. Above this layer are protected subsystems for each application. Windows NT can run application written for MS-Dos, OS/2 and MS Windows. These application will interact with a different subsystem.

SUMMARY

• Security and protection deal with the control of unauthorized use and access to the hardware and software resources of computer system.

• Internal security deals with the access and the use of computer hardware and software information stored in computer system.

• The protection domain of a process specifies the resources that it can access and the types of operations that the process can perform on the resources.

• The access matrix model was first proposed by Lampson. It was further enhanced and refined by Graham and Denning and Harrison.

• Current objects are a finite. set of entities that access current is to be controlled. The set is denoted by ‘O'.

• Currents subjects are a finite set of entities that access current objects. The set is denoted by 'S'.

• A security policy is enforced by validating every user access for appropriate access rights.

• The capability based method corresponds to the row-wise decomposition of the access matrix.

• The access control list method corresponds to the column wise decomposition of the access matrix.

• The password is the most commonly used scheme which is also easy to implement.

• A Trojan Horse program is a program that consists of clandestine code to do nasty things in addition to its usual function but appears to be benign.

• Encryption is one of the most important tools in protection, security and authentication.

• A masquerade attack takes place when one entity pretends to be a different entity.

• Security services are the services that enhance the security of the data processing system and the information transfers of an organization.

• Access control is the prevention of unauthorized use of a resource i.e., this service controls who can' have access to a resource, under what conditions access can occur and what those accessing the resource are allowed to do.

• Access control is the ability to limit and control the access to host systems and applications via communication links.

• Data confidentiality is a service, which provides protection of data from unauthorized disclosure.

• Security mechanisms are that which are designed to detect, prevent or recover from a security attack.

REVIEW QUESTIONS

1. Discuss about the Access Matrix Model.
2. Explain the implementation of the Access Matrix.
3. What do you mean by capability? Give some advantages of capability.
4. Explain the drawbacks of capability.
5. What do you mean by virus? How it can affect the system?

FURTHER READING

1. Operating Systems: A practical Approach: Rajiv Chopra, S. Chand Publisher, 2010
2. Operating Systems: Principles and Design: Pabitra Pal Choudhury PHI Learning
3. Operating Systemes and Software Diagnostics: Ramesh Bangia and Balvir Singh, Laxmi Pub., 2007
4. Principles of Operating Systoles: Sri V. Ramesh, Laxmi Pub., 2010