KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari Post, Coimbatore - 641021
(For the candidates admitted from 2016 onwards)
DEPARTMENT OF INFORMATION TECHNOLOGY

SUBJECT : OPERATING SYSTEMS	**SEMESTER : IV**
SUBJECT CODE: 18CTU401	**CLASS : II B.Sc. (CT)**
STAFF NAME : K.VEERASAMY	**BATCH : 2018-2019**

4H – 4C

Instruction Hours / week: L: 4 T: 0 P: 0 **Marks:** Int : **40** Ext : **60** Total: **100**

End Semester Exam : 3 Hours

Course Objectives

- To understand the main components of an OS & their functions.
- To study the process management and scheduling.
- To understand various issues in Inter Process Communication (IPC) and the role of OS in IPC. To understand the concepts and implementation Memory management policies and virtual memory.
- To understand the working of an OS as a resource manager, file system manager, process manager, memory manager and I/O manager and methods used to implement the different parts of OS
- To study the need for special purpose operating system with the advent of new emerging technologies

Course Outcomes (COs)

Students will able to:

1. Describe the important computer system resources and the
2. Perform the role of operating system in their management policies and algorithms.
3. Understand the process management policies and scheduling of processes by CPU
4. Evaluate the requirement for process synchronization and coordination handled by operating system
5. Describe and analyze the memory management and its allocation policies.
6. Identify use and evaluate the storage management policies with respect to different storage management technologies.
7. Identify the need to create the special purpose operating system.

Unit I

Introduction to Operating System: Basic OS Functions-Resource Abstraction-Types of Operating Systems–Multiprogramming Systems-Batch Systems-Time Sharing Systems-Operating Systems for Personal Computers & Workstations-Process Control & Real Time Systems.

Unit II

Operating System Organization: Processor and user modes-Kernels-System Calls and System Programs. **Process Management:** System view of the process and resources- Process abstraction-Process hierarchy-Threads-Threading issues-Thread libraries-Process Scheduling-Non pre-emptive and Preemptive scheduling algorithms-Concurrent and processes-Critical Section-Semaphores-Methods for inter-process communication- Deadlocks.

Unit III

Memory Management: Physical and Virtual address space-Memory Allocation strategies – Fixed and Variable partitions-Paging-Segmentation-Virtual memory.

Unit IV

File and I/O Management: Directory structure-File operations-File Allocation methods- Device management.

Unit V

Protection and Security: Policy mechanism-Authentication-Internal aITUess Authorization.

Suggested Readings

1. A .Silberschatz, , P.B Galvin, G.Gagne (2008). Operating Systems Concepts, 8th ed.). John Wiley Publications.
2. A.S. Tanenbaum, (2007).Modern Operating Systems (3rd ed.). New Delhi: Pearson Education.
3. W. Stallings, (2008). Operating Systems, Internals & Design Principles (5th ed.). Prentice Hall of India.

Web Sites

1. www.cs.columbia.edu/~nieh/teaching/e6118_s00/
2. www.clarkson.edu/~jnm/cs644
3. pages.cs.wisc.edu/~remzi/Classes/736/Fall2002/

ESE Pattern		**Internal Pattern**	
Part A (Online)	20*1=20	Part A	20*1=20
Part B	5*2=10	Part B	3*2=06
Part C(Either or)	5*6=30	Part C	3*8=24
Total	**60 Marks**	**Total**	**50 Marks**

Faculty **HoD**

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
Eachanari (po), Coimbatore-21
DEPARTMENT OF CS, CA & IT

SUBJECT NAME : OPERATING SYSTEMS **SEMESTER: IV**
SUB.CODE :18CTU401 **CLASS : II B.Sc. (CT)**
STAFF NAME: K.VEERASAMY **BATCH: 2018-2019**

S. No	Lecture Duration (Hr)	Topics to be Covered	Support Materials
		UNIT – I	
1.	1	**Basic OS Functions**	W1
		Resource Abstraction	SR1: 4-6, W1
2.	1	Types of Operating Systems	SR2: 32-35, W1
3.	1	Multiprogramming Systems	SR1: 19, SR2:32, W1
4.	1	Batch Systems, Time Sharing Systems	SR1: 19-20, W1
5.	1	Operating Systems for Personal Computers & Workstations	W1, W2
6.	1	Process Control, Real Time Systems	SR2: 34, W1
7.	1	Recapitulation and discussion of important questions	
	Total no. of Hours		**7**
		UNIT II	
1.	1	**Operating System Organization:** Processor and user modes-Kernels, System Calls and System Programs	SR1: 20-23; SR2: 47-59,
2.	1	Core Kernel of OS	SR1:55-58
3.	1	System Calls and system programs, Process Management	SR2:47-59
4.	1	Semaphores, Critical section, Thread, Process Scheduling	SR1:234-239
5.	1	Methods for inter-process communication	SR1: 897 – 898 SR2: 115 – 126
6.	1	Deadlocks	SR1: 283 – 305 SR2: 435 – 459
7.	1	Recapitulation and discussion of important	

		questions	
	Total no. of Hours		**7**
UNIT - III			
1.	1	**Memory Management:** Physical address space, Virtual address space, Memory Allocation strategies	SR1: 315 - 320 SR2: 173 – 185 W3
2.	1	Fixed partitions	SR1: 325; W3
3.	1	Variable partitions	SR1: 326 ; W3
4.	1	Paging	SR2: 186 – 195, SR1:328 - 342
5.	1	Segmentation	SR1: 342 - 345
6.	1	Virtual memory	SR1: 357 – 393 SR2: 186 - 196
7.	1	Recapitulation and discussion of important questions	
	Total no. of Hours		**7**
UNIT - IV			
1.	1	**File and I/O Management**	SR1: 64-67, 421; W4, W5
2.	1	Directory structure	SR2: 266 - 270
3.	1	File operations	SR1: 423-425
4.	1	File operations - Example	SR2: 262 - 263
5.	1	File Allocation methods – Contiguous and linked	SR1: 471 – 476
6.	1	File allocation methods - Indexed	SR1: 476 - 479
		Device Management – Device type,	SR1: 64
7.	1	Device management - Device drivers, detection, IPC, driver interface	W7
8.	1	Recapitulation and discussion of important questions	
	Total no. of Hours		**8**
UNIT – V			
1.	1	**Protection and Security**-Introduction	SR1:629
2.	1	Policy Mechanism	SR2:620-623
3.	1	Authentication- (i)Password authentication	SR2:639
4.	1	(ii)biometric authentication	SR2:651-654
5.	1	(iii)Encrypted password and one time	SR 1:661-662

		password	
6.	1	Program threats	SR 1:663-664
7.	1	Internal Access authorization	W1
8.	1	Recapitulation and discussion of important questions	
9.	1	Previous ESE Question Paper Discussion	
10.	1	Previous ESE Question Paper Discussion	
11.	1	Previous ESE Question Paper Discussion	
	Total no. of Hours		**11**
	Total No Hours		**40**

Suggested Readings

SR1: A. Silberschatz, P.B Galvin, G. Gagne (2008). Operating Systems Concepts, 8th ed., John Wiley Publications.

SR2: A.S. Tanenbaum, (2007).Modern Operating Systems (3rd ed.). New Delhi: Pearson Education.

SR3: W. Stallings, (2008). Operating Systems, Internals & Design Principles (5th ed.). Prentice Hall of India.

Websites

W1: https://www.tutorialspoint.com/operating_system/os_overview.htm
W2: https://en.wikipedia.org/wiki/Workstation
W3: u.cs.biu.ac.il/~ariel/download/os381/ppts/os7-2_rea.ppt
W4: https://www.slideshare.net/DamianGordon1/operating-systems-file-management
W5: http://nptel.ac.in/courses/106108101/pdf/Lecture_Notes/Mod%205_LN.pdf
W6: http://wiki.osdev.org/Device_Management

Faculty **HoD**

SYLLABUS

Unit I - Introduction to Operating system: Basic Operating system functions – Resource abstraction – Types of Operating systems – Multiprogramming systems- Batch systems – Time sharing systems – Operating systems for personal computers & work stations – Process control & real time systems

1.1 Introduction

An operating system is a program that manages the computer hardware. An Operating System (OS) is an interface between a computer user and computer hardware. An OS is Resource manager. It manages the resources like CPU, Memory and IO Devices.

An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers. Some popular Operating Systems include Linux, Windows, Macintosh, etc.

Definition

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

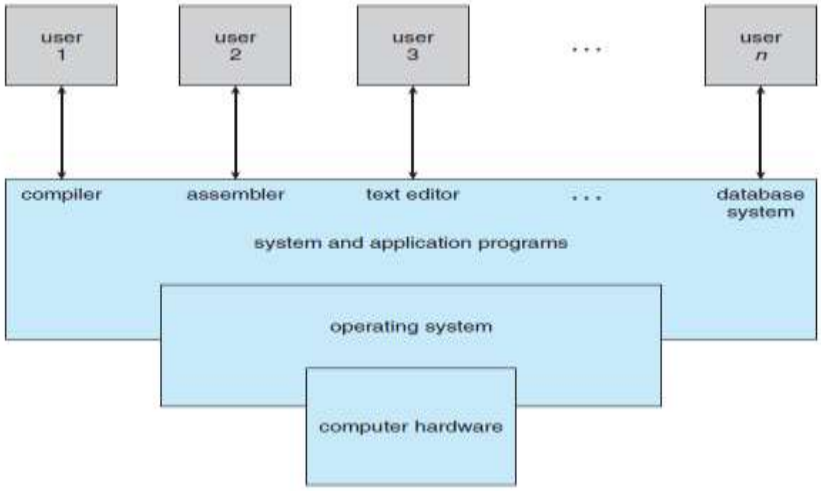


Figure 1.1 Abstract view of the components of a computer system

The role of operating system's in the overall computer system is divided roughly into four components: the ***hardware,*** the ***operating system,*** the ***application programs,*** and the ***users*** (Figure 1.1).

1.1 Basic OS Functions

Following are some of important functions of an operating System.

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

Memory Management

Memory management refers to management of Primary Memory or Main Memory. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management −

- Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

File Management

An Operating System does the following activities for file management −

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

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

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

1.2 Resource Abstraction

The concept of an operating system as primarily providing abstractions to application programs is a top-down view. An alternative, bottom-up view holds that the operating system is there to manage all the pieces of a complex system. Modern computers consist of processors, memories, timers, disks, mice, network interfaces, printers, and a wide variety of other devices. In the alternative view, the job of the operating system is to provide for an orderly and controlled allocation of the processors, memories, and input/output devices among the various programs competing for them.

When a computer (or network) has multiple users, the need for managing and protecting the memory, input/output devices, and other resources is even greater, since the users might otherwise interface with one another. In addition, users often need to share not only hardware, but information (files, databases, etc.) as well. In short, this view of the operating system holds that its primary task is to keep track of which programs are using which resources, to grant resource requests, to account for usage, and to mediate conflicting requests from different programs and users.

Resource 2management includes **multiplexing** (sharing) resources in two different ways:

1. Time Multiplexing
2. Space Multiplexing

1. Time Multiplexing

When the resource is time multiplexed, different programs or users take turns using it. First one of them gets to use the resource, then another, and so on.

For example:

With only one CPU and multiple programs that want to run on it, operating system first allocates the CPU to one long enough, another one gets to use the CPU, then another and then eventually the first one again.

Determining how the resource is time multiplexed – who goes next and for how long – is the task of the operating system.

2. Space Multiplexing

In space multiplexing, instead of the customers taking turns, each one gets part of the resource.

For example:

Main memory is normally divided up among several running programs, so each one can be resident at the same time (for example, in order to take turns using the CPU). Assuming there is enough memory to hold multiple programs, it is more efficient to hold several programs in memory at once rather than give one of them all of it, especially if it only needs a small fraction of the total. Of course, this raises issues of fairness, protection, and so on, and it is up to the operating system to solve them.

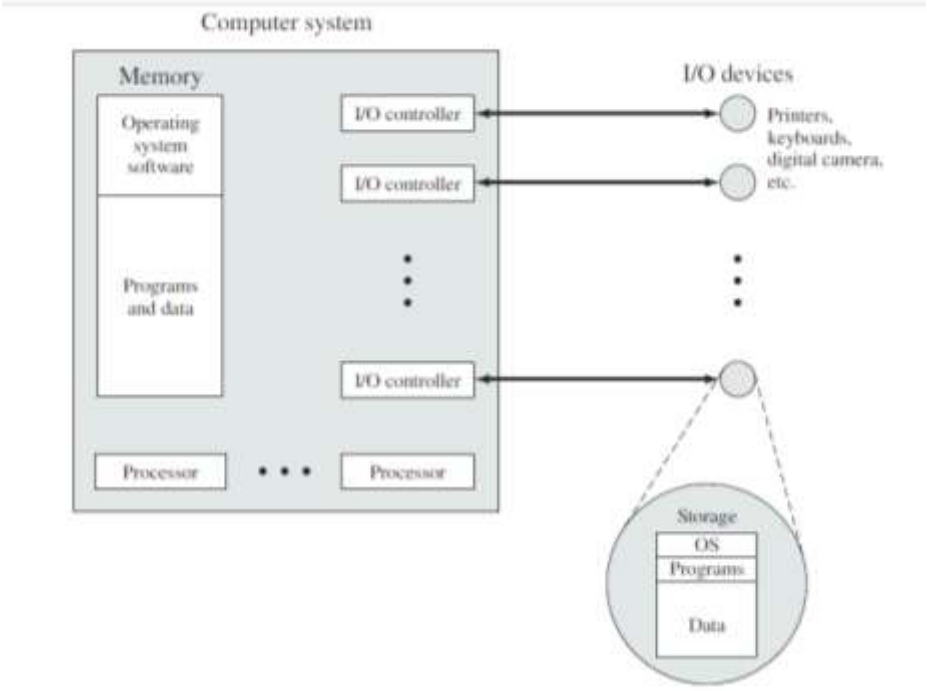


Fig 1.2 Operating system as resource manager

Figure 2.2 sugggests the main resources that are managed by the OS. This includes the Kernel or Nuvleus, which contains the most frequently used functions in the OS at a given time. The remainder of the main memory contains user programs and data.

1.3 Types of Operating Systems

1.3.1 Mainframe Operating Systems

At the high end are the operating systems for mainframes, those room-sized computers still found in major corporate data centers. These computers differ from personal computers in terms of their I/O capacity.

A mainframe with 1000 disks and millions of gigabytes of data is not unusual. The operating systems for mainframes are heavily oriented toward processing many jobs at once, most of which need prodigious amounts of I/O.

They typically offer three kinds of services: batch, transaction processing, and timesharing.

1.3.2 Server Operating Systems

One level down are the server operating systems. They run on servers, which are either very large personal computers, workstations, or even mainframes. They serve multiple users at once over a network and allow the users to share hardware and software resources. Servers can provide print service, file service, or Web service.

Internet providers run many server machines to support their customers and Websites use servers to store the Web pages and handle the incoming requests.

1.3.3 Multiprocessor Operating Systems

An increasingly common way to get major-league computing power is to connect multiple CPUs into a single system. Depending on precisely how they are connected and what is shared, these systems are called parallel computers, multicomputers, or multiprocessors.

They need special operating systems, but often these are variations on the server operating systems, with special features for communication, connectivity, and consistency.

1.3.4 Personal Computer Operating Systems

The next category is the personal computer operating system. Modern ones all support multiprogramming, often with dozens of programs started up at boot time. Their job is to provide good support to a single user.

They are widely used for word processing, spreadsheets, games, and Internet access.

1.3.5 Handheld Computer Operating Systems

Continuing on down to smaller and smaller systems, we come to tablets, smartphones and other handheld computers.

A handheld computer, originally known as a **PDA** (**Personal Digital Assistant**), is a small computer that can be held in our hand during operation.

As we have already seen, this market is currently dominated by Google's Android and Apple's iOS, but they have many competitors. Most of these devices boast multicore CPUs, GPS, cameras and other sensors, copious amounts of memory, and sophisticated operating systems.

1.3.6 Embedded Operating Systems

Embedded systems run on the computers that control devices that are not generally thought of as computers and which do not accept user-installed software. Typical examples are microwave ovens, TV sets, cars, DVD recorders, traditional phones, and MP3 players.

The main property which distinguishes embedded systems from handhelds is the certainty that no untrusted software will ever run on it. This means that there is no need for protection between applications, leading to design simplification.

1.3.7 Sensor-Node Operating Systems

Networks of tiny sensor nodes are being deployed for numerous purposes. These nodes are tiny computers that communicate with each other and with a base station using wireless communication. Sensor networks are used to protect the perimeters of buildings, guard national borders, detect fires in forests, measure temperature and precipitation for weather forecasting, glean information about enemy movements on battlefields, and much more.

The sensors are small battery-powered computers with built-in radios. They have limited power and must work for long periods of time unattended outdoors, frequently in environmentally harsh conditions.

Each sensor node is a real computer, with a CPU, RAM, ROM, and one or more environmental sensors.

It runs a small, but real operating system, usually one that is event driven, responding to external events or making measurements periodically based on an internal clock.

The operating system has to be small and simple because the nodes have little RAM and battery lifetime is a major issue.

Also, as with embedded systems, all the programs are loaded in advance; users do not suddenly start programs they downloaded from the Internet, which makes the design much simpler. TinyOS is a well-known operating system for a sensor node.

1.3.8 Real-Time Operating Systems

Another type of operating system is the real-time system. These systems are characterized by having time as a key parameter. For example, in industrial process-control systems, real-time computers have to collect data about the production process and use it to control machines in the factory.

Often there are hard deadlines that must be met. For example, if a car is moving down an assembly line, certain actions must take place at certain instants of time. If, for example, a welding robot welds too early or too late, the car will be ruined. If the action absolutely *must* occur at a certain moment (or within a certain range), we have a **hard real-time system**. Many

of these are found in industrial process control, avionics, military, and similar application areas. These systems must provide absolute guarantees that a certain action will occur by a certain time.

A **soft real-time system**, is one where missing an occasional deadline, while not desirable, is acceptable and does not cause any permanent damage. Digital audio or multimedia systems fall in this category. Smartphones are also soft real-time systems.

Since meeting deadlines is crucial in (hard) real-time systems, sometimes the operating system is simply a library linked in with the application programs, with everything tightly coupled and no protection between parts of the system.

1.3.9 Smart Card Operating Systems

The smallest operating systems run on smart cards, which are credit-card-sized devices containing a CPU chip. They have very severe processing power and memory constraints. Some are powered by contacts in the reader into which they are inserted, but contactless smart cards are inductively powered, which greatly limits what they can do. Some of them can handle only a single function, such as electronic payments, but others can handle multiple functions.

Some smart cards are Java oriented. This means that the ROM on the smart card holds an interpreter for the Java Virtual Machine (JVM). Java applets (small programs) are downloaded to the card and are interpreted by the JVM interpreter. Some of these cards can handle multiple Java applets at the same time, leading to multiprogramming and the need to schedule them. Resource management and protection also become an issue when two or more applets are present at the same time. These issues must be handled by the (usually extremely primitive) operating system present on the card.

1.4 Multiprogramming Systems

To overcome the problem of underutilization of CPU and main memory, the multiprogramming was introduced. The multiprogramming is interleaved execution of multiple jobs by the same computer. In multiprogramming system, when one program is waiting for I/O transfer; there is another program ready to utilize the CPU. So it is possible for several jobs to share the time of the CPU. There are a number of jobs available to the CPU (placed in main memory) and a portion of one is executed then a segment of another and so on.

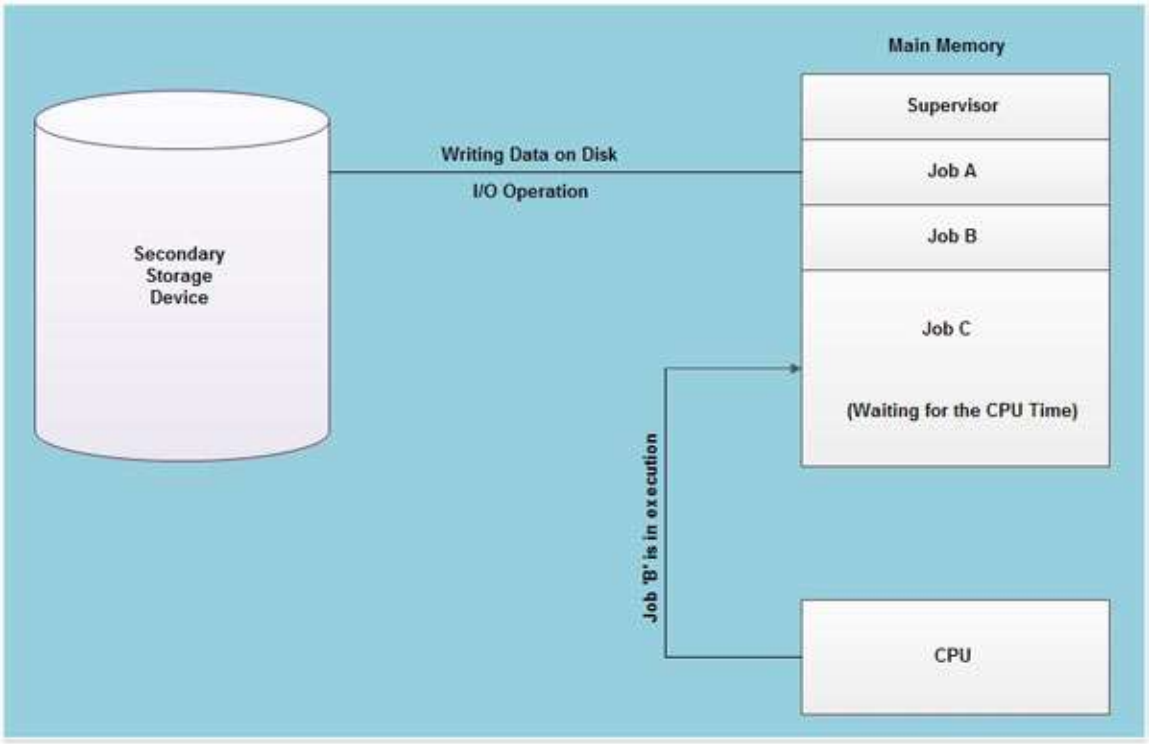


Figure 1.3. A simple process of multiprogramming

as shown in Figure 1.3, at the particular situation, job' A' is not utilizing the CPU time because it is busy in I/ 0 operations. Hence the CPU becomes busy to execute the job 'B'. Another job C is waiting for the CPU for getting its execution time. So in this state the CPU will never be idle and utilizes maximum of its time.

A program in execution is called a "Process", "Job" or a "Task". The concurrent execution of programs improves the utilization of system resources and enhances the system throughput as compared to batch and serial processing. In this system, when a process requests some I/O to allocate; meanwhile the CPU time is assigned to another ready process. So, here when a process is switched to an I/O operation, the CPU is not set idle.

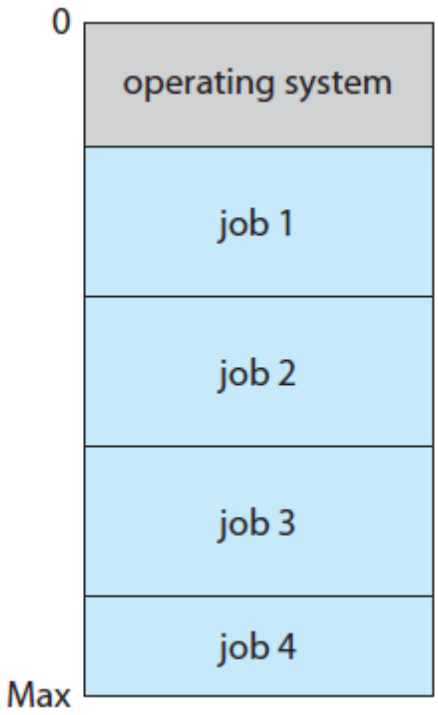


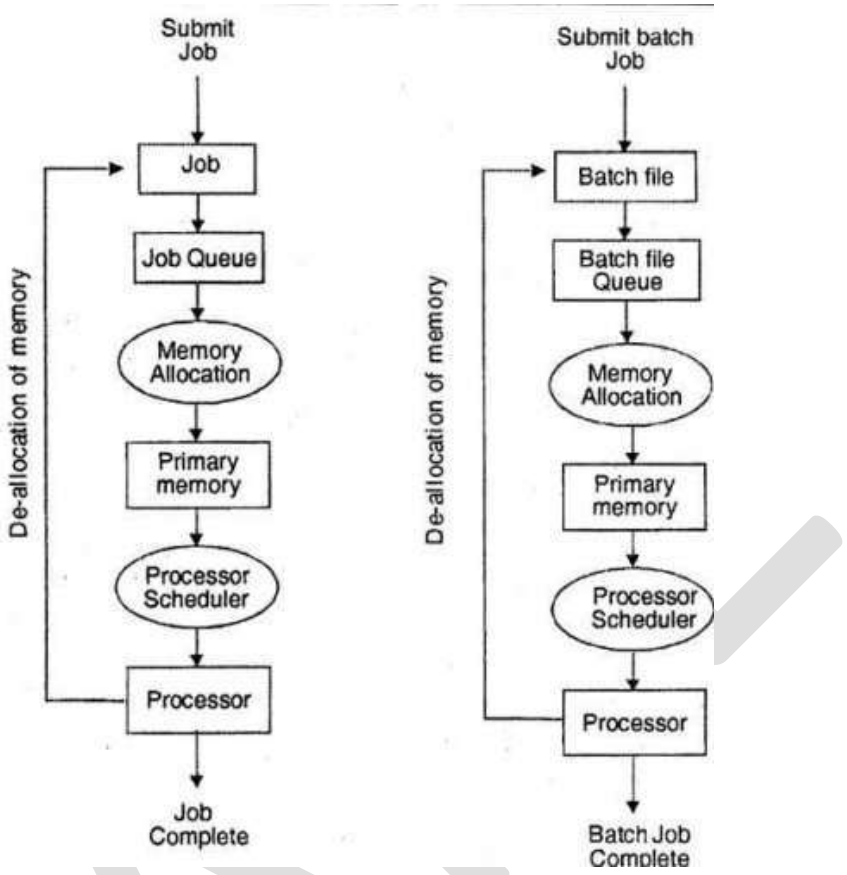
Figure 1.4 Memory layout for a multiprogramming system.

1.5 Batch Systems

To avoid the problems of early systems, the batch processing systems were introduced. The problem of early systems was more setup time. So the problem of more set up time was reduced by processing the jobs in batches, known as *batch processing system.* In this approach similar jobs were submitted to the CPU for processing and were run together.

The main function of a batch processing system is to automatically keep executing the jobs in a batch. This is the important task of a batch processing system i.e. performed by the 'Batch Monitor' resided in the low end of main memory.

This technique was possible due to the invention of hard-disk drives and card readers. By placing the identical jobs (jobs with the similar needs) in the same batch, the batched jobs were executed automatically one after another saving its time by performing the activities (like loading of compiler) only for once. It resulted in improved system utilization due to reduced turnaround time.



Traditional Job Processing **Batch File Processing**

Figure 1.5

In this environment there was no interactivity and the users had no direct control. In this system, only one job could engage the processor at a time and if there was any input/ output operation the processor had to sit idle till the completion of I/O job. So it resulted to the underutilization of CPU time.

Though, it was an improved technique in reducing the system setup time but still there were some limitations with this technique like as under-utilization of CPU time, non-interactivity of user with the running jobs etc. In batch processing system, the jobs of a batch were executed one after another. But while these jobs were performing I/O operations; meantime the CPU was sitting idle resulting to low degree of resource utilization.

1.6 Time Sharing Operating System

A **time sharing system** allows many users to share the computer resources simultaneously. In other words, time sharing refers to the allocation of computer resources in time slots to several programs simultaneously. For example a mainframe computer that has many users logged on to

it. Each user uses the resources of the mainframe -i.e. memory, CPU etc. The users feel that they are exclusive user of the CPU, even though this is not possible with one CPU i.e. shared among different users. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

The time sharing systems were developed to provide an interactive use of the computer system. A time shared system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. It allows many users to share the computer resources simultaneously. As the system switches rapidly from one user to the other, a short time slot is given to each user for their executions. In time-shared systems multiple processes are managed simultaneously which requires an adequate management of main memory so that the processes can be swapped in or swapped out within a short time.

The time sharing system provides the direct access to a large number of users where CPU time is divided among all the users on scheduled basis. The OS allocates a set of time to each user. When this time is expired, it passes control to the next user on the system. The time allowed is extremely small and the users are given the impression that they each have their own CPU and they are the sole owner of the CPU. This short period of time during that a user gets attention of the CPU; is known as a *time slice or a quantum.* The concept of time sharing system is shown in figure.

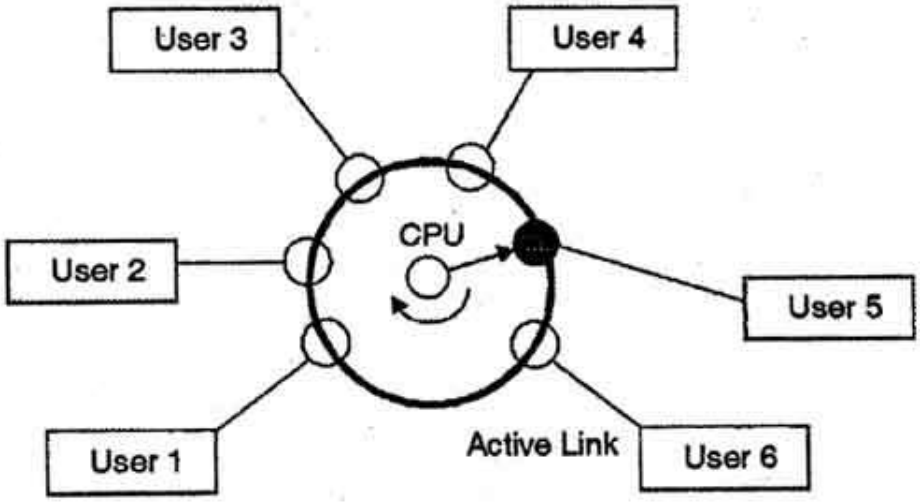


Figure 1.6 Time Sharing System

In above figure the user 5 is active but user 1, user 2, user 3, and user 4 are in waiting state whereas user 6 is in ready status.

As soon as the time slice of user 5 is completed, the control moves on to the next ready user i.e. user 6. In this state user 2, user 3, user 4, and user 5 are in waiting state and user 1 is in ready state. The process continues in the same way and so on.

The time-shared systems are more complex than the multi-programming systems. In time-shared systems multiple processes are managed simultaneously which requires an adequate management of main memory so that the processes can be swapped in or swapped out within a short time.

Note: The term 'Time Sharing' is no longer commonly used, it has been replaced by 'Multitasking System'.

1.7 Operating Systems for Personal Computers

The next category is the personal computer operating system. Modern ones all support multiprogramming, often with dozens of programs started up at boot time. Their job is to provide good support to a single user. They are widely used for word processing, spreadsheets, and Internet access. Common examples are Linux, FreeBSD, Windows Vista, and the Macintosh operating system.

A local computer manufacturer, Seattle Computer Products, developed an operating system, **DOS (Disk Operating System).** The revised system was renamed **MS-DOS (MicroSoft Disk Operating System)** and quickly came to dominate the IBM PC market. Although the initial version of MS-DOS was fairly primitive, subsequent versions included more advanced features, including many taken from UNIX. CP/M, MS-DOS, and other operating systems for early microcomputers were all based on users typing in commands from the keyboard.

Engelbart invented the **GUI Graphical User Interface,** complete with windows, icons, menus, and mouse. These ideas were adopted by researchers at Xerox PARC and incorporated into machines they built. In the creative world of graphic design, professional digital photography, and professional digital video production, Macintoshes are very widely used and their users are very enthusiastic about them. When Microsoft decided to build a successor to MS-DOS, it was strongly influenced by the success of the Macintosh. It produced a GUI-based system called Windows, which originally ran on top of MS-DOS (i.e., it was more like a shell than a true operating system).

Personal computer (PC) operating systems support complex games, business applications, and everything in between. The PC was an inherently single-user machine. Modern Windows, however, supports the sharing of a PC among multiple users. Each user that is logged on using the GUI has a **session** created to represent the GUI environment he will be using and to contain all the processes created to run his applications. Windows allows multiple sessions to exist at the same time on a single machine. However, Windows only supports a single console, consisting of all the monitors, keyboards, and mice connected to the PC. Only one session can be connected to the console at a time. From the logon screen displayed on the console, users can create new sessions or attach to an existing session that was previously created. This allows multiple users to share a single PC without having to log off and on between users.

1.8 Workstation Operating System

Workstation operating system is primarily designed to run applications. Those applications can be text processor, a spreadsheet application, presentation software, video or audio editors, games, etc. Workstation operating systems can run services, but are not really designed for it. By services we mean on services that other users can use on the network. For example, services like DHCP, DNS, FTP, Mail, Web servers, etc. Some of that services actually are available on workstation operating systems, but they are not optimized for them. Almost all workstation operating systems support multiple user accounts on the same workstation, but the thing is they are not designed to be concurrent multi-user. Workstation OS are not designed to support multiple users at the same time, meaning they don't do it very well. Most Windows operating systems have a limit of 10 concurrent users at the time. This limit is applied when we share something on our workstation computer, for example printer or some folder. Only 10 users maximum will be able to utilize our shared resources on the workstation OS. Also, workstation operating systems are designed to run on lower end hardware. That's why the workstation operating system can run on many different and cheap computers. Examples of workstation operating systems include Windows 95, Windows 98, Windows ME, Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, and various Macintosh operating systems as well.

1.9 Process control

A process or job executing one program may want to load() and execute() another program. This feature allows the command interpreter to execute a program as directed by, for example, a user command, the click of a mouse, or a batch command. An interesting question is where to return control when the loaded program terminates. This question is related to whether the existing program is lost, saved, or allowed to continue execution concurrently with the new program.

If control returns to the existing program when the new program terminates, we must save the memory image of the existing program; thus, we have effectively created a mechanism for one program to call another program. If both programs continue concurrently, we have created a new job or process to be multiprogrammed. Often, there is a system call specifically for this purpose (create process() or submit job()).

If we create a new job or process, or perhaps even a set of jobs or processes, we should be able to control its execution. This control requires the ability to determine and reset the attributes of a job or process, including the job's priority, its maximum allowable execution time, and so on (get process attributes() and set process attributes()).We may also want to terminate a job or process that we created (terminate process()) if we find that it is incorrect or is no longer needed.

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

Process control
◦ end, abort
◦ load, execute
◦ create process, terminate process
◦ get process attributes, set process attributes
◦ wait for time
◦ wait event, signal event
◦ allocate and free memory

To start a new process, the shell executes a fork() system call. Then, the selected program is loaded into memory via an exec() system call, and the program is executed. Depending on the way the command was issued, the shell then either waits for the process to finish or runs the process "in the background." In the latter case, the shell immediately requests another command. When a process is running in the background, it cannot receive input directly from the keyboard, because the shell is using this resource. I/O is therefore done through files or through a GUI interface. Meanwhile, the user is free to ask the shell to run other programs, to monitor the progress of the running process, to change that program's priority, and so on. When the process is done, it executes an exit() system call to terminate, returning to the invoking process a status code of 0 or a nonzero error code. This status or error code is then available to the shell or other programs.

1.10 Real Time System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021

(For the candidates admitted in 2018 onwards)

Unit I

CLASS : II B.Sc CT Semester:IV

SUBJECT : Operating Systems Batch:2018-2021

	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	File management function of the operating system includes	File creation and deletion	Disk scheduling	Process scheduling	Multiprogramming	File creation and deletion
2	What is a shell	It is a hardware component	It is a command interpreter	It is a part in compiler	It is a tool in CPU scheduling	It is a command interpreter
3	The queue has maximum length 0; thus, the link cannot have any	Zero capacity	Bounded capacity	Unbounded capacity	synchronous	Zero capacity
4	Let S and Q be two semaphores initialized to 1,where Po and p1 processes the following statements wait(S);wait(Q);--:signal(S) and signal(Q) and wait(Q);wait(S);_---;signal(Q);signal(S);respectively.The above sitaution depicts a	semaphore	deadlock	signal	interrupt	deadlock
5	The OS that groups similar jobs is called as	Network OS	Distributed OS	Distributed OS	Batch OS	Batch OS
6	_____is a program that manages the computer hardware and acts as an intermediary between the computer user and the computer hardware.	hardware acceleration	Operating System	compiler	logical transcation	Operating System

7	_____manages the execution of user programs to prevent errors and improper use of the computer	resource allocator	work station	main frame	control program	control program
8	_____ is a program associated with the operating system but are not part of the	System Program	User program	System calls	Functions	System Program
10	A more common definition is that the operating system is the one program running at all times on the computer usually called_____	bootstrap	firmware	kernel	read-only memory	kernel
11	The simplest CPU scheduling algorithm is the	FCS	SJS	FCFS	DFG	FCFS
12	computer system can be divided into _____ components	2	3	4	5	2
13	____were the first computers used to tackle many commercial & scientific application.	Mainframe computer system	Mainframe computer service	multiframe computer system	multiframe computer service	Mainframe computer system
14	_____system is the collection of computer that act,work and appear as one large computer size of a distributed system.	distributed	symmetric	asymmetric	multiple	distributed
15	_____contains the address of an instruction to be fetched from memory	Program counter (PC)	Instruction register (IR)	Control registers	Status registers	Instruction register (IR)
16	_____contains the instruction most recently fetched.	Program counter (PC)	Instruction register (IR)	Control registers	Status registers	Program counter (PC)
17	The kernel is a_____	memory manager	resource manager	file manager	directory manager	resource manager

18	Main function of shared memory is_____	to use primary memory efficently	to do intra process communication	to do inter process communication	to do other process communication	to do inter process communicatio n
19	Disk scheduling includes deciding_____	which should be accessed next	order in which disk access requests must be serviced	the physical location of the file	the logical location of the file	order in which disk access requests must be serviced
20	Memory protection is normally done by _____	the processor and the associated hardware	the operating system	the compiler	the user program	the processor and the associated hardware
21	With _____ more than one process can be running simultaneously each on a different processer.	Multiprogrammin g	Uniprocessing	Multiprocessing	Uniprogrammin g	Multiprocessin g
22	In Banker's Algorithm _____ conditions are allowed	mutual-exclusion	Time-sharing	race condition	cooperating processes	mutual-exclusion
23	_____is the process of actually determining that a deadlock exists	Deadlock detection	Deadlock prevention	fault tolerant	process synchronization	Deadlock detection
24	Once a system has become _____ the deadlock must be broken by removing one or more of the necessary conditions	deadlocked	race condition	mutual-exclusion	cooperating processes	deadlocked
25	_____ is also known as parallel systems or tightly coupled systems)	Multiprocessor systems	desktop systems	Time sharing systems	Multiprogramme d systems	Multiprocessor systems

26	_____operating systems are even more complex than multi programmed operating systems.	Time-sharing	desktop systems	Multiprogrammed systems	Multiprocessor systems	Time-sharing
27	_____ operating system keeps several jobs in memory simultaneously.	Time-sharing	desktop systems	Multiprogrammed systems	Multiprocessor systems	Multiprogrammed systems
28	_____can save more money than multiple single-processor systems	Multiprocessor systems	desktop systems	Time sharing systems	Multiprogrammed systems	Multiprocessor systems
29	This ability to continue providing service proportional to the level of surviving hardware is called	fault tolerant.	graceful degradation	Economy of scale	Increased throughput	graceful degradation
30	Systems designed for graceful degradation are also called	graceful degradation	Economy of scale	fault tolerant	Increased throughput	fault tolerant
31	The most common multiple-processor systems now use	symmetric multiprocessing	asymmetric multiprocessing	multithreading	multiprogramming	symmetric multiprocessing
32	Process states are _____	Submit, Ready	Submit, Run	Ready, Run	Submit, block	Ready, Run
33	The assignment of the CPU to the first process on the ready list is called	graceful degradation	Time-sharing	dispatching	Multiprocessor systems	dispatching
34	The manifestation of a process in an operating system is a	Process state transitions	process control block	child process	cooperating processes	process control block
35	A process may spawn a new process. If it does, the creating process is called the parent process and the created process is called the	child process	Process state transitions	Process state transitions	process control block	child process
36	The communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue known as	Buffering	synchronization	asynchronization	communication link	Buffering

37	_____ is a program that includes all programs not associated with the operation of the system	Application Programs	Kernel	Thread Program	Process	Application Programs
38	_____can assume only the value 0 or the value 1	Binary semaphores	Counting semaphores	semaphore operations	normal semaphores	Binary semaphores
39	Semophores are used to solve the problem of	race condition	process synchronizatio n	mutual exclusion	belady problem	mutual exclusion
40	For multiprogramming operating system	special support from processor is essential	special support from processor is not essential	cache memory is essential	cache memory is not essential	special support from processor is not essential
41	Which is single user operating system	MS-DOS	UNIX	XENIX	LINUX	MS-DOS
42	Which operating system reacts in the actual time	Batch system	Quick response system	Real time system	Time sharing system	Real time system
43	In real time OS, which is most suitable scheduling scheme	round robin	FCFS	pre-emptive scheduling	random scheduling	pre-emptive scheduling
44	Dispatcher function is to	put tasks in I/O wait	schedule tasks in processor	change task priorities	Multitasking	put tasks in I/O wait
45	Multiprogramming systems	Are easier to develop than single programming systems	Execute each job faster	Execute more jobs in the same time	Are used only on large main frame computers	Execute more jobs in the same time
46	Operating system is	A collection of hardware components	A collection of input output devices	A collection of software routines	last entered the queue	A collection of software routines

47	Semaphores function is to	synchronize critical resources to prevent deadlock	synchronize processes for better CPU utilization	used for memory management	may cause a high I/O rate	synchronize critical resources to prevent deadlock
48	Which operating system use write through catches	UNIX	XENIX	ULTRIX	DOS	DOS
49	Which process is known for initializing a microcomputer with its OS	cold booting	boot recording	booting	warm booting	booting
50	Four necessary conditions for deadlock are non pre-emption, circular wait, hold and wait and	mutual exclusion	race condition	buffer overflow	multiprocessing	mutual exclusion
51	Remote computing services involves the use of timesharing and	multiprocessing	interactive processing	batch processing	real time processing	batch processing
52	A series of statements explaining how the data is to be processed is called	instruction	compiler	program	interpretor	program
53	Banker's algorithm deals with	deadlock prevention	deadlock avoidance	deadlock recovery	mutual exclusion	deadlock avoidance
54	Which is non pre-emptive	Round robin	FIFO	MQS	MQSF	FIFO
55	A hardware device which is capable of executing a sequence of instructions, is known as	CPU	ALU	CU	Processor	Processor
56	Distributed systems should	high security	have better resource sharing	better system utilization	low system overhead	have better resource sharing
57	Which of the following is always there in a computer	Batch system	Operating system	Time sharing system	Controlling system	Operating system

58	Which of following is not an advantage of multiprogramming	increased throughput	shorter response time	ability to assign priorities of jobs	decreased system overload	decreased system overload
59	In which of the following usually a front end processor is used	Virtual storage	Timesharing	Multiprogamming	Multithreading	Timesharing
60	Remote computing services involves the use of	multiprocessing	multiprogramming	batch processing	real time processing	batch processing
61	Banker's algorithm for resource allocation deals with	deadlock prevention	deadlock aviodance	deadlock recovery	circular wait	deadlock aviodance
62	In Unix, Which system call creates the new process?	Fork	Create	New	Old	Fork
63	The queue has finite length n; thus, at most n messages can reside in it is known as	Zero capacity	Bounded capacity	Unbounded capacity	multiprocessing	Bounded capacity
64	The queue has potentially infinite length; thus, any number of messages can wait in it is known as	Zero capacity	Bounded capacity	Unbounded capacity	multiprocessing	Unbounded capacity
65	Each process accessing the shared data excludes all others from doing so simultaneously and this is called	mutual exclusion	multiprocessing	real time processing	multiprogramming	mutual exclusion
	_____ OS runs on Apple produced hardware	Unix	Linux	Linux Mint	Apple	Apple
	PC _____ supports complex games	OS	PMT	PCB	Software	OS
66	The primary job of an OS is to _____	command resource	manage resource	provide utilities	Be user friendly	manage resource

CLASS: II B.Sc. [CT] **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: II (Operating System Organization)**

SEMESTER IV **BATCH (2018-2021)**

SYLLABUS

Operating System Organization: Processor and user modes-Kernels-System Calls and System Programs. **Process Management:** System view of the process and resources- Process abstraction-Process hierarchy-Threads-Threading issues-Thread libraries-Process Scheduling-Non pre-emptive and Preemptive scheduling algorithms-Concurrent and processes-Critical Section-Semaphores-Methods for inter-process communication- Deadlocks.

2.1 Processor and user modes

The unrestricted mode is often called *kernel mode*, but many other designations exist (*master mode*, *supervisor mode*, *privileged mode*, etc.).
Restricted modes are usually referred to as *user modes*, but are also known by many other names (*slave mode, problem state,* etc.).

1. Kernel Mode

When CPU is in **kernel mode**, the code being executed can access any memory address and any hardware resource.

- Hence kernel mode is a very privileged and powerful mode.
- If a program crashes in kernel mode, the entire system will be halted.
- It can execute any CPU instruction and reference any memory address.
- Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

2. User Mode

- When CPU is in **user mode**, the programs don't have direct access to memory and hardware resources.
- In user mode, if any program crashes, only that particular program is halted.
- That means the system will be in a safe state even if a program in user mode crashes.
- Hence, most programs in an OS run in user mode.
- Code running in user mode must allot to system APIs to access hardware or memory.
- Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

At the very least, we need two separate ***modes*** of operation: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**).
A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).

With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.

When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request. This is shown in Figure 2.1. As we shall see, this architectural enhancement is useful for many other aspects of system operation as well.

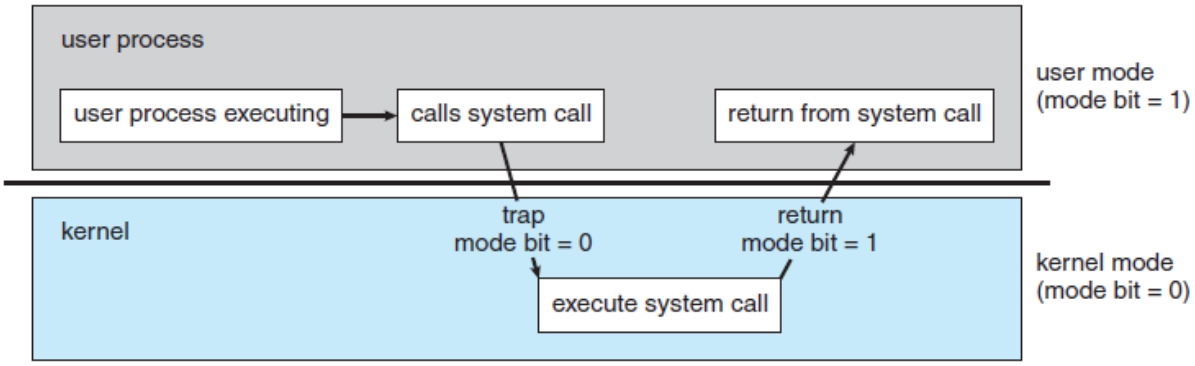


Fig 2.1 Transition from user to kernel mode.

At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another. We accomplish this protection by designating some of the machine instructions that may cause harm as **privileged instructions**.

The life cycle of instruction execution in a computer system is as follows: Initial control resides in the operating system, where instructions are executed in kernel mode. When control is given to a user application, the mode is set to user mode. Eventually, control is switched back to the operating system via an interrupt, a trap, or a system call.

2.2 Kernels

A kernel is a central component of an operating system. It acts as an interface between the user applications and the hardware. The sole aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc). The main tasks of the kernel are:

- Process management
- Device management

- Memory management
- Interrupt handling
- I/O communication
- File system...etc...

Types of Kernels

Kernels may be classified mainly in two categories

1. Monolithic
2. Micro Kernel

Monolithic Kernels

Earlier in this type of kernel architecture, all the basic system services like process and memory management, interrupt handling etc were packaged into a single module in kernel space.
This type of architecture led to some serious drawbacks like
1) Size of kernel, which was huge.
2) Poor maintainability, which means bug fixing or addition of new features resulted in recompilation of the whole kernel which could consume hours.

In a modern day approach to monolithic architecture, the kernel consists of different modules which can be dynamically loaded and un-loaded. This modular approach allows easy extension of OS's capabilities. With this approach, maintainability of kernel became very easy as only the concerned module needs to be loaded and unloaded every time there is a change or bug fix in a particular module. So, there is no need to bring down and recompile the whole kernel for a smallest bit of change. Also, stripping of kernel for various platforms (say for embedded devices etc) became very easy as we can easily unload the module that we do not want. Linux follows the monolithic modular approach.

Microkernels

This architecture caters to the problem of ever growing size of kernel code which we could not control in the monolithic approach. This architecture allows some basic services like device driver management, protocol stack, file system etc to run in user space. This reduces the kernel code size and also increases the security and stability of OS as we have the bare minimum code running in kernel. So, if suppose a basic service like network service crashes due to buffer overflow, then only the networking service's memory would be corrupted, leaving the rest of the system still functional.

In this architecture, all the basic OS services which are made part of user space are made to run as servers which are used by other programs in the system through inter process communication (IPC). eg: we have servers for device drivers, network protocol stacks, file systems, graphics, etc. Microkernel servers are essentially daemon programs like any others, except that the kernel grants

some of them privileges to interact with parts of physical memory that are otherwise off limits to most programs. This allows some servers, particularly device drivers, to interact directly with hardware. These servers are started at the system start-up.

So, what the bare minimum that Microkernel architecture recommends in kernel space?

- Managing memory protection
- Process scheduling
- Inter Process communication (IPC)

Apart from the above, all other basic services can be made part of user space and can be run in the form of servers.

2.3 System Calls and System Programs

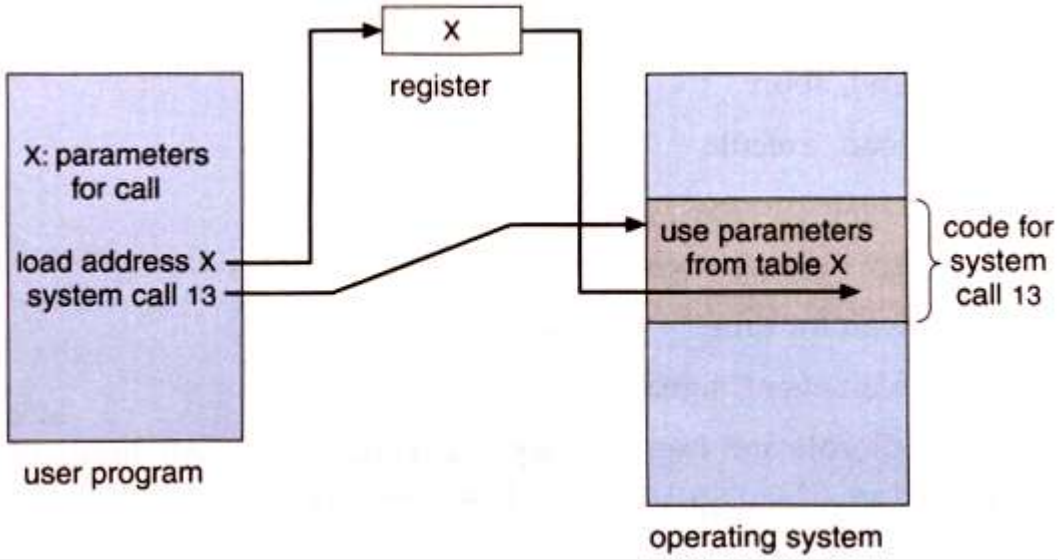
System Call

System calls provide an interface between the process and the operating system. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do. In handling the trap, the operating system will enter in the kernel mode, where it has access to privileged instructions, and can perform the desired service on the behalf of user-level process. It is because of the critical nature of operations that the operating system itself does them every time they are needed. For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

System Call Parameters

Three general methods exist for passing parameters to the OS:

1. Parameters can be passed in registers.
2. When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.
3. Parameters can also be pushed on or popped off the stack by the operating system.



Types of System Calls

There are 5 different categories of system calls: process control, file manipulation, device manipulation, information maintenance and communication.

Process Control

A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

File Management

Some common system calls are *create*, *delete*, *read*, *write*, *reposition*, or *close*. Also, there is a need to determine the file attributes – *get* and *set* file attribute. Many times the OS provides an API to make these system calls.

Device Management

Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.

User programs *request* the device, and when finished they *release* the device. Similar to files, we can *read*, *write*, and *reposition* the device.

Information Management

Some system calls exist purely for transferring information between the user program and the operating system. An example of this is *time*, or *date*.

The OS also keeps information about all its processes and provides system calls to report this information.

Communication

There are two models of interprocess communication, the message-passing model and the shared memory model.

- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

System Programs

System programs, also known as **system utilities**, provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls. These programs are not usually part of the OS kernel, but are part of the overall operating system. System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells). In some sense, they are bundles of useful system calls. They can be divided into these categories:

- **File management**. These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status information**. Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI. Some systems also support a **registry**, which is used to store and retrieve configuration information.
- **File modification**. Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.
- **Programming-language support**. Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, and PERL) are often provided with the operating system or available as a separate download.
- **Program loading and execution**. Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.
- **Communications**. These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse Web pages, to send e-mail messages, to log in remotely, or to transfer files from one machine to another.
- **Background services**. All general-purpose systems have methods for launching certain system-program processes at boot time. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted. Constantly running system-program processes are known as **services**, **subsystems**, or daemons.

2.4 Process Management

System view of the process and resources

The Process:

A process generally consists of:

• The program's instructions (aka. the "program text")
• CPU state for the process (program counter, registers, flags, …)
• Memory state for the process
• Other resources being used by the process

Informally, as mentioned earlier, a process is a program in execution. A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers. A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables. A process may also include a heap, which is memory that is dynamically allocated during process run time. The structure of a process in memory is shown in Figure 2.1.

We emphasize that a program by itself is not a process; a program is a *passive* entity, such as a file containing a list of instructions stored on disk (often called an executable file), whereas a process is an *active* entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program becomes a process when an executable file is loaded into memory. Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line (as in prog. exe or a. out.)

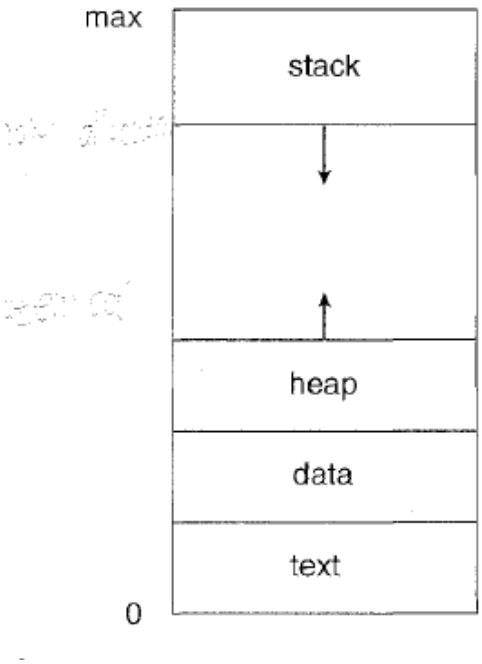


Fig 2.1 Process in memory

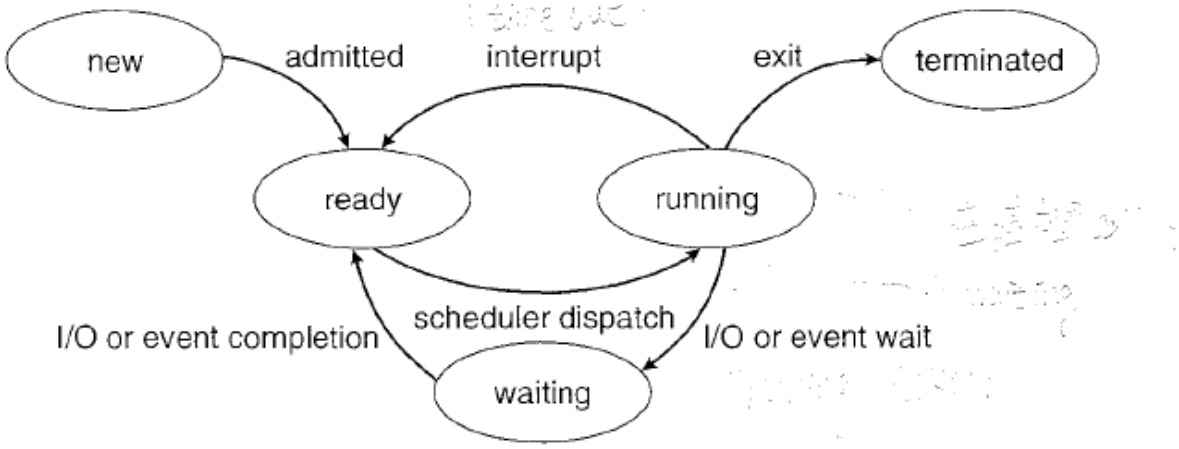


Fig 2.2 Diagram of process state

Although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences. For instance, several users may be running different copies of the mail program, or the same user may invoke many copies of the Web browser program. Each of these is a separate process; and although the text sections are equivalent, the data, heap, and stack sections vary. It is also common to have a process that spawns many processes as it runs.

Process State Transition

As a process executes, it changes state.

The state of a process is defined in part by the current activity of that process.

Each process may be in one of the following states:

New. The process is being created.

Ready. The process is waiting to be assigned to a processor.

Running. Instructions are being executed.

Waiting. The process is waiting for some event to occur (such as an I/0 completion or reception of a signal).

Terminated. The process has finished execution.

These names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems, however. Certain operating systems also more finely delineate process states. It is important to realize that only one process can be *running* on any processor at any instant. Many processes may be *ready* and *waiting*, however.

The state diagram corresponding to these states is presented in Figure 2.2.

2.5 Process Abstraction

Process abstraction Main Point: What are processes? How are process, programs, threads, and address spaces related?

The Abstraction: A Process

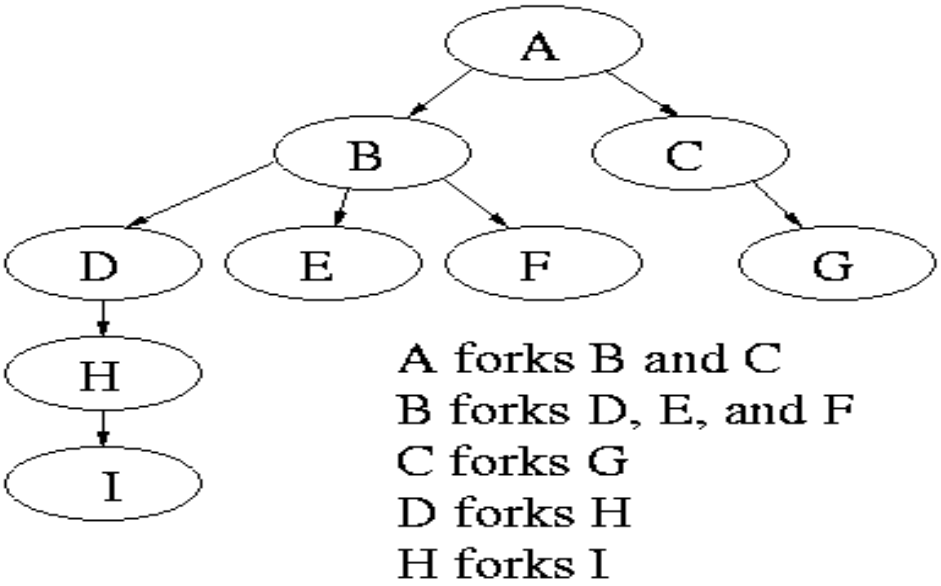
- The abstraction provided by the OS of a running program is something we will call a process. As we said above, a process is simply a running program; at any instant in time,
- One obvious component of machine state that comprises a process is its memory.
- Instructions lie in memory; the data that the running program reads and writes sits in memory as well.
- Thus the memory that the process can address (called its address space) is part of the process.
- Also part of the process's machine state are registers; many instructions explicitly read or update registers and thus clearly they are important to the execution of the process.
- Note that there are some particularly special registers that form part of this machine state.
- Finally, programs often access persistent storage devices too. Such I/O information might include a list of the files the process currently has open.

2.6 Process Hierarchies

Modern general purpose operating systems permit a user to create and destroy processes.

- In unix this is done by the **fork** system call, which creates a **child** process, and the **exit** system call, which terminates the current process.
- After a fork both parent and child keep running (indeed they have the *same* program text) and each can fork off other processes.
- A process tree results. The root of the tree is a special process created by the OS during startup.
- A process can *choose* to wait for children to terminate. For example, if C issued a wait() system call it would block until G finished.

Old or primitive operating system like MS-DOS are not multiprogrammed so when one process starts another, the first process is *automatically* blocked and waits until the second is finished.



In some systems, when a process creates another process, the parent process and child process continue to be associated in certain ways. The child process can itself create more processes, forming a process hierarchy.

In UNIX, a process and all of its children and further descendants together form a process group. When a user sends a signal from the keyboard, the signal is delivered to all members of the process group currently associated with the keyboard. Individually, each process can catch the signal, ignore the signal, or take the default action, which is to be killed by the signal.

As another example of where the process hierarchy plays a role, let us look at how UNIX initializes itself when it is started. A special process, called *init,* is present in the boot image. When it starts running, it reads a file telling how many terminals there are. Then it forks off one new process per terminal. These processes wait for someone to log in. If a login is successful, the login process executes a shell to accept commands. These commands may start up more processes, and so forth. Thus, all the processes in the whole system belong to a single tree, with *init* at the root.

In contrast, Windows has no concept of a process hierarchy. All processes are equal. The only hint of a process hierarchy is that when a process is created, the parent is given a special token (called a **handle)** that it can use to control the child. However, it is free to pass this token to some other process, thus invalidating the hierarchy. Processes in UNIX cannot disinherit their children.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021

(For the candidates admitted in 2018 onwards)

Unit II

CLASS : II B.Sc CT Semester:IV

SUBJECT : Operating Systems Batch:2018-2021

	Questions	opt1	opt2	opt3	opt4	Answer
1	Memory is array of _____	bytes	circuits	ics	ram	bytes
2	CPU fetches instructions from _____	memory	pendrive	dvd	cmos	memory
3	Program must be in _____	memory	pendrive	dvd	cmos	memory
4	OS chooses one process whenever CPU is _____.	Waiting	Idle	Busy	Ready	Idle
5	Collection of process in disk forms_____	input queue	output queue	stack	circle	input queue
6	Address space of computer starts at _____	0000	4444	3333	2222	0000
7	If process location is found during compile time then _____ code is generated	absolute	relative	approximate	more or less	absolute
8	Address generated by CPU is _____ address	logical	physical	direct	indirect	logical
9	Logical address can be also called as _____ address	physical	virtual	direct	indirect	virtual
10	Run time mapping is done using _____	MMU	CPU	CU	IU	MMU
11	In address binding base register is also called as _____	relocation register	memory register	hard disk	pendrive	relocation register
12	Better memory space is utilized using _____	dynamic loading	dynamic linking	registers	array of words	dynamic loading
13	_____ routine is never loaded in dynamic loading	unused	used	regular	recursive	unused

14	Some operating systems support only _____ linking	static	dynamic	temporary	interruptive	static
15	_____ is a code that locates library routine	stub	dll	recursive routine	exe file	stub
16	_____ can be used to manage large memory requirement for a process	overlays	swapping	roll in and out	libraries	overlays
17	_____ error is raised in memory	addressing	swapping	dynamic	index	addressing
18	Set of _____ are scattered throughout the memory	holes	gaps	free space	words	holes
19	_____ can be internal and external	fragmentation	merging	grouping	fixing	fragmentation
20	_____ is used to divide a process into fixed size chunks	paging	segmentation	sp	swapping	paging
21	In paging physical memory is divided into _____	frames	pages	segments	bytes	frames
22	In paging virtual memory is divided into _____	frames	pages	segments	bytes	pages
23	_____ is first of virtual address in paging	page number	segment number	frame number	offset	page number
24	_____ is second part of virtual address in paging	page number	segment number	frame number	offset	offset
25	Page mapping entries are found in _____	page table	segment table	hash table	pointing table	page table
26	Page size is defined by _____	hardware	software	os	kernel	hardware
27	_____ is first in mapping of virtual to physical address in paging	direct	associate	direct & associative	pointing	direct
28	_____ is second in mapping of virtual to physical address in paging	direct	associate	direct & associative	pointing	associate
29	_____ is third in mapping of virtual to physical address in paging	direct	associate	direct & associative	pointing	direct & associative

30	_____ is used to divide a process into variable size chunks	paging	segmentation	sp	swapping	segmentation
31	In segmentation virtual memory is divided into _____	frames	pages	segments	bytes	segments
32	_____ view is supported in segmentation	user	system	cpu	manager	user
33	_____ is format for segmentation virtual address	(s,d)	(p,d)	(v,d)	(k,d)	(s,d)
34	_____ is the first element in segment table	limit	base	offset	page number	limit
35	_____ is the second element in segment table	limit	base	offset	page number	base
36	Addressing in segmentation is similar as _____ addressing in paging	direct	associate	direct & associative	pointing	direct
37	The term PDA is _____	Personal Digital Assistant	Personal Data Assistant	Personal Data Accountant	Private Digital Assistant	Personal Digital Assistant
38	_____ is organization in physical memory in segmentation	frames	pages	segments	bytes	frames
39	_____ memory is used to manage incompleteness of a process execution	virtual	physical	rom	eprom	virtual
40	virtual memory abstracts _____ memory	virtual	eerom	main	eprom	main
41	_____ reasons are there for existence for virtual memory	1	2	3	4	3
42	_____ benefits are there from virtual memory	1	2	3	4	3
43	Virtual memory is commonly implemented by _____ paging	demand	bargain	quarrel	order	demand

44	_____ fault occurs when desired page is not in memory	page	segment	pages	segments	page
45	_____ table is used in demand paging	page	segment	pages	segments	page
46	_____ methods are there for process creation	1	2	3	4	2
47	_____ method implements partial sharing in process creation	copy on write	memory mapping	paging	segmentation	copy on write
48	_____ is done for page fault	replacement	swapping	logging	locking	replacement
49	_____ is unrealizable page replacement algorithm	optimal	FIFO	LRU	NRU	optimal
50	_____ is first page replacement algorithm	optimal	FIFO	LRU	NRU	FIFO
51	_____ is second page replacement algorithm	optimal	FIFO	LRU	NRU	optimal
52	_____ is third page replacement algorithm	optimal	FIFO	LRU	NRU	NRU
53	_____ is associated with each page in optimal algorithm	label	index	number	identity	label
54	_____ labelled page replaced in optimal algorithm	highest	lowest	moderate	below average	highest
55	_____ end page is removed in fifo algorithm	rear	head	top	bottom	head
56	Operating systems have a _____for each device controller	Process	Device driver	Controller	Allocator	Device driver
57	_____ is called as high paging activity	thrashing	smashing	mocking	breaking	thrashing
58	_____ occurs frequently during thrashing	page fault	segment fault	memory fault	address fault	page fault

59	_____ strategy is used to solve thrashing a little	working set	pff	lpr algorithm	gpl algorithm	working set
60	_____ algorithm is used to solve thrashing a little	working set	pff	lpr algorithm	gpl algorithm	lpr algorithm
61	To access the services of operating system, the interface is provided by the _____	System calls	API	Library	Assembly instructions	API

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

SYLLABUS

Memory Management: Physical and Virtual address space-Memory Allocation strategies – Fixed and Variable partitions-Paging-Segmentation-Virtual memory

Memory Management

The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be at least partially in main memory during execution. To improve both the utilization of the CPU and the speed of its response to users, a general-purpose computer must keep several processes in memory. Many memory-management schemes exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation.

Selection of a memory-management scheme for a system depends on many factors, especially on the hardware design of the system. Most algorithms require hardware support.

Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes.

It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Memory consists of a large array of bytes, each with its own address.

The memory unit sees only a stream of memory addresses; it does not know how they are generated (by the instruction counter, indexing, indirection, literal addresses, and so on) or what they are for (instructions or data).

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

Basic Hardware: Main memory and the registers built into the processor itself are the only general-purpose storage that the CPU can access directly. There are machine instructions that take memory addresses as arguments, but none that take disk addresses. Therefore, any instructions in execution, and any data being used by the instructions, must be in one of these direct-access storage devices. If the data are not in memory, they must be moved there before the CPU can operate on them. Make sure that each process has a separate memory space. Separate per-process memory space protects the processes from each other and is fundamental to having multiple processes loaded in memory for concurrent execution. To separate memory spaces, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses. We can provide this protection by using two registers, usually a base and a limit, as illustrated in Figure 3.1.

The **base register** holds the smallest legal physical memory address; the **limit register** specifies the size of the range. For example, if the base register holds 300040 and the limit register is 120900, then the program can legally access all addresses from 300040 through 420939 (inclusive).

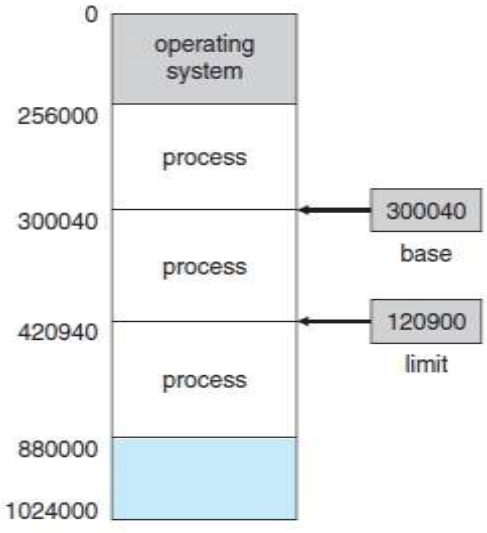


Fig 3.1. A base and a limit register define a logical address space.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

Address Binding

Usually, a program resides on a disk as a binary executable file. To be executed, the program must be brought into memory and placed within a process. Depending on the memory management in use, the process may be moved between disk and memory during its execution. The processes on the disk that are waiting to be brought into memory for execution form the **input queue**.

Addresses in the source program are generally symbolic (such as the variable count). A compiler typically **binds** these symbolic addresses to relocatable addresses (such as "14 bytes from the beginning of this module"). The linkage editor or loader in turn binds the relocatable addresses to absolute addresses (such as 74014). Each binding is a mapping from one address space to another.

Process Address Space

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

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

1 Symbolic addresses

The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.

2 Relative addresses

At the time of compilation, a compiler converts symbolic addresses into relative addresses.

3 Physical addresses

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

Classically, the binding of instructions and data to memory addresses can be done at any step along the way:

• **Compile time**. If you know at compile time where the process will reside in memory, then **absolute code** can be generated. The MS-DOS .COM-format programs are bound at compile time.

Load time. If it is not known at compile time where the process will reside in memory, then the compiler must generate **relocatable code**.

Execution time. If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware must be available for this scheme to work.

Logical versus Physical Address Space

An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address. Logical address is also known a Virtual address.

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

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

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

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

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

An address generated by the CPU is commonly referred to as a **logical address**, whereas an address seen by the memory unit—that is, the one loaded into the **memory-address register** of the memory—is commonly referred to as a **physical address**. The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, the execution-time address binding scheme results in differing logical and physical addresses. In this case, we usually refer to the logical address as a **virtual address**.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

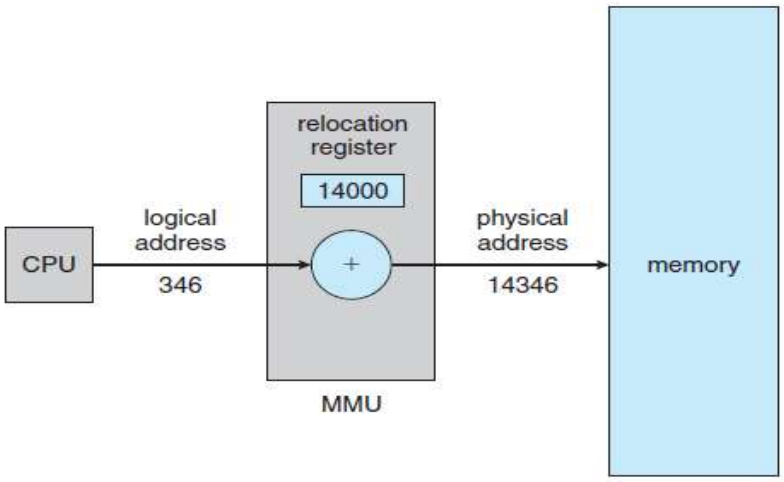


Fig 3.2. Physical address and Logical Address

The set of all logical addresses generated by a program is a **logical address space**. The set of all physical addresses corresponding to these logical addresses is a **physical address space**. Thus, in the execution-time address-binding scheme, the logical and physical address spaces differ.

The run-time mapping from virtual to physical addresses is done by a hardware device called the **memory-management unit (MMU)**. The base register is now called a **relocation register**. The value in the relocation register is added to every address generated by a user process at the time the address is sent to memory (see Figure 8.4). For example, if the base is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location14000; an access to location 346 is mapped to location 14346.

The user program never sees the real physical addresses.

Static vs Dynamic Loading

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

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

Static vs Dynamic Linking

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

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

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory to a backing store , and then brought back into memory for continued execution.

Backing store is a usually a hard disk drive or any other secondary storage which fast in access and large enough to accommodate copies of all memory images for all users. It must be capable of providing direct access to these memory images.

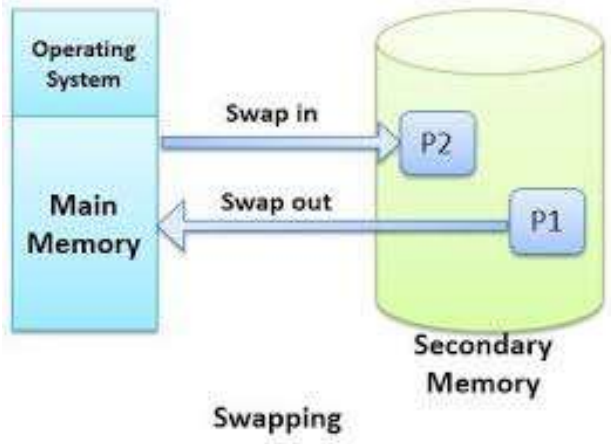


Fig 3.3. Process of Swapping

Memory Allocation

Now we are ready to turn to memory allocation. One of the simplest methods for allocating memory is to divide memory into several fixed-sized **partitions**. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. In this **multiplepartition method**, when a partition is free, a process is selected from the input

queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. it is used primarily in batch environments.

In the **variable-partition** scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes and is considered one large block of available memory, a **hole**. Eventually, as you will see, memory Contains a set of holes of various sizes.

This procedure is a particular instance of the general **dynamic storageallocation problem**, which concerns how to satisfy a request of size *n* from a list of free holes. There are many

solutions to this problem. The **first-fit**, **best-fit**, and **worst-fit** strategies are the ones most commonly used to select a free hole from the set of available holes.

Memory Allocation Strategies

1. First Fit
2. Best fit
3. Worst fit
4. Next fit

First Fit

• **First fit**. Allocate the first hole that is big enough. Searching can start eitherat the beginning of the set of holes or at the location where the previousfirst-fit search ended. We can stop searching as soon as we find a free holethat is large enough.

In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

Advantage

Fastest algorithm because it searches as little as possible.

DisadvantageThe remaining unused memory areas left after allocation become waste if it is too smaller. Thus request for larger memory requirement cannot be accomplished.

First Fit

The first of these is called first fit. The basic idea with first fit allocation is that we begin searching the list and take the first block whose size is greater than or equal to the request size. If we reach the end of the list without finding a suitable block, then the request fails.

To illustrate the behavior of first fit allocation, as well as the other allocation policies later, we trace their behavior on a set of allocation and de-allocation requests. We denote this sequence as A20, A15, A10, A25, D20, D10, A8, A30, D15, A15, where An denotes an allocation request for n KB and Dn denotes a de-allocation request for the allocated block of size n KB. (For simplicity of notation, we have only one block of a given size allocated at a time. None of the policies depend on this property; it is used here merely for clarity.) In these examples, the memory space from which we serve requests is 128 KB. Each row of Figure 9-9 shows the state of memory after the operation labeling it on the left.

Shaded blocks are allocated and un-shaded blocks are free. The size of each block is shown in the corresponding box in the figure. In this, and other allocation figures in this chapter, time moves downward in the figure. In other words, each operation happens prior to the one below it.

A20	20	108					
A15	20	15	93				
A10	20	15	10	83			
A25	20	15	10	25	58		
D20	20	15	10	25	58		
D10	20	15	10	25	58		
A8	8	12	15	10	25	58	
A30	8	12	15	10	25	30	28
D15	8	37			25	30	28
A15	8	15	22		25	30	28

Fig 3.4. First Fit Allocation

Best Fit

• **Best fit**. Allocate the smallest hole that is big enough.Wemust search theentire list, unless the list is ordered by size. This strategy produces thesmallest leftover hole.

The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers

the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

Advantage

Memory utilization is much better than first fit as it searches the smallest free partition first available.

Disadvantage

It is slower and may even tend to fill up memory with tiny useless holes.

In many ways, the most natural approach is to allocate the free block that is closest insize to the request. This technique is called best fit. In best fit, we search the list for theblock that is smallest but greater than or equal to the request size.

As with the next fit example, we show only the final four steps of best fit allocation forour example.

A8	20	15	8	2	25	58		
A30	20	15	8	2	25	30	28	
D15	35		8	2	25	30	28	
A15	35		8	2	25	30	15	13

Fig 3.6. **Best Fit Allocation**

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

Worst fit

• **Worst fit**. Allocate the largest hole. Again, we must search the entire list,unless it is sorted by size. This strategy produces the largest leftover hole,which may be more useful than the smaller leftover hole from a best-fitapproach.

In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

Advantage

Reduces the rate of production of small gaps.

Disadvantage

If a process requiring larger memory arrives at a later stage then it cannot be accommodated as the largest hole is already split and occupied.

Worst Fit

If best fit allocates the smallest block that satisfies the request, then worst fit allocatesthe largest block for every request. Although the name would suggest that we wouldnever use the worst fit policy, it does have one advantage: If most of the requests are ofsimilar size, a worst fit policy tends to minimize external fragmentation.

A8	20	15	10	25	8	50	
A30	20	15	10	25	8	30	20
D15	45			25	8	30	20
A15	15	30		25	8	30	20

Fig 3.7. Worst fit allocation

Main memory usually has two partitions −

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

Operating system uses the following memory allocation mechanism.

1 Single-partition allocation

In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.

2 Multiple-partition allocation

In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks

considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types −

1 External fragmentation

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

2 Internal fragmentation

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

Fixed and Variable partitions

1. Fixed Partitioning: Main memory is divided into a no. of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.

Memory Manager will allocate a region to a process that best fits it

Unused memory within an allocated partition called internal fragmentation.

Advantages:

Simple to implement

Little OS overhead

Disadvantages:

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

Inefficient use of Memory due to internal fragmentation. Main memory utilization is extremely inefficient. Any program, no matter how small, occupies an entire partition. This phenomenon, in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition, is referred to as internal fragmentation.

Two possibilities:

a). Equal size partitioning

b). Unequal size Partition

Not suitable for systems in which process memory requirements not known ahead of time; i.e. timesharing systems

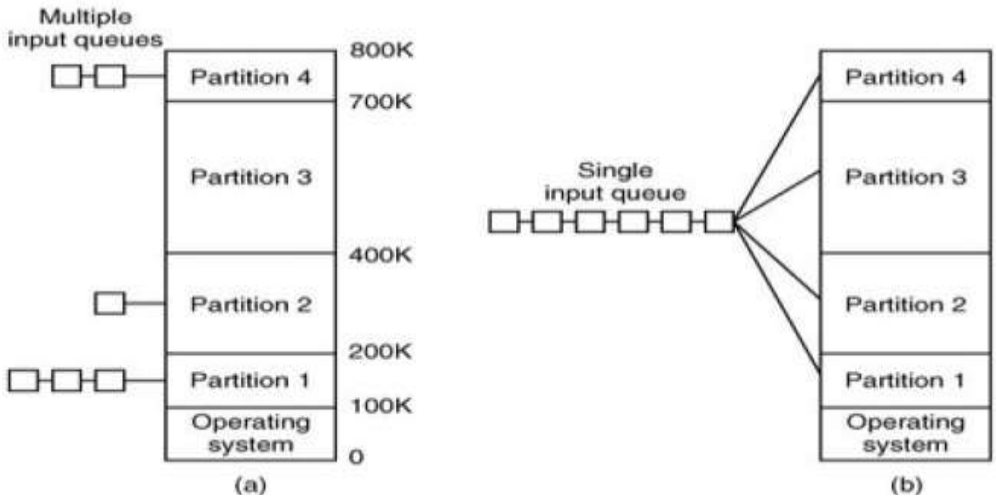


Fig 3.8. (a) Fixed memory partitions with separate input queues for each partition.

(b) Fixed memory partitions with a single input queue.

When the queue for a large partition is empty but the queue for a small partition is full, as is the case for partitions 1 and 3. Here small jobs have to wait to get into memory, even though plenty of memory is free.

An alternative organization is to maintain a single queue as in Fig. 4-2(b). Whenever a partition

becomes free, the job closest to the front of the queue that fits in it could be loaded into the empty partition and run.

2. Dynamic/Variable Partitioning:

To overcome some of the difficulties with fixed partitioning, an approach known as dynamic

partitioning was developed . The partitions are of variable length and number. When a process is

brought into main memory, it is allocated exactly as much memory as it requires and no more. An example, using 64 Mbytes of main memory, is shown in Figure Eventually it leads to a situation in which there are a lot of small holes in memory. As time goes on, memory

becomes more and more fragmented, and memory utilization declines. This phenomenon is referred to as **external fragmentation**, indicating that the memory that is external to all partitions becomes increasingly fragmented.

One technique for overcoming external fragmentation is compaction: From time to time, the operating system shifts the processes so that they are contiguous and so that all of the free memory is together in one block. For example, in Figure h, compaction will result in a block of free memory of length 16M.

This may well be sufficient to load in an additional process. The difficulty with compaction is that it is a time consuming procedure and wasteful of processor time.

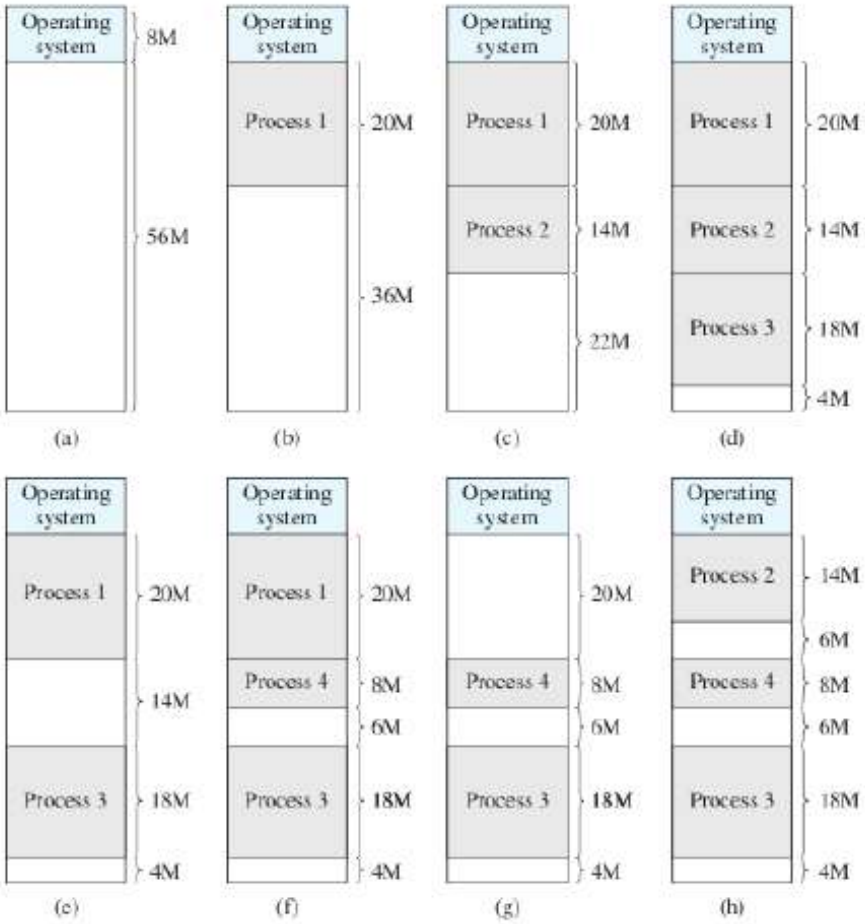


Fig.3.9. The Effect of dynamic partitioning

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to

emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

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

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

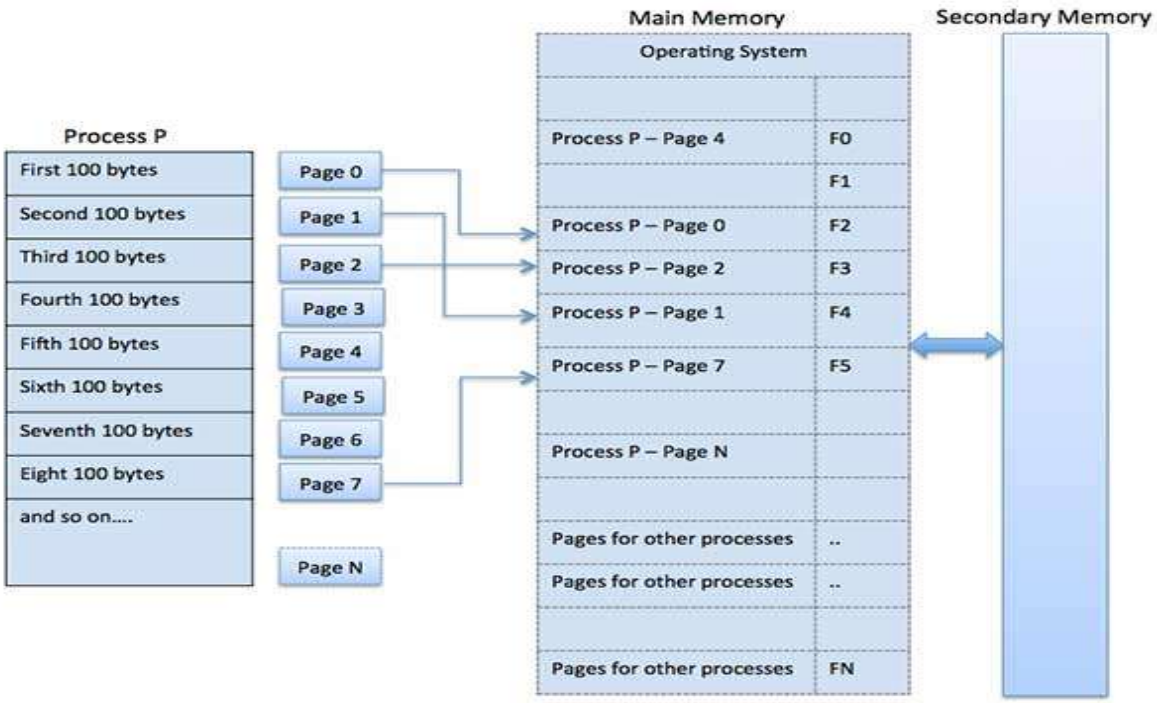


Fig 3.10. Paging Process

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

Address Translation

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

Logical Address = Page number + page offset

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

Physical Address = Frame number + page offset

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

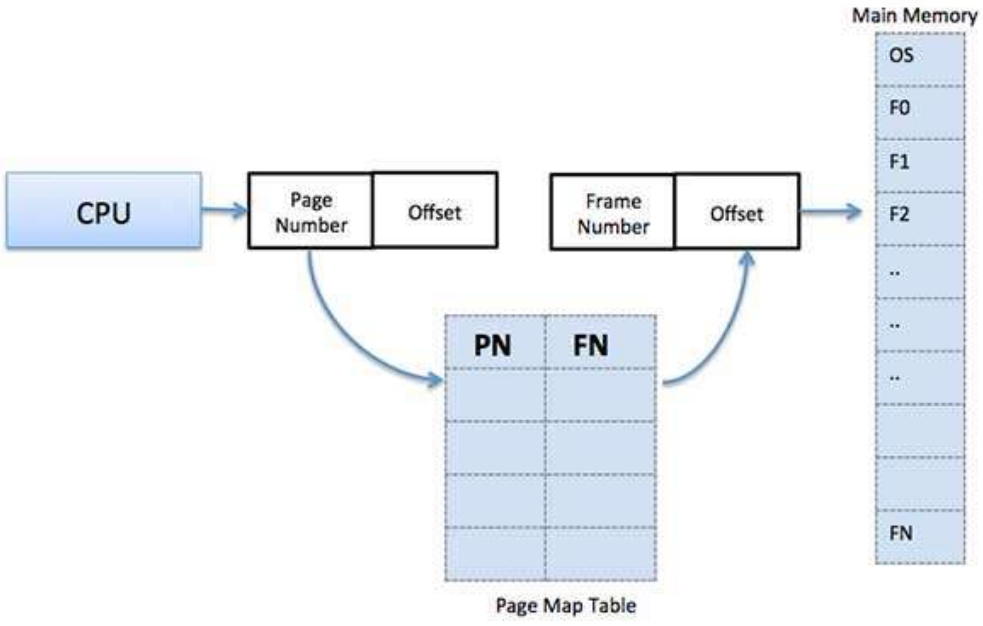


Fig 3.11. Paging with Page Table

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

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

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

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

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

Paging

External fragmentation is avoided by using paging technique. Paging is a technique in which physical memory is broken into blocks of the same size called pages (size is power of 2, between

512 bytes and 8192 bytes). When a process is to be executed, it's corresponding pages are loaded into any available memory frames.

Logical address space of a process can be non-contiguous and a process is allocated physical memory whenever the free memory frame is available. Operating system keeps track of all free frames. Operating system needs n free frames to run a program of size n pages.

Address generated by CPU is divided into

- **Page number (p)** -- page number is used as an index into a page table which contains base address of each page in physical memory.
- **Page offset (d)** -- page offset is combined with base address to define the physical memory address.

● Each virtual address space is divided into fixed-size chunks called pages.

● The physical address space is divided into fixed-size chunks called frames.

● Pages have same size as frames.

● The kernel maintains a page table (or page-frame table) for each process, specifying the frame within which each page is located.

● The CPU's memory management unit (MMU) translates virtual addresses to physical addresses on-the-fly for every memory access.

Properties:

● relatively simple to implement (in hardware);

● virtual address space need not be physically contiguous.

Basic Method

The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called **frames** and breaking logical memory into blocks of the same size called **pages**.

When a process is to be executed, its pages are loaded into any available memory frames from their source (a filesystem or the backing store). The backing store is divided into fixed-sizedblocks that are the same size as the memory frames or clusters of multipleframes. This rather simple idea has great functionality and wide ramifications.For example, the logical address space is now totally separate from the physicaladdress space, so a process can have a logical 64-bit address space even thoughthe system has less than 264 bytes of physical memory.

The hardware support for paging is illustrated. Every addressgenerated by the CPU is divided into two parts: a **page number (p)** and a **pageoffset (d)**. The page number is used as an index into a **page table**. The page tablecontains the base address of each page in physical memory. This base addressis combined with the page offset to define the physical memory address thatis sent to the memory unit. The paging model of memory is shown in Figure.

Following figure show the paging table architecture.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

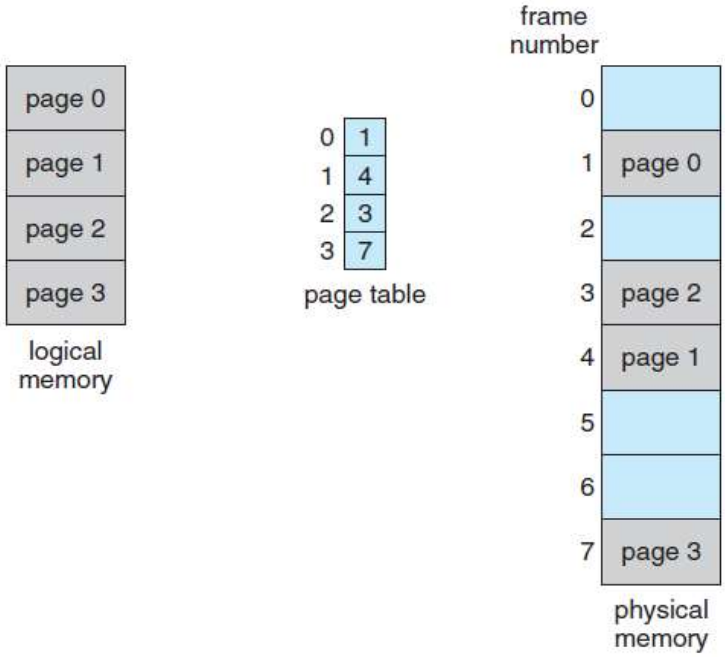
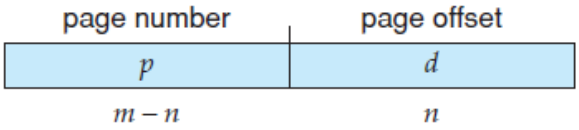


Fig 3.12. Page Table

The page size (like the frame size) is defined by the hardware. The size of apage is a power of 2, varying between 512 bytes and 1 GB per page, dependingon the computer architecture. The selection of a power of 2 as a page sizemakes the translation of a logical address into a page number and page offsetparticularly easy. If the size of the logical address space is 2*m*, and a page size is2*n* bytes, then the high-order *m*− *n* bits of a logical address designate the pagenumber, and the *n* low-order bits designate the page offset. Thus, the logicaladdress is as follows:



where *p* is an index into the page table and *d* is the displacement within thepage.

Segmentation

Segmentation is a memory-management scheme that supports this programmerview of memory. A logical address space is a collection of segments.

As we've already seen, the user's view of memory is not the same as the actualphysical memory. This is equally true of the programmer's view of memory.Indeed, dealing with memory in terms of its physical properties is inconvenientto both the operating system and the programmer. What if the hardware couldprovide a memory mechanism that mapped the programmer's view to theactual physical memory? The system would have more freedom to managememory, while the programmer would have a more natural programmingenvironment. Segmentation provides such a mechanism.

Basic Method Do programmers think of memory as a linear array of bytes, some containinginstructions and others containing data? Most programmers would say "no."Rather, they prefer to view memory as a collection of variable-sized segments,with no necessary ordering among the segments. When writing a program, a programmer thinks of it as a main programwith a set of methods, procedures, or functions. Itmayalso include various datastructures: objects, arrays, stacks, variables, and so on. Each of these modules ordata elements is referred to by name. The programmer talks about "the stack,""the math library," and "the main program" without caring what addressesin memory these elements occupy. She is not concerned with whether thestack is stored before or after the Sqrt() function. Segments vary in

length,and the length of each is intrinsically defined by its purpose in the program.Elements within a segment are identified by their offset from the beginning ofthe segment: the first statement of the program, the seventh stack frame entryin the stack, the fifth instruction of the Sqrt(), and so on.

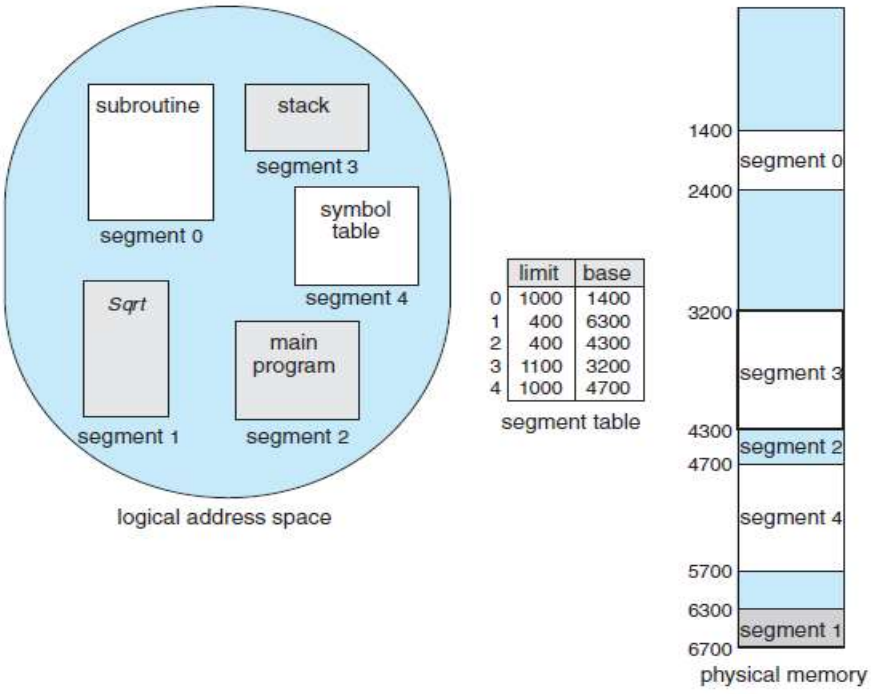


Fig 3.13. Segmentation

Each segment has a name and a length. The addresses specify both the segmentname and the offset within the segment. The programmer therefore specifieseach address by two quantities: a segment name and an offset.For simplicity of implementation, segments are numbered and are

referredto by a segment number, rather than by a segment name. Thus, a logical addressconsists of a *two tuple:*<segment-number, offset>.

Normally, when a program is compiled, the compiler automatically constructs

segments reflecting the input program.

A C compiler might create separate segments for the following:

1. The code

2. Global variables

3. The heap, from which memory is allocated

4. The stacks used by each thread

5. The standard C library

Libraries that are linked in during compile time might be assigned separatesegments. The loader would take all these segments and assign them segmentnumbers.

Segmentation

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

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

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

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

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

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

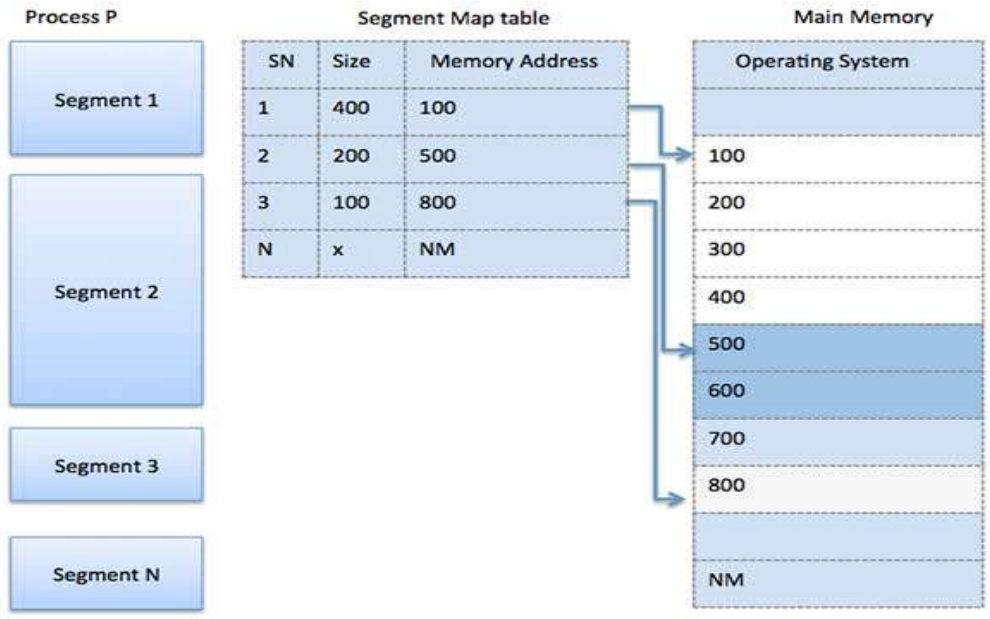


Fig 3.14. Segmentation Process

Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

Benefits of having Virtual Memory :

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

Demand Paging

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them(On demand). This is termed as lazy swapper, although a pager is a more accurate term.

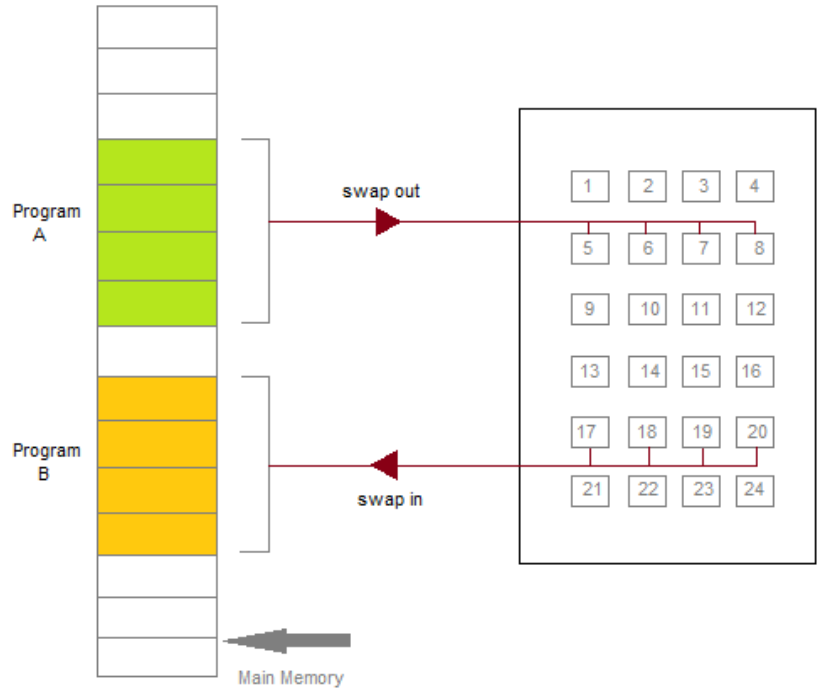


Fig 3.15. Demand Paging Process

Initially only those pages are loaded which will be required the process immediately.

The pages that are not moved into the memory, are marked as invalid in the page table. For an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.

2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.
6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. Its is not a big issue for small programs, but for larger programs it affects performance drastically.

Page Replacement

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.

2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Basic Page Replacement

- Find the location of the page requested by ongoing process on the disk.
- Find a free frame. If there is a free frame, use it. If there is no free frame, use a page-replacement algorithm to select any existing frame to be replaced, such frame is known as **victim frame**.
- Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
- Move the required page and store it in the frame. Adjust all related page and frame tables to indicate the change.
- Restart the process that was waiting for this page.

FIFO Page Replacement

- A very simple way of Page replacement is FIFO (First in First Out)
- As new pages are requested and are swapped in, they are added to tail of a queue and the page which is at the head becomes the victim.
- Its not an effective way of page replacement but can be used for small systems.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: III (Memory Management)**

SEMESTER IV **BATCH (2018-2021)**

LRU Page Replacement

FIFO algorithm uses the time when a page was brought into memory, whereasthe OPT algorithm uses the time when a page is to be ***used.*** If we use the recentpast as an approximation of the near future, then we can replace the page that***has not been used*** for the longest period of time. This approach is the **least recently used (LRU) algorithm**.

POSSIBLE QUESTIONS
(2 MARKS)

1. What is Paging?
2. What is a Physical address space?
3. What is Segmentation?
4. What is Virtual memory?
5. What is Fixed partition?

(6 MARKS)

1. Write about Memory Allocation strategies with neat sketch.
2. Explain the concept of Virtual memory in detail
3. Write about Physical and Virtual address space in detail.
4. Explain in detail about Segmentation with neat sketch.
5. Discuss about Paging in detail.
6. Write about Fixed and Variable partitions in detail.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021

(For the candidates admitted in 2018 onwards)

Unit III

CLASS : II B.Sc CT Semester:IV

SUBJECT : Operating Systems Batch:2018-2021

No.	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	A _____ is executed until it must wait, typically for the completion of some i/o request	reverse	deadlock avoidance	deadlock	process	process
2	_____ is a fundamental operating system function.	RR	CPU	Scheduling	nonpreemptive	Scheduling
3	Process execution begins with a_____	CPU burst	RR scheduling	SJF scheduling	SRT scheduling	CPU burst
4	The operating system must select one of the processes in the ready queue to be executed by the _____	nonpreemptive	short term scheduler	long term scheduler	low level	short term scheduler
5	When scheduling takes place only under circumstances 1 and 4 called _____	variable class	real time class	priority class	nonpreemptive	nonpreemptive
6	Another component involved in the CPU scheduling function is the _____	central edge	dispatcher	claim edge	graph edge	dispatcher
7	One measure of work is the number of processes completed per time unit called _____	throughput	variable class	real time class	priority class	throughput
8	Which of the following is the simplest scheduling discipline?	FIFO scheduling	RR scheduling	SJF scheduling	SRT scheduling	FIFO scheduling

9	In which scheduling, processes are dispatched according to their arrival time on the ready queue?	FIFO scheduling	RR scheduling	SJF scheduling	SRT scheduling	FIFO scheduling
10	In which scheduling, processes are dispatched FIFO but are given a limited amount of CPU time?	FIFO scheduling	RR scheduling	SJF scheduling	SRT scheduling	RR scheduling
11	Which scheduling is effective in time sharing environments	FIFO scheduling	RR scheduling	SJF scheduling	SRT scheduling	RR scheduling
12	Variable size blocks are called	Pages	Segments	Tables	None	Segments
13	Which scheduling is effective in time sharing environments	FIFO scheduling	RR scheduling	SJF scheduling	SRT scheduling	RR scheduling
14	Which of the following is non-preemptive scheduling?	RR scheduling	SJF scheduling	SRT scheduling	None	SJF scheduling
15	The interval from the time of submission of a process to the time of completion is the ____	Queues	Processor Sharing	Sharing resources	turaround time	turaround time
16	The simplest CPU sceduling algorithm is the	FCS	SJS	FCFS	DFG	FCFS
17	Multiprogramming was made possible by _____.	input/output units that operate independently of the CPU	operating systems	both (a) & (b) above	neither (a) or (b) above	both (a) & (b) above
18	The SJF algorithm is a special case of the general _____ algorithm	FCS	SJS	Roundrobin	FCSC	Round Robin
19	_____ scheduling algorithm is designed especially for time sharing systems.	CFS	FSCS	priority	Round Robin	Round Robin
20	The seek optimization strategy in which there is no reordering of the queue is called _____.	FCFS	SSTF	SCAN	C-SCAN	FCFS

21	A major problem with priority scheduling algorithms is _____	tail	Starvation	time first	time quantum	Starvation
22	If the time quantum is very small the RR aproach is called _____	Queues	Processor Sharing	Sharing resources	Context switching	Context switching
23	The seek optimization strategy in which the disk arm is positioned next at the request (inward or outward) that minimizes arm movement is called _____.	FCFS	SSTF	SCAN	C-SCAN	SSTF
24	If several identical processors are available then _____ can occur.	heterogeneous	homogeneous	load sharing	UMA	heterogeneous
25	The high priority process would be waiting for a lower priority one to finish is called _____	resources inversion	Priority inversion	priority	Priority inheritance	Priority inversion
26	_____ systems are required to complete a critical task within a guaranteed amount of time.	hard real time	Priority inversion	load sharing	Priority inheritance	hard real time
27	The scheduler than either admits a process guarenteeing that the process will complete on time known as _____	Priority inversion	resources reservation	load sharing	Sharing resources	resources reservation
28	_____ uses the given algorithm and the system workload to produce a formula.	deterministic modelling	scheduling process	Analaytic evaluation	Queuing model	deterministic modelling
29	If no thread is found the dispatcher will execute a special thread called_____	variable class	real time class	priority class	idle thread	idle thread

30	Deadlocks can be described more precisely in terms of a directed graph called	resource graph	system graph	system resources allocation graph	request graph	system resources allocation graph
31	_____ is th set of methods for ensuring that at atleast one of the necessary condition.	Deadlock prevention	deadlock avoidance	handling deadlock	resource deadlock	Deadlock prevention
32	_____ is possible to construct an algorithm that ensures that the system will never enter the deadlock state.	Deadlock prevention	deadlock avoidance	handling deadlock	resource deadlock	deadlock avoidance
33	A system is in a safe state only if there exists a _____	Safe state	unsafe state	normal	deadlock	Safe state
34	A critical section is a program segment _____.	which should run in a certain specified amount	which avoids deadlocks	where shared resources are accessed	which must be enclosed by a pair of semaphore operations, P and V	where shared resources are accessed
35	The deadlock avoidance algorithm are described in next system but is less efficient than the resource allocation graph called _____	Deadlock prevention	deadlock avoidance	bankers algorithm	bankers allocation	bankers algorithm
36	A semaphore	Is a binary mutex	Must be accessed from only one process	Can be accessed from multiple processes	Can be accessed from two processes	Must be accessed from only one process
37	A direct method of deadlock prevention is to prevent the occurrences of _____	Mutual exclusion	Hold and wait	Circular waits	No preemption	Circular waits

38	_____ is a lightweight process where the context switching is low.	Process	Thread	Kernel	Minikernel	Thread
39	Each process accessing the shared data excludes all the others from doing so simultaneously	Mutual exclusion	Deadlock prevention	Preemption	Circular Wait	Mutual exclusion
40	The structure of parent - child process known as	Master - Slave structure	Parent- Children structure	Hierarchical structure	All the above	All the above
41	. _____ refers to the ability of an operating system to support multiple threads of	Multithreading	Multiprocessing	Multiexecuting	Bi-threading	Multithreading
42	_____ semaphore can assume only the value 0 or 1	Binary	Counting	Operations	Normal	Binary

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: IV (File and I/O Management)**

SEMESTER IV **BATCH (2018-2021)**

SYLLABUS

File and I/O Management-Directory structure-File Operations – File allocation methods-Device management-Device Drivers

No general-purpose computer stores just one file. There are typically thousands, millions, even billions of files within a computer. Files are stored on random-access storage devices, including hard disks, optical disks, and solid-state (memory-based) disks. A storage device can be used in its entirety for a file system. It can also be subdivided for finer-grained control. For example, a disk can be **partitioned** into quarters, and each quarter can hold a separate file system.

A file system can be created on each of these parts of the disk. Any entity containing a file system is generally known as a **volume**. Each volume can be thought of as a virtual disk. Volumes can also store multiple operating systems, allowing a system to boot and run more than one operating system.

Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a **device directory** or **volume table of contents**. The device directory (more commonly known simply as the **directory**) records information—such as name, location, size, and type—for all files on that volume.

Directory Overview

The directory can be viewed as a symbol table that translates file names into their directory entries. If we take such a view, we see that the directory itself can be organized in many ways. The organization must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory. In this section, we examine several schemes for defining the logical structure of the directory system. When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory:

• **Search for a file**. We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names may indicate a relationship among files, we may want to be able to find all files whose names match a particular pattern. • **Create a file**. New files need to be created and added to the directory.

• **Delete a file**. When a file is no longer needed, we want to be able to remove it from the directory.

• **List a directory**. We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.

• **Rename a file**. Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.

• **Traverse the file system**. We may wish to access every directory and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals. Often, we do this by copying all files to magnetic tape. This technique provides a backup copy in case of system failure. In addition, if a file is no longer in use, the file can be copied to tape and the disk space of that file released for reuse by another file.

Directory structure

In the following sections, we describe the most common schemes for defining the logical structure of a directory.

Single-Level Directory

The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand.

A single-level directory has significant limitations, however, when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names.

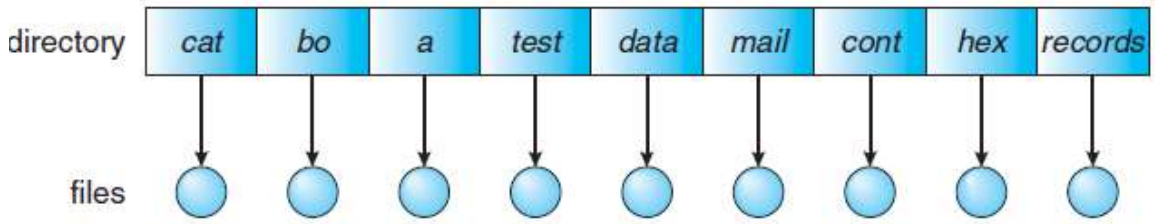


Fig 4.1.Single-level directory.

If two users call their data file test.txt, then the unique-name rule is violated.

For example, in one programming class, 23 students called the program for their second assignment prog2.c; another 11 called it assign2.c. Fortunately, most file systems support file names of up to 255 characters, so it is relatively easy to select unique file names.

Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases. It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system.

Keeping track of so many files is a daunting task.

Two-Level Directory

As we have seen, a single-level directory often leads to confusion of file names among different users. The standard solution is to create a separate directory for each user.

In the two-level directory structure, each user has his own**user file directory (UFD)**. The UFDs have similar structures, but each lists only the files of a single user.

When a user job starts or a user logs in, the system's **master file directory (MFD)** is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user. When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.

To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.

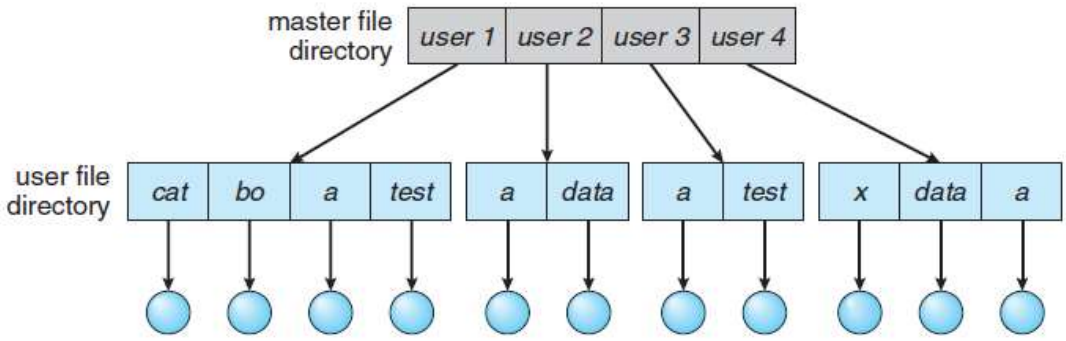


Fig 4.2.Two-level directory structure.

The user directories themselves must be created and deleted as necessary. **A special system program** is run with the appropriate user name and account information. The program creates a

new UFD and adds an entry for it to the MFD. The execution of this program might be restricted to system administrators. The allocation of disk space for user directories can be handled with the techniques for files themselves.

Although the two-level directory structure solves the name-collision problem, it still has disadvantages. This structure effectively isolates one user from another. **Isolation** is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files. Some systems simply do not allow local user files to be accessed by other users.

If access is to be permitted, one user must have the ability to name a file in another user's directory. To name a particular file uniquely in a two-level directory, we must give both the user name and the file name. A two-level directory can be thought of as a tree, or an inverted tree, of height 2. The root of the tree is the MFD. Its direct descendants are the UFDs. The descendants of the UFDs are the files themselves. The files are the leaves of the tree. Specifying a user name and a file name defines a path in the tree from the root (the MFD) to a leaf (the specified file). Thus, a user name and a file name define a **path name**. Every file in the system has a path name.

To name a file uniquely, a user must know the path name of the file desired. For example, if user A wishes to access her own test file named test.txt, she can simply refer to test.txt. To access the file named test.txt of user B (with directory-entry name userb), however, she might have to refer to /userb/test.txt. Every system has its own syntax for naming files in directories other than the user's own. Additional syntax is needed to specify the volume of a file. For instance, in Windows a volume is specified by a letter followed by a colon. Thus, a file specification might be C:\userb\test.

Whenever a file name is given to be loaded, the operating system first searches the local UFD. If the file is found, it is used. If it is not found, the system automatically searches the special user directory that contains the system files. The sequence of directories searched when a file is named is called the **search path**.

The search path can be extended to contain an unlimited list of directories to search when a command name is given. This method is the one most used in UNIX and Windows. Systems can also be designed so that each user has his own search path.

Tree-Structured Directories

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: IV (File and I/O Management)**

SEMESTER IV **BATCH (2018-2021)**

Once we have seen how to view a two-level directory as a two-level tree, the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows users to create their own subdirectories and to organize their files accordingly.

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

A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way.

All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).

Special system calls are used to create and delete directories.

In normal use, each process has a current directory. The **current directory** should contain most of the files that are of current interest to the process. When reference is made to a file, the current directory is searched. If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file.

To change directories, a system call is provided that takes a directory name as a parameter and uses it to redefine the current directory. Thus, the user can change her current directory whenever she wants.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: IV (File and I/O Management)**

SEMESTER IV **BATCH (2018-2021)**

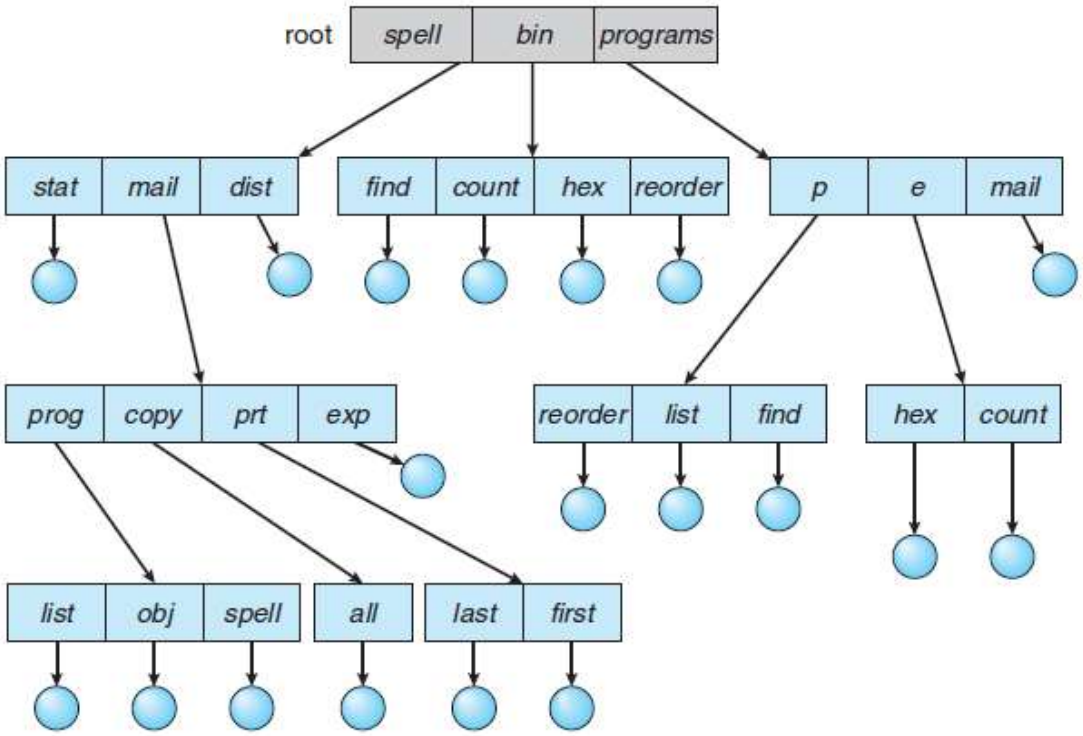


Fig 4.3.Tree-structured directory structure.

From **one change directory() system call to the next, all open() system calls search the current directory for the specified file.** Note that the search path may or may not contain a special entry that stands for "the current directory." The initial current directory of a user's login shell is designated when the user job starts or the user logs in.

The current directory of any subprocess is usually the current directory of the parent when it was generated. Path names can be of two types: absolute and relative.

An **absolute path name** begins at the root and follows a path down to the specified file, giving the directory names on the path. A **relative path name** defines a path from the current directory.

For example, in the tree-structured file system, if the current directory is root/spell/mail, then the relative path name prt/first refers to the same file as does the absolute path name root/spell/mail/prt/first. Allowing a user to define her own subdirectories permits her to impose a structure on her files.

This structure might result in separate directories for files associated with different topics (for example, a subdirectory was created to hold the text of this book) or different forms of information (for example, the directory programs may contain source programs; the directory bin may store all the binaries).

An interesting policy decision in a tree-structured directory concerns **how to handle the deletion of a directory**. If a directory is empty, its entry in the directory that contains it can simply be deleted. However, suppose the directory to be deleted is not empty but contains several files or subdirectories. One of two approaches can be taken. Some systems will not delete a directory unless it is empty. Thus, to delete a directory, the user must first delete all the files in that directory. If any subdirectories exist, this procedure must be applied recursively to them, so that they can be deleted also.

This approach can result in a substantial amount of work. An alternative approach, such as that taken by the UNIX rm command, is to provide an option: when a request is made to delete a directory, all that directory's files and subdirectories are also to be deleted. Either approach is fairly easy to implement; the choice is one of policy.

Acyclic-Graph Directories

Consider two programmers who are working on a joint project. The files associated with that project can be stored in a subdirectory, separating them from other projects and files of the two programmers. But since both programmers are equally responsible for the project, both want the subdirectory to be in their own directories. In this situation, the common subdirectory should be ***shared.*** A shared directory or file exists in the file system in two (or more) places at once.

A tree structure prohibits the sharing of files or directories.

An **acyclic graph** —that is, a graph with no cycles—allows directories to share subdirectories and files. The same file or subdirectory may be in two different directories. The acyclic graph is

a natural generalization of the tree-structured directory scheme. It is important to note that a shared file (or directory) is not the same as two copies of the file.

With two copies, each programmer can view the copy rather than the original, but if one programmer changes the file, the changes will not appear in the other's copy.

With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other. Sharing is particularly important for subdirectories; a new file created by one person will automatically appear in all the shared subdirectories. When people are working as a team, all the files they want to share can be put into one directory. The UFD of each team member will contain this directory of shared files as a subdirectory.

Even in the case of a single user, the user's file organization may require that some file be placed in different subdirectories. For example, a program written for a particular project should be both in the directory of all programs and in the directory for that project. Shared files and subdirectories can be implemented in several ways. A common way, exemplified by many of the UNIX systems, is to create a new directory entry called a link. A **link** is effectively a pointer to another file or subdirectory. For example, a link may be implemented as an absolute or a relative path name.

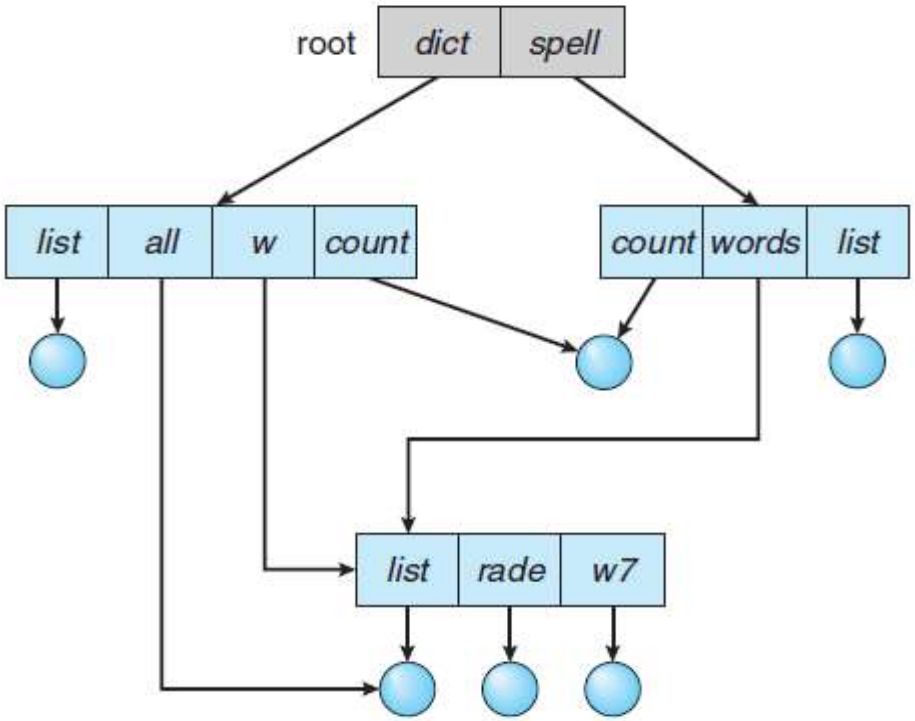


Fig 4.4.Acyclic-graph directory structure.

When a reference to a file is made, we search the directory. If the directory entry is marked as a link, then the name of the real file is included in the link information.

We **resolve** the link by using that path name to locate the real file. Links are easily identified by their format in the directory entry (or by having a special type on systems that support types) and are effectively indirect pointers. The operating system ignores these links when traversing directory trees to preserve the acyclic structure of the system.

Another common approach to implementing shared files is simply to duplicate all information about them in both sharing directories. Thus, both entries are identical and equal. Consider the difference between this approach and the creation of a link. The link is clearly different from the original directory entry; thus, the two are not equal.

The deletion of a link need not affect the original file; only the link is removed. If the file entry itself is deleted, the space for the file is deallocated, leaving the links dangling.We can search for

these links and remove them as well, but unless a list of the associated links is kept with each file, this search can be expensive.

General Graph Directory

A serious problem with using an acyclic-graph structure is ensuring that there are no cycles. If we start with a two-level directory and allow users to create subdirectories, a tree-structured directory results. It should be fairly easy to see that simply adding new files and subdirectories to an existing tree-structured directory preserves the tree-structured nature. However, when we add links, the tree structure is destroyed, resulting in a simple graph structure.

If we have just searched a major shared subdirectory for a particular file without finding it, we want to avoid searching that subdirectory again; the second search would be a waste of time.

If cycles are allowed to exist in the directory, we likewise want to avoid searching any component twice, for reasons of correctness as well as performance. A poorly designed algorithm might result in an infinite loop continually searching through the cycle and never terminating. One solution is to limit arbitrarily the number of directories that will be accessed during a search.

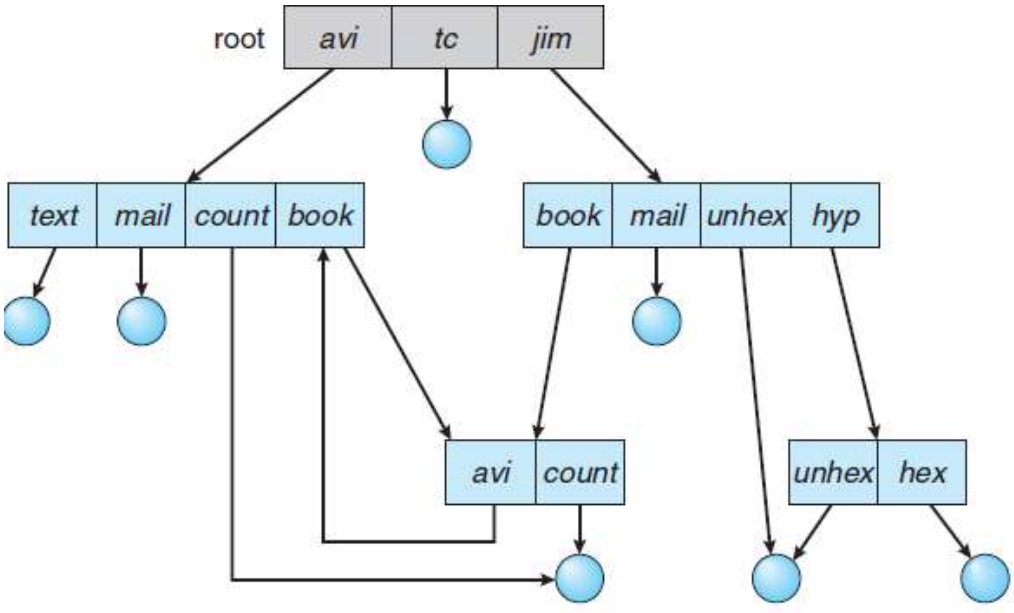


Fig 4.5.General graph directory.

A similar problem exists when we are trying to determine when a file can be deleted.

With acyclic-graph directory structures, a value of 0 in the reference count means that there are no more references to the file or directory, and the file can be deleted.

However, when cycles exist, the reference count may not be 0 even when it is no longer possible to refer to a directory or file. This anomaly results from the possibility of self-referencing (or a cycle) in the directory structure.

In this case, we generally need to use a **garbage collection** scheme to determine when the last reference has been deleted and the disk space can be reallocated. Garbage collection involves traversing the entire file system, marking everything that can be accessed.

Thus, an acyclic-graph structure is much easier to work with. The difficulty is to avoid cycles as new links are added to the structure. How do we know when a new link will complete a cycle? There are algorithms to detect cycles in graphs; however, they are computationally expensive, especially when the graph is on disk storage. A simpler algorithm in the special case of directories and links is to bypass links during directory traversal. Cycles are avoided, and no extra overhead is incurred.

File Operations

A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files. The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.

Files exist to store information and allow it to be retrieved later. Different systems provide different operations to allow storage and retrieval. Below is a discussion of the most common system calls relating to files.

1. **Create.** The file is created with no data. The purpose of the call is to announce that the file is coming and to set some of the attributes.

2. Delete. When the file is no longer needed, it has to be deleted to free up disk space. There is always a system call for this purpose.
3. Open. Before using a file, a process must open it. The purpose of the **open** call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.
4. Close. When all the accesses are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up internal table space. Many systems encourage this by imposing a maximum number of open files on processes.
A disk is written in blocks, and closing a file forces writing of the file's last block, even though that block may not be entirely full yet. 5. Read. Data are read from file. Usually, the bytes come from the current position. The caller must specify how many data are needed and must also provide a buffer to put them in.
6. Write. Data are written to the file again, usually at the current position. If the current position is the end of the file, the file's size increases. If the current position is in the middle of the file, existing data are overwritten and lost forever.
7. Append. This call is a restricted form of write. It can only add data to the end of the file. Systems that provide a minimal set of system calls do not generally have append, but many systems provide multiple ways of doing the same thing, and these systems sometimes have append.
8. Seek. For random access files, a method is needed to specify from where to take the data. One common approach is a system call, seek, that repositions the file pointer to a specific place in the file. After this call has completed, data can be read from, or written to, that position.
9. Get attributes. Processes often need to read file attributes to do their work. For example, the UNIX *make* program is commonly used to manage software development projects consisting of many source files. When *make* is called, it examines the modification times of all the source and object files and arranges for the minimum number of compilations required to bring everything up to date. To do its job, it must look at the attributes, namely, the modification times.
10. Set attributes. Some of the attributes are user settable and can be changed after the file has been created. This system call makes that possible. The protection mode information is an obvious example. Most of the flags also fall in this category.
11. Rename. It frequently happens that a user needs to change the name of an existing file. This system call makes that possible. It is not always strictly necessary, because the file can usually be copied to a new file with the new name, and the old file then deleted.

• **Creating a file**. Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
• **Writing a file**. To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the

directory to find the file's location. The system must keep a **write pointer** to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

• **Reading a file**. To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a **read pointer** to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process **currentfile- position pointer**. Both the read and write operations use this same pointer, saving space and reducing system complexity.

• **Repositioning within a file**. The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file **seek**.

• **Deleting a file**. To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.

• **Truncating a file**. The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

These six basic operations comprise the minimal set of required file operations. Other common operations include appending new information to the end of an existing file and renaming an existing file.

These primitive operations can then be combined to perform other file operations. For instance, we can create a copy of a file—or copy the file to another I/O device, such as a printer or a display—by creating a new file and then reading from the old and writing to the new. Most of the file operations mentioned involve searching the directory for the entry associated with the named file. To avoid this constant searching, many systems require that an open() system call be made before a file is first used. The operating system keeps a table, called the **open-file table**, containing information about all open files. When a file operation is requested, the file is specified via an index into this table, so no searching is required. When the file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table. create() and delete() are system calls that work with closed rather than open files.

In summary, several pieces of information are associated with an open file.

• **File pointer**. On systems that do not include a file offset as part of the read() and write() system calls, the system must track the last read– write location as a current-file-position pointer. This pointer is unique to each process operating on the file and therefore must be kept separate from the on-disk file attributes.

• **File-open count**. As files are closed, the operating system must reuse its open-file table entries, or it could run out of space in the table. Multiple processes may have opened a file, and the system must wait for the last file to close before removing the open-file table entry. The file-open count tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry.

• **Disk location of the file**.Most file operations require the systemtomodify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.

• **Access rights**. Each process opens a file in an accessmode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O requests. Some operating systems provide facilities for locking an open file (or sections of a file). File locks provide functionality similar to reader–writer locks, covered in

A **shared lock** is akin to a reader lock in that several processes can acquire the lock concurrently. An **exclusive lock** behaves like a writer lock; only one process at a time can acquire such a lock. It is important to note that not all operating systems provide both types of locks: some systems only provide exclusive file locking.

Furthermore, operating systems may provide either **mandatory** or **advisory** file-locking mechanisms. If a lock is mandatory, then once a process acquires an exclusive lock, the operating system will prevent any other process from accessing the locked file.

File Allocation Methods

The direct-access nature of disks gives us flexibility in the implementation of files. In almost every case, many files are stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.

Three major methods of allocating disk space are in wide use: contiguous, linked, and indexed. Each method has advantages and disadvantages.

Although some systems support all three, it is more common for a system to use one method for all files within a file-system type.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: IV (File and I/O Management)**

SEMESTER IV **BATCH (2018-2021)**

Contiguous Allocation

Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessing the disk, accessing block$b + 1$ after block b normally requires no head movement.

When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head need only move from one track to the next. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed. Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b, then it occupies blocks $b, b + 1, b + 2, ..., b + n - 1$.

The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file (Figure 12.5). Accessing a file that has been allocated contiguously is easy. For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block. For direct access to block i of a **554**

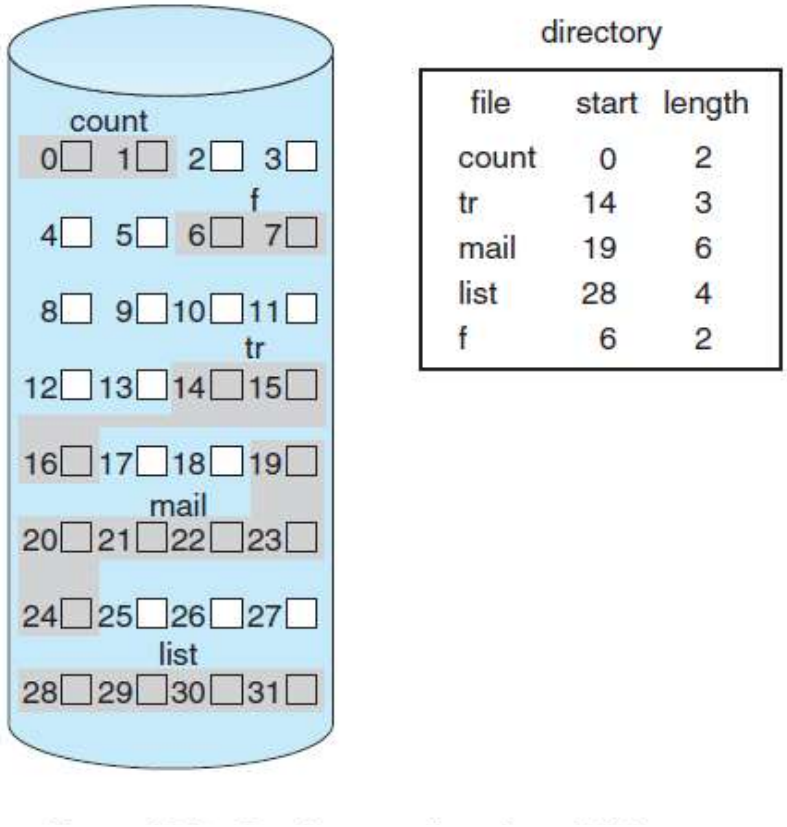


Fig 4.6. Contiguous allocation of disk space

file that starts at block *b*, we can immediately access block *b* + *i.* Thus, both sequential and direct access can be supported by contiguous allocation. Contiguous allocation has some problems, however. One difficulty is finding space for a new file. The system chosen to manage free space determines how this task is accomplished;

Any management system can be used, but some are slower than others. The contiguous-allocation problem can be seen as a particular application of the general **dynamic storage-allocation** problem which involves how to satisfy a request of size *n* from a list of free holes.

First fit and best fit are the most common strategies used to select a free hole from the set of available holes. Simulations have shown that both first fit and best fit are more efficient than worst fit in terms of both time and storage utilization. Neither first fit nor best fit is clearly best in

terms of storage utilization, but first fit is generally faster. All these algorithms suffer from the problem of **external fragmentation**.

As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists whenever free space is broken into chunks. It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, none of which is large enough to store the data.

Depending on the total amount of disk storage and the average file size, external fragmentation may be a minor or a major problem. One strategy for preventing loss of significant amounts of disk space to external fragmentation is to copy an entire file system onto another disk. The original disk is then freed completely, creating one large contiguous free space. We then copy the files back onto the original disk by allocating contiguous space from this one large hole. This scheme effectively **compacts**all free space into one contiguous space, solving the fragmentation problem. The cost of this compaction is time, however, and the cost can be particularly high for large hard disks. Compacting these disks may take hours and may be necessary on a weekly basis.

Some systems require that this function be done **off-line**, with the file system unmounted. During this **down time**, normal systemoperation generally cannot be permitted, so such compaction is avoided at all costs on production machines. Most modern systems that need defragmentation can perform it **on-line** during normal system operations, but the performance penalty can be substantial.

Another problem with contiguous allocation is determining how much space is needed for a file. When the file is created, the total amount of space it will need must be found and allocated. How does the creator (program or person) know the size of the file to be created? In some cases, this determination may be fairly simple (copying an existing file, for example). In general, however, the size of an output file may be difficult to estimate. If we allocate too little space to a file, we may find that the file cannot be extended. Especially with a best-fit allocation strategy, the space on both sides of the file may be in use. Hence, we cannot make the file larger in place.

Two possibilities then exist. First, the user program can be terminated, with an appropriate error message. The user must then allocate more space and run the program again. These repeated runs may be costly. To prevent them, the user will normally overestimate the amount of space needed, resulting in considerable wasted space. The other possibility is to find a larger hole, copy the contents of the file to the new space, and release the previous space. This series of actions can be repeated as long as space exists, although it can be time consuming.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: IV (File and I/O Management)**

SEMESTER IV **BATCH (2018-2021)**

Linked Allocation

Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

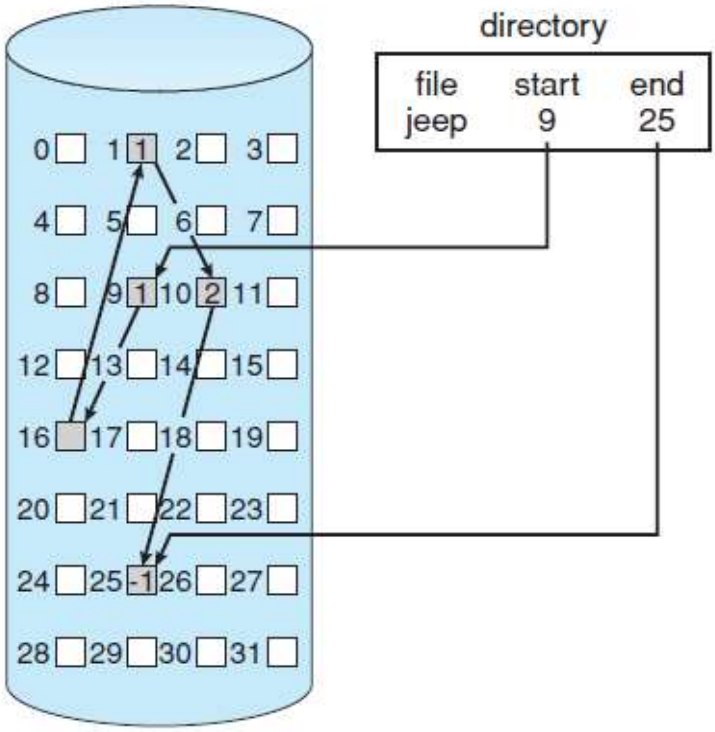


Fig 4.7.Linked allocation of disk space.

For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25 . Each block contains a pointer to the next block. These pointers are not made available to the user. Thus, if each block is 512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes. To create a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file.

This pointer is initialized to null (the end-of-list pointer value) to signify an empty file. The size field is also set to 0. A write to the file causes the free-space management system to find a free block, and this new block is written to and is linked to the end of the file.

To read a file, we simply read blocks by following the pointers from block to block. There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request. The size of a file need not be declared when the file is created. A file can continue to grow as long as free blocks are available. Consequently, it is never necessary to compact disk space. Linked allocation does have disadvantages, however. The major problem is that it can be used effectively only for sequential-access files.

To find the *i*th block of a file, we must start at the beginning of that file and follow the pointers until we get to the *i*th block. Each access to a pointer requires a disk read, and some require a disk seek. Consequently, it is inefficient to support a direct-access capability for linked-allocation files. Another disadvantage is the space required for the pointers. If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.

Each file requires slightly more space than it would otherwise. The usual solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate clusters rather than blocks. For instance, the file system may define a cluster as four blocks and operate on the disk only in cluster units. Pointers then use a much smaller percentage of the file's disk space. This method allows the logical-to-physical block mapping to remain simple but improves disk throughput (because fewer disk-head seeks are required) and decreases the space needed for block allocation and free-list management.

The cost of this approach is an increase in internal fragmentation, because more space is wasted when a cluster is partially full than when a block is partially full. Clusters can be used to improve the disk-access time for many other algorithms as well, so they are used in most file systems. Yet another problem of linked allocation is reliability. Recall that the files are linked together by pointers scattered all over the disk, and consider what would happen if a pointer were lost or damaged.

A bug in the operating-system software or a disk hardware failure might result in picking up the wrong pointer. This error could in turn result in linking into the free-space list or into another file. One partial solution is to use doubly linked lists, and another is to store the file name and

relative block number in each block. However, these schemes require even more overhead for each file. An important variation on linked allocation is the use of a **file-allocation table (FAT)**.

This simple but efficient method of disk-space allocation was used by the MS-DOS operating system. A section of disk at the beginning of each volume is set aside to contain the table. The table has one entry for each disk block and is indexed by block number. The FAT is used in much the same way as a linked list. The directory entry contains the block number of the first block of the file. The table entry indexed by that block number contains the block number of the next block in the file.

This chain continues until it reaches the last block, which has a special end-of-file value as the table entry. An unused block is indicated by a table value of 0. Allocating a new block to a file is a simple matter of finding the first 0-valued table entry and replacing the previous end-of-file value with the address of the new block. The 0 is then replaced with the end-of-file value. An illustrative example is the FAT structure shown in Figure 12.7 for a file consisting of disk blocks 217, 618, and 339. The FAT allocation scheme can result in a significant number of disk head seeks, unless the FAT is cached. The disk head must move to the start of the volume to read the FAT and find the location of the block in question, then move to the location of the block itself. In the worst case, both moves occur for each of the blocks. A benefit is that random-access time is improved, because the disk head can find the location of any block by reading the information in the FAT.

Indexed Allocation

Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation. However, in the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order. **Indexed allocation** solves this problem by bringing all the pointers together into one location: the **index block**. Each file has its own index block, which is an array of disk-block addresses. The *ith*entry in the index block points to the *ith*block of the file.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: IV (File and I/O Management)**

SEMESTER IV **BATCH (2018-2021)**

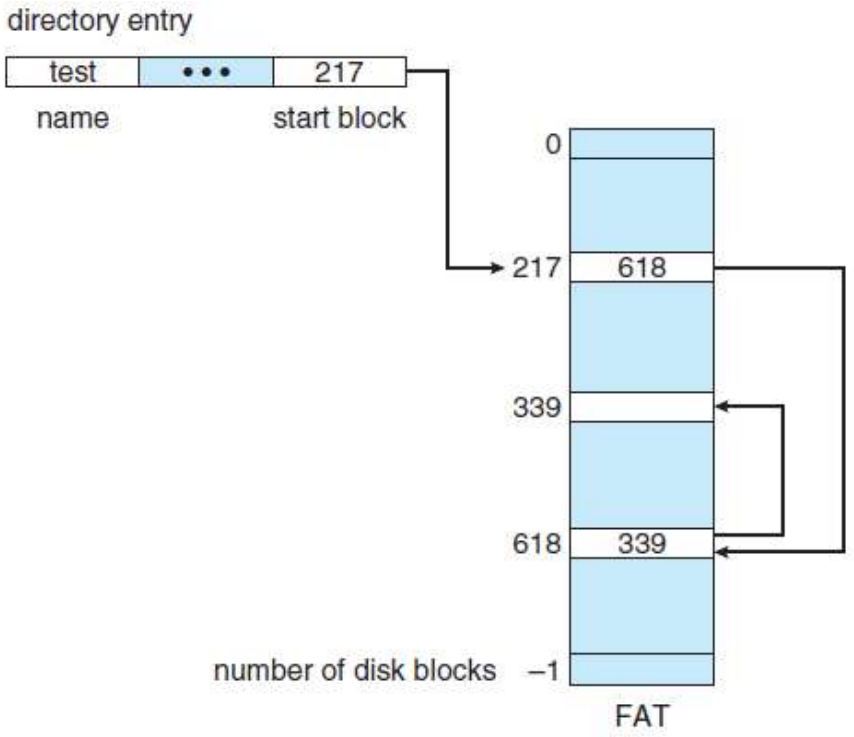


Fig 4.8.File-allocation table.

The *ith*entry in the index block points to the *ith*block of the file. The directorycontains the address of the index block (Figure 12.8). To find and read the *ith*block, we use the pointer in the *ith*index-block entry. This scheme is similar to the paging scheme described in Section 8.5. When the file is created, all pointers in the index block are set to null. When the *ith*block is first written, a block is obtained from the free-space manager, and its address is put in the *i*th index-block entry. Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space. Indexed allocation does suffer from wasted space, however. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

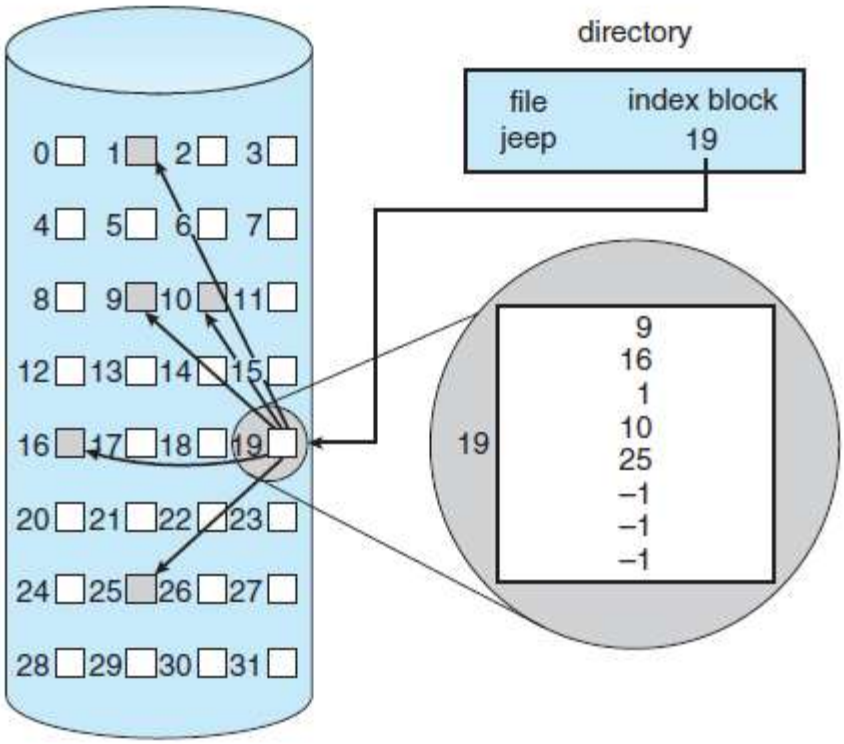


Fig 4.9.Indexed allocation of disk space.

Consider a common case in which we have a file of only one or two blocks.With linked allocation, we lose the space of only one pointer per block. With indexed allocation, an entire index block must be allocated, even if only one or two pointers will be non-null.

This point raises the question of how large the index block should be. Every file must have an index block, so we want the index block to be as small as possible. If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue.

Space Allocation

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation

- Linked Allocation
- Indexed Allocation

Contiguous Allocation

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Indexed Allocation

- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

To keep track of files, file systems normally have **directories** or **folders,**
which in many systems are themselves files. In this section we will discuss directories, their organization, their properties, and the operations that can be performed on them.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: IV (File and I/O Management)**

SEMESTER IV **BATCH (2018-2021)**

Device Management

A process may need several resources to execute—main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.

The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, disk drives), while others can be thought of as abstract or virtual devices (for example, files).A system with multiple users may require us to first request() a device, to ensure exclusive use of it. After we are finished with the device, we release() it.

These functions are similar to the open() and close() system calls for files. Other operating systems allow unmanaged access to devices. The hazard then is the potential for device contention and perhaps deadlock.

Once the device has been requested (and allocated to us), we can read(), write(), and (possibly) reposition() the device, just as we can with files. In fact, the similarity between I/O devices and files is so great that many operating systems, including UNIX, merge the two into a combined file–device structure. In this case, a set of system calls is used on both files and devices.

Sometimes, I/O devices are identified by special file names, directory placement, or file attributes. The user interface can also make files and devices appear to be similar, even though the underlying system calls are dissimilar. This is another example of the many design decisions that go into building an operating system and user interface.

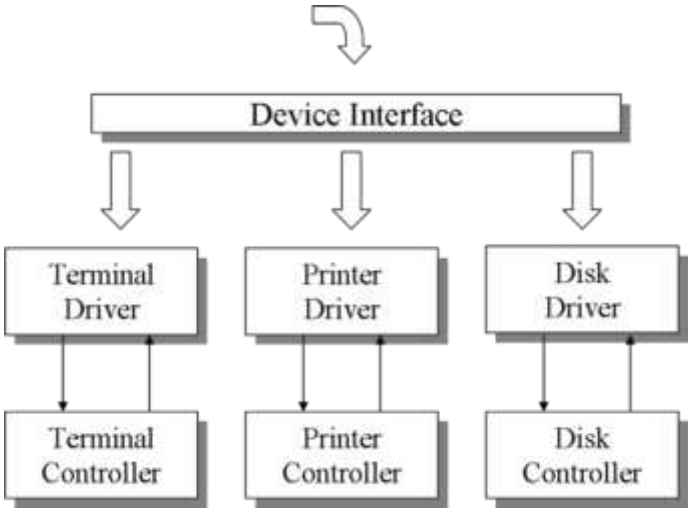
Hardware devices typically provide the ability to **input** data into the computer or **output** data from the computer. To simplify the ability to support a variety of hardware devices, standardized application programming interfaces (API) are used.

- Application programs use the System Call API to request one of a finite set of preset I/O requests from the Operating System.
- The Operating System uses algorithms for processing the request that are device independent.
- The Operating System uses another API to request data from the device driver.

- The device driver is third party software that knows how to interact with the specific device to perform the I/O.
- Sometimes we have a layering of device drivers where one device driver will call on another device driver to facilitate the I/O. An example of this is when devices are connected to a USB port. The driver for the device will make use of the USB device driver to facilitate passing data to and from the device.

Device Drivers

The Device Driver Interface



System Call Interface

- Functions available to application programs
- Abstract all devices (and files) to a few interfaces
- Make interfaces as similar as possible
  - Block vs character
  - Sequential vs direct access
- Device driver implements functions (one entry point per API function)

. Example - UNIX Driver

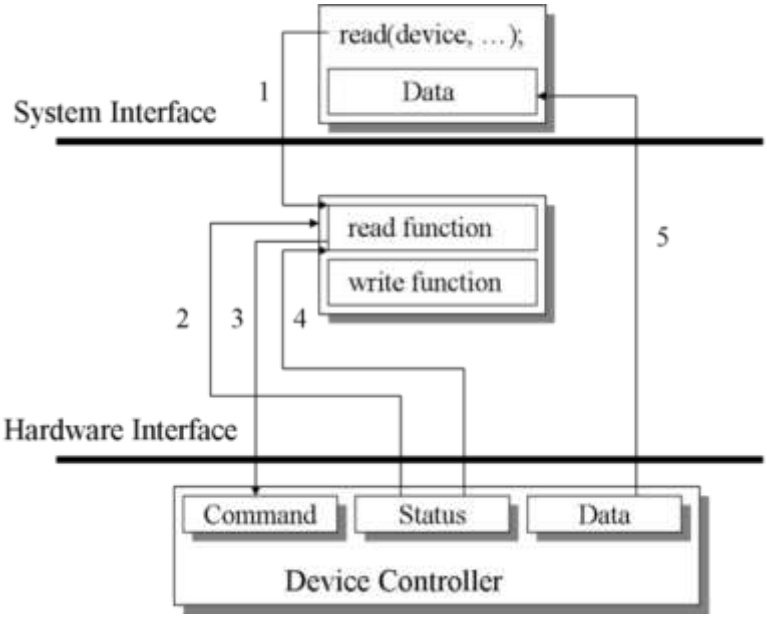
open	Prepare dev for operation
close	No longer using the device
ioctl	Character dev specific info
read	Character dev input op
write	Character dev output op
strategy	Block dev input/output ops
select	Character dev check for data
stop	Discontinue a stream output op

. Waiting for I/O

Many types of input / output (I/O) do not occur immediately. So the process must wait in a waiting queue and the device driver needs a strategy of how to effectively wait for I/O data.

Polling

Polling is not very efficient because the system must continually check the device for data.



Interrupt Driven

When an interrupt occurs it calls the interrupt handlers.

It is more efficient to start an I/O peripheral doing a task and let it interrupt the system when the I/O operation is finished.

The interrupt driven approach causes drivers to consist of a top and bottom half.

Non maskable interrupt

These kind of interrupts are reserved for events like unrecoverable memory errors

Maskable interrupt

Such interrupts can be switched off by the CPU before the execution of critical instructions that must not be interrupted.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

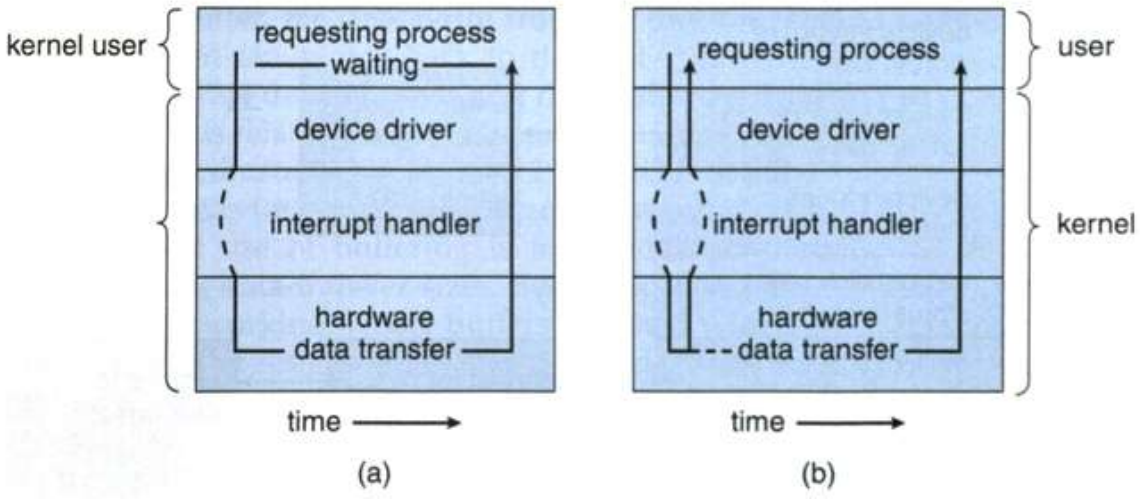
COURSE CODE: 18CTU401 **UNIT: IV (File and I/O Management)**

SEMESTER IV **BATCH (2018-2021)**

Blocking and Nonblocking I/O

Some control over how the wait for I/O to complete is accommodated is available to the programmer of user applications. Most I/O requests are considered **blocking** requests, meaning that control does not return to the application until the I/O is complete. The delayed from systems calls such `read()` and `write()` can be quite long. Using systems call that block is sometimes call **synchronous** programming. In most cases, the wait is not really a problem because the program can not do anything else until the I/O is finished.

One solution for these situations is to use multiple threads so that one part of the program is not waiting for unrelated I/O to complete. Another alternative is to use **asynchronous** programming techniques with **nonblocking** system calls. An asynchronous call returns immediately, without waiting for the I/O to complete.



Blocking I/O system calls (a) do not return until the I/O is complete. Nonblocking I/O sytems calls return immediately. The process is later notified when the I/O is complete.

A good example of nonblocking behavior is the `select()` system cal for network sockets. Using `select()`, an application can monitor several resources at the same time and can also poll

for network activity without blocking. The select() system call identifies if data is pending or not, then read() or write()may be used knowing that they will complete immediately.

POSSIBLE QUESTIONS

Part B (2 Marks)

1. List the File Operations?
2. What are the file allocation methods?
3. What are the directory structures available?
4. What is device management?
5. What is linked allocation?

Part C(6 Marks)

1. Explain File operations with example
2. Explain Device management in detail.
3. Describe about Directory structure with neat diagram
4. Write about File Allocation methods
5. Explain in detail about Tree structures directories
6. Write about Linked Allocation methods
7. Explain about Acyclic Graph directories in detail
8. Explain Device Driver Interface in detail.
9. Describe about Indexed Allocation methods

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021

(For the candidates admitted in 2018 onwards)

Unit I

CLASS : II B.Sc CT Semester:IV

SUBJECT : Operating Systems Batch:2018-2021

	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	File management function of the operating system includes	File creation and deletion	Disk scheduling	Process scheduling	Multiprogramming	File creation and deletion
2	What is a shell	It is a hardware component	It is a command interpreter	It is a part in compiler	It is a tool in CPU scheduling	It is a command interpreter
3	The queue has maximum length 0; thus, the link cannot have any	Zero capacity	Bounded capacity	Unbounded capacity	synchronous	Zero capacity
4	Let S and Q be two semaphores initialized to 1,where Po and p1 processes the following statements wait(S);wait(Q);--:signal(S) and signal(Q) and wait(Q);wait(S);_---;signal(Q);signal(S);respectively.The above sitaution depicts a	semaphore	deadlock	signal	interrupt	deadlock
5	The OS that groups similar jobs is called as	Network OS	Distributed OS	Distributed OS	Batch OS	Batch OS
6	_____is a program that manages the computer hardware and acts as an intermediary between the computer user and the computer hardware.	hardware acceleration	Operating System	compiler	logical transcation	Operating System

7	_____manages the execution of user programs to prevent errors and improper use of the computer	resource allocator	work station	main frame	control program	control program
8	_____ is a program associated with the operating system but are not part of the	System Program	User program	System calls	Functions	System Program
10	A more common definition is that the operating system is the one program running at all times on the computer usually called_____	bootstrap	firmware	kernel	read-only memory	kernel
11	The simplest CPU scheduling algorithm is the	FCS	SJS	FCFS	DFG	FCFS
12	computer system can be divided into _____ components	2	3	4	5	2
13	____were the first computers used to tackle many commercial & scientific application.	Mainframe computer system	Mainframe computer service	multiframe computer system	multiframe computer service	Mainframe computer system
14	_____system is the collection of computer that act,work and appear as one large computer size of a distributed system.	distributed	symmetric	asymmetric	multiple	distributed
15	_____contains the address of an instruction to be fetched from memory	Program counter (PC)	Instruction register (IR)	Control registers	Status registers	Instruction register (IR)
16	_____contains the instruction most recently fetched.	Program counter (PC)	Instruction register (IR)	Control registers	Status registers	Program counter (PC)
17	The kernel is a_____	memory manager	resource manager	file manager	directory manager	resource manager

18	Main function of shared memory is_____	to use primary memory efficently	to do intra process communication	to do inter process communication	to do other process communication	to do inter process communicatio n
19	Disk scheduling includes deciding_____	which should be accessed next	order in which disk access requests must be serviced	the physical location of the file	the logical location of the file	order in which disk access requests must be serviced
20	Memory protection is normally done by _____	the processor and the associated hardware	the operating system	the compiler	the user program	the processor and the associated hardware
21	With _____ more than one process can be running simultaneously each on a different processer.	Multiprogrammin g	Uniprocessing	Multiprocessing	Uniprogrammin g	Multiprocessin g
22	In Banker's Algorithm _____ conditions are allowed	mutual-exclusion	Time-sharing	race condition	cooperating processes	mutual-exclusion
23	_____is the process of actually determining that a deadlock exists	Deadlock detection	Deadlock prevention	fault tolerant	process synchronization	Deadlock detection
24	Once a system has become _____ the deadlock must be broken by removing one or more of the necessary conditions	deadlocked	race condition	mutual-exclusion	cooperating processes	deadlocked
25	_____ is also known as parallel systems or tightly coupled systems)	Multiprocessor systems	desktop systems	Time sharing systems	Multiprogramme d systems	Multiprocessor systems

26	_____operating systems are even more complex than multi programmed operating systems.	Time-sharing	desktop systems	Multiprogrammed systems	Multiprocessor systems	Time-sharing
27	_____ operating system keeps several jobs in memory simultaneously.	Time-sharing	desktop systems	Multiprogrammed systems	Multiprocessor systems	Multiprogrammed systems
28	_____can save more money than multiple single-processor systems	Multiprocessor systems	desktop systems	Time sharing systems	Multiprogrammed systems	Multiprocessor systems
29	This ability to continue providing service proportional to the level of surviving hardware is called	fault tolerant.	graceful degradation	Economy of scale	Increased throughput	graceful degradation
30	Systems designed for graceful degradation are also called	graceful degradation	Economy of scale	fault tolerant	Increased throughput	fault tolerant
31	The most common multiple-processor systems now use	symmetric multiprocessing	asymmetric multiprocessing	multithreading	multiprogramming	symmetric multiprocessing
32	Process states are _____	Submit, Ready	Submit, Run	Ready, Run	Submit, block	Ready, Run
33	The assignment of the CPU to the first process on the ready list is called	graceful degradation	Time-sharing	dispatching	Multiprocessor systems	dispatching
34	The manifestation of a process in an operating system is a	Process state transitions	process control block	child process	cooperating processes	process control block
35	A process may spawn a new process. If it does, the creating process is called the parent process and the created process is called the	child process	Process state transitions	Process state transitions	process control block	child process
36	The communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue known as	Buffering	synchronization	asynchronization	communication link	Buffering

37	_____ is a program that includes all programs not associated with the operation of the system	Application Programs	Kernel	Thread Program	Process	Application Programs
38	_____can assume only the value 0 or the value 1	Binary semaphores	Counting semaphores	semaphore operations	normal semaphores	Binary semaphores
39	Semophores are used to solve the problem of	race condition	process synchronizatio n	mutual exclusion	belady problem	mutual exclusion
40	For multiprogramming operating system	special support from processor is essential	special support from processor is not essential	cache memory is essential	cache memory is not essential	special support from processor is not essential
41	Which is single user operating system	MS-DOS	UNIX	XENIX	LINUX	MS-DOS
42	Which operating system reacts in the actual time	Batch system	Quick response system	Real time system	Time sharing system	Real time system
43	In real time OS, which is most suitable scheduling scheme	round robin	FCFS	pre-emptive scheduling	random scheduling	pre-emptive scheduling
44	Dispatcher function is to	put tasks in I/O wait	schedule tasks in processor	change task priorities	Multitasking	put tasks in I/O wait
45	Multiprogramming systems	Are easier to develop than single programming systems	Execute each job faster	Execute more jobs in the same time	Are used only on large main frame computers	Execute more jobs in the same time
46	Operating system is	A collection of hardware components	A collection of input output devices	A collection of software routines	last entered the queue	A collection of software routines

47	Semaphores function is to	synchronize critical resources to prevent deadlock	synchronize processes for better CPU utilization	used for memory management	may cause a high I/O rate	synchronize critical resources to prevent deadlock
48	Which operating system use write through catches	UNIX	XENIX	ULTRIX	DOS	DOS
49	Which process is known for initializing a microcomputer with its OS	cold booting	boot recording	booting	warm booting	booting
50	Four necessary conditions for deadlock are non pre-emption, circular wait, hold and wait and	mutual exclusion	race condition	buffer overflow	multiprocessing	mutual exclusion
51	Remote computing services involves the use of timesharing and	multiprocessing	interactive processing	batch processing	real time processing	batch processing
52	A series of statements explaining how the data is to be processed is called	instruction	compiler	program	interpretor	program
53	Banker's algorithm deals with	deadlock prevention	deadlock avoidance	deadlock recovery	mutual exclusion	deadlock avoidance
54	Which is non pre-emptive	Round robin	FIFO	MQS	MQSF	FIFO
55	A hardware device which is capable of executing a sequence of instructions, is known as	CPU	ALU	CU	Processor	Processor
56	Distributed systems should	high security	have better resource sharing	better system utilization	low system overhead	have better resource sharing
57	Which of the following is always there in a computer	Batch system	Operating system	Time sharing system	Controlling system	Operating system

58	Which of following is not an advantage of multiprogramming	increased throughput	shorter response time	ability to assign priorities of jobs	decreased system overload	decreased system overload
59	In which of the following usually a front end processor is used	Virtual storage	Timesharing	Multiprogamming	Multithreading	Timesharing
60	Remote computing services involves the use of	multiprocessing	multiprogramming	batch processing	real time processing	batch processing
61	Banker's algorithm for resource allocation deals with	deadlock prevention	deadlock aviodance	deadlock recovery	circular wait	deadlock aviodance
62	In Unix, Which system call creates the new process?	Fork	Create	New	Old	Fork
63	The queue has finite length n; thus, at most n messages can reside in it is known as	Zero capacity	Bounded capacity	Unbounded capacity	multiprocessing	Bounded capacity
64	The queue has potentially infinite length; thus, any number of messages can wait in it is known as	Zero capacity	Bounded capacity	Unbounded capacity	multiprocessing	Unbounded capacity
65	Each process accessing the shared data excludes all others from doing so simultaneously and this is called	mutual exclusion	multiprocessing	real time processing	multiprogramming	mutual exclusion
	_____ OS runs on Apple produced hardware	Unix	Linux	Linux Mint	Apple	Apple
	PC _____ supports complex games	OS	PMT	PCB	Software	OS
66	The primary job of an OS is to _____	command resource	manage resource	provide utilities	Be user friendly	manage resource

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: V (Protection and Security)**

SEMESTER IV **BATCH (2018-2021)**

SYLLABUS

Protection and Security- Policy mechanism-Authentication-Internal Access Authorization

Policy Mechanism

The policies what is to be done while the mechanism specifies how it is to be done. For instance, the timer construct for ensuring CPU protection is mechanism. On the other hand, the decision of how long the timer is set for a particular user is a policy decision.

The separation of mechanism and policy is important to provide flexibility to a system. If the interface between mechanism and policy is well defined, the change of policy may affect only a few parameters. On the other hand, if interface between these two is vague or not well defined, it might involve much deeper change to the system.

Once the policy has been decided it gives the programmer the choice of using his/her own implementation. Also, the underlying implementation may be changed for a more efficient one without much trouble if the mechanism and policy are well defined. Specifically, separating these two provides flexibility in a variety of ways. First, the same mechanism can be used to implement a variety of policies, so changing the policy might not require the development of a new mechanism, but just a change in parameters for that mechanism, but just a change in parameters for that mechanism from a library of mechanisms. Second, the mechanism can be changed for example, to increase its efficiency or to move to a new platform, without changing the overall policy.

Policy vs mechanism OS examples

- Granting a resource to a process using first come first serve algorithm (policy). This policy can be implemented using a queue (mechanism).
- Thread scheduling or answering the question "which thread should be given the chance to run next?" is a policy. For example, is it priority based ? or just round robin ?. Implementing context switching is the corresponding mechanism.
- In virtual memory, keeping track of free and occupied pages in memory is a mechanism. Deciding what to do when a page fault occurs is a policy. You may check the following articles cpu scheduling, paging vs segmentation and page tables

Separation of mechanism and policy

Separation of policy and mechanism is a design principle to achieve flexibility. In other words, adopting a certain mechanism should not restrict existing policies. The idea behind this concept

is to have the least amount of implementation changes if we decide to change the way a particular feature is used. We can also look at it from the other side. For example, if a certain implementation needs to be changed (ex. improve efficiency). This must not greatly influence the way it is used. In the login example mentioned earlier (logging to a website) switching from a user name password pair to Facebook account should not prevent a user from logging in to the website.

Summary

- Policy is the what and mechanism is the how.
- The separation between the two gives us the flexibility to add and modify existing policies and reuse existing mechanisms for implementing new policies.

Mechanism versus Policy

- Another principle that helps architectural coherence, along with keeping things small and well structured, is that of separating mechanism from policy. By putting the mechanism in the operating system and leaving the policy to user processes, the system itself can be left unmodified, even if there is a need to change policy. Even if the policy module has to be kept in the kernel, it should be isolated from the mechanism, if possible, so that changes in the policy module do not affect the mechanism module.

- To make the split between policy and mechanism clearer, let us consider two real-world examples. As a first example, consider a large company that has a payroll department, which is in charge of paying the employees' salaries. It has computers, software, blank checks, agreements with banks, and more mechanism for actually paying out the salaries. However, the policy—determining who gets paid how much—is completely separate and is decided by management. The payroll department just does what it is told to do.

- As the second example, consider a restaurant. It has the mechanism for serving diners, including tables, plates, waiters, a kitchen full of equipment, agreements with credit card companies, and so on. The policy is set by the chef, namely, what is on the menu. If the chef decides that tofu is out and big steaks are in, this new policy can be handled by the existing mechanism.

- Now let us consider some operating system examples. First, consider thread scheduling. The kernel could have a priority scheduler, with *k* priority levels. The mechanism is an array, indexed by priority level, as shown in Fig. 10-11 or Fig. 11-19. Each entry is the head of a list of ready threads at that priority level. The scheduler just searches the array from highest priority to lowest priority, selecting the first threads it hits. The policy is setting the priorities. The system may have different classes of users, each with a different priority, for example. It might also allow user processes to set the relative priority of its threads. Priorities might be increased after completing I/O or decreased after using up a quantum. There are numerous other policies that could be followed, but the idea here is the separation between setting policy and carrying it out.

- A second example is paging. The mechanism involves MMU management, keeping lists of occupied pages and free pages, and code for shuttling pages to and from disk. The policy is deciding what to do when a page fault occurs. It could be local or global, LRU-based or FIFO-based, or something else, but this algorithm can (and should) be completely separate from the mechanics of actually managing the pages.

- A third example is allowing modules to be loaded into the kernel. The mechanism concerns how they are inserted, how they are linked, what calls they can make, and what calls can be made on them. The policy is determining who is allowed to load a module into the kernel and which modules. Maybe only the superuser can load modules, but maybe any user can load a module that has been digitally signed by the appropriate authority.

Policy and Mechanism

Critical to our study of security is the distinction between policy and mechanism.

- **Definition 1–1**. A *security policy* is a statement of what is, and what is not, allowed.

- **Definition 1–2**. A *security mechanism* is a method, tool, or procedure for enforcing a security policy.

Mechanisms can be nontechnical, such as requiring proof of identity before changing a password; in fact, policies often require some procedural mechanisms that technology cannot enforce.

As an example, suppose a university's computer science laboratory has a policy that prohibits any student from copying another student's homework files. The computer system provides mechanisms for preventing others from reading a user's files. Anna fails to use these mechanisms to protect her homework files, and Bill copies them. A breach of security has occurred, because Bill has violated the security policy. Anna's failure to protect her files does not authorize Bill to copy them.

In this example, Anna could easily have protected her files. In other environments, such protection may not be easy. For example, the Internet provides only the most rudimentary security mechanisms, which are not adequate to protect information sent over that network. Nevertheless, acts such as the recording of passwords and other sensitive information violate an implicit security policy of most sites (specifically, that passwords are a user's confidential property and cannot be recorded by anyone).

Policies may be presented mathematically, as a list of allowed (secure) and disallowed (nonsecure) states. For our purposes, we will assume that any given policy provides an axiomatic description of secure states and nonsecure states. In practice, policies are rarely so precise; they normally describe in English what users and staff are allowed to do. The ambiguity inherent in such a description leads to states that are not classified as "allowed" or "disallowed." For example, consider the homework policy discussed above. If someone looks through another user's directory without copying homework files, is that a violation of security? The answer depends on site custom, rules, regulations, and laws, all of which are outside our focus and may change over time.

When two different sites communicate or cooperate, the entity they compose has a security policy based on the security policies of the two entities. If those policies are inconsistent, either or both sites must decide what the security policy for the combined site should be. The inconsistency often manifests itself as a security breach. For example, if proprietary documents

were given to a university, the policy of confidentiality in the corporation would conflict with the more open policies of most universities. The university and the company must develop a mutual security policy that meets both their needs in order to produce a consistent policy. When the two sites communicate through an independent third party, such as an Internet service provider, the complexity of the situation grows rapidly.

Authentication

Every *secured* computer system must require all users to be authenticated at login time. After all, if the operating system cannot be sure who the user is, it cannot know which files and other resources he can access. While authentication may sound like a trivial topic, it is a bit more complicated than you might expect.

Early minicomputers (e.g., PDP-1 and PDP-8) did not have a login procedure,

but with the spread of UNDC on the PDP-11 minicomputer, logging in was again needed. Early personal computers (e.g., Apple II and the original IBM PC) did not have a login procedure, but more sophisticated personal computer operating systems, such as Linux and Windows Vista, do (although foolish users can disable it). Machines on corporate LANs almost always have a login procedure configured so that users cannot bypass it. Finally, many people nowadays (indirectly) log into remote computers to do Internet banking, e-shopping, download music, and other commercial activities. All of these things require authenticated login, so user authentication is once again an important topic.

Having determined that authentication is often important, the next step is to find a good way to achieve it. Most methods of authenticating users when they attempt to log in are based on one of three general principles, namely identifying

1. Something the user knows.
2. Something the user has.
3. Something the user is.

Sometimes two of these are required for additional security. These principles lead to different authentication schemes with different complexities and security properties. In the following sections we will examine each of these in turn. People who want to cause trouble on a particular system have to first log in to that system, which means getting past whichever authentication procedure is used. In the popular press, these people are called **hackers.** In deference to true

hackers, we will use the term in the original sense and will call people who try to break into computer systems where they do not belong **crackers.**

Authentication Using Passwords

The most widely used form of authentication is to require the user to type a login name and a password. Password protection is easy to understand and easy to implement.

The simplest implementation just keeps a central list of (login-name, password) pairs. The login name typed in is looked up in the list and the typed password is compared to the stored password. If they match, the login is allowed; if they do not match, the login is rejected.

It goes almost without saying that while a password is being typed in, the computer should not display the typed characters, to keep them from prying eyes near the monitor. With Windows, as each character is typed, an asterisk is displayed. With UNIX, nothing at all is displayed while the password is being typed. These schemes have different properties. The Windows scheme may make it easy for absent-minded users to see how many characters they have typed so far, but it also discloses the password length to "eavesdroppers" (for some reason, English has a word for auditory snoopers but not for visual snoopers, other than perhaps Peeping Tom, which does not seem right in this context). From a security perspective, silence is golden.

How Crackers Break In

Most crackers break in by connecting to the target computer (e.g., over the Internet) and trying many (login name, password) combinations until they find one that works. Many people use their name in one form or another as their login name. Of course, guessing the login name is not enough. The password has to be guessed, too. How hard is that? Easier than you might think. The classic work on password security was done by Morris and Thompson (1979) on UNIX systems.

They compiled a list of likely passwords: first and last names, street names, city names, words from a moderate-sized dictionary (also words spelled backward), license plate numbers, and short strings of random characters. They then compared their list to the system password file to see if there were any matches. Over 86% of all passwords turned up in their list.

UNIX Password Security

Some (older) operating systems keep the password file on the disk in unencrypted form, but protected by the usual system protection mechanisms. Having all the passwords in a disk file in unencrypted form is just looking for trouble because all too often many people have access to it.

These may include system administrators, machine operators, maintenance personnel, programmers, management, and maybe even some secretaries.

A better solution, used in UNIX, works like this. The login program asks the user to type his name and password. The password is immediately "encrypted" by using it as a key to encrypt a fixed block of data. Effectively, a one-way function is being run, with the password as input and a function of the password as output. This process is not really encryption, but it is easier to speak of it as encryption. The login program then reads the password file, which is just a series of ASCII lines, one per user, until it finds the line containing the user's login name.

If the (encrypted) password contained in this line matches the encrypted password just computed, the login is permitted, otherwise it is refused. The advantage of this scheme is that no one, not even the superuser, can look up any users' passwords because they are not stored in unencrypted form anywhere in the system.

One-Time Passwords

Most superusers exhort their mortal users to change their passwords once a month. It falls on deaf ears. Even more extreme is changing the password with every login, leading to one-time passwords. When one-time passwords are used, the user gets a book containing a list of passwords. Each login uses the next password in the list. If an intruder ever discovers a password, it will not do him any good, since next time a different password must be used. It is suggested that the user try to avoid losing the password book.

Authentication Using a Physical Object

The second method for authenticating users is to check for some physical object they have rather than something they know. Metal door keys have been used for centuries for this purpose. Nowadays, the physical object used is often a plastic card that is inserted into a reader associated with the computer. Normally, the user must not only insert the card, but must also type in a password, to prevent someone from using a lost or stolen card. Viewed this way, using a bank's ATM (Automated Teller Machine) starts out with the user logging in to the bank's computer via a remote terminal (the ATM machine) using a plastic card and a password (currently a 4-digit

PIN code in most countries, but this is just to avoid the expense of putting a full keyboard on the ATM machine).

Information-bearing plastic cards come in two varieties: magnetic stripe cards and chip cards. Magnetic stripe cards hold about 140 bytes of information written on a piece of magnetic tape glued to the back of the card. This information can be read out by the terminal and sent to the central computer. Often the information contains the user's password (e.g., PIN code) so the terminal can do an identity check even if the link to the main computer is down. Typically the password is encrypted by a key known only to the bank.

Authentication Using Biometrics

The third authentication method measures physical characteristics of the user that are hard to forge. These are called biometrics (Pankanti et al., 2000). For example, a fingerprint or voiceprint reader hooked up to the computer could verify the user's identity.

A typical biometrics system has two parts: enrollment and identification. During enrollment, the user's characteristics are measured and the results digitized. Then significant features are extracted and stored in a record associated with the user. The record can be kept in a central database (e.g., for logging in to a remote computer), or stored on a smart card that the user carries around and inserts into a remote reader (e.g., at an ATM machine).

The other part is identification. The user shows up and provides a login name. Then the system makes the measurement again. If the new values match the ones sampled at enrollment time, the login is accepted; otherwise it is rejected. The login name is needed because the measurements are never exact, so it is difficult to index them and then search the index. Also, two people might have the same characteristics, so requiring the measured characteristics to match those of a specific user is stronger than just requiring it to match those of any user.

Access Control

Operating System Security

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by unauthorized user then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc. We're going to discuss following topics in this article.

- Authentication
- One Time passwords
- Program Threats
- System Threats
- Computer Security Classifications

Authentication

Authentication refers to identifying the each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways:

- **Username / Password** - User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** - User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** - User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One time passwords provides additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used then it cannot be used again. One time password are implemented in various ways.

Random numbers - Users are provided cards having numbers printed along with **corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.**

- **Secret key** - User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.
- **Network password** - Some commercial applications send one time password to user on registered mobile/ email which is required to be entered prior to login.

Program Threats

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks then it is known as Program Threats. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well known program threats.

- **Trojan Horse** - Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.
- **Trap Door** - If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
- **Logic Bomb** - Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.
- **Virus** - Virus as name suggests can replicate themselves on computer system .They are highly dangerous and can modify/delete user files, crash systems. A virus is generally a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user.

System Threats

System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well known system threats.

- **Worm** -Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network.
- **Port Scanning** - Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.
- **Denial of Service** - Denial of service attacks normally prevents user to make legitimate use of the system. For example user may not be able to use internet if denial of service attacks browser's content settings.

Identification, Authentication, Authorization

For a user to be able to access a resource, he must first prove that he is who he claims to be, has the necessary credentials, and has been given the necessary rights or privileges to perform the actions he is requesting.

Identification describes a method of ensuring that a subject (user, program, or process) is the entity it claims to be. Identification can be provided with the use of a username or account number.

To be properly **authenticated**, the subject is usually required to provide a second piece to the credential set (a password, a cryptographic key, personal identification number (PIN), ….).

Identification, Authentication, Authorization (cont'd)

If identification and authentication credentials match the stored information, the subject is **authenticated**.

Once the subject is authenticated, the system it is trying to access needs to determine if this subject has been given the necessary rights and privileges to carry out the requested actions.

If the system determines that the subject may access the resource, it **authorizes** the subject. These mechanisms are enforced through AAA (Authentication, Authorization and Auditing) tools.

AAA tools

Access controls tools are used for identification, authentication, authorization, and auditability. They are software components that enforce access control measures for systems, programs, processes, and information.

They can be embedded within operating systems, applications, add-on security packages, or database and telecommunication management systems.

They can be offered as outsourced services by trusted third parties.

It can be challenging to synchronize all access controls and ensure that all vulnerabilities are covered without producing overlaps of functionality.

Identification and Authentication

Once a person has been identified, through the user ID, he must be authenticated; He must prove he is who he says he is.

There are three general factors that can be used for authentication:

- something a person knows (a password, PIN, …);
- something a person has (a key, an access card, a badge);
- something a person is (physical attributes).

Identification and Authentication (cont'd)

Authenticating a person by so*mething that he knows* is usually the least expensive to implement, but it is less secure, too. Another person may easily acquire this knowledge and gain unauthorized access to a system.

Something a person has is a very common mechanism but the token's life-cycle needs to be managed, they can be lost or stolen, which could result in unauthorized access.

Authenticating a person's identity based on a unique *physical attribute* is referred to as biometrics.

Strong authentication contains two out of these three methods: something a person knows, has, or is (*two-factors authentication*).

Identification Component Requirements

When issuing identification values to users, the following should be in place:

- Each value should be unique, for user accountability.
- A standard naming scheme should be followed.
- The value should be non-descriptive of the user's position or tasks.
- The value should not be shared between users.

Identity management

Identity management is a broad term that encompasses the use of different products to identify, authenticate, and authorize users through automated means.

The continual increase in complexity and diversity of networked environments only increases the complexity of keeping track of who can access what and when.

Users usually access several different types of systems throughout their daily tasks, which makes controlling access and providing the necessary level of protection on different data types difficult and full of obstacles.

This complexity usually results in unforeseen and unidentified holes in asset protection, overlapping and contradictory controls, and policy and regulation noncompliance.

It is the goal of identity management technologies to simplify the administration of these tasks and bring sanity to chaos.

Identity management (cont'd)

The following are many of the common problems that enterprises deal with today in controlling access to assets:

- Various types of users need different levels of access: internal users, contractors, outsiders, partners, etc.
- Resources have different classification levels: confidential, internal use only, private, public, etc.

- Diverse identity data must be kept on different types of users: credentials, personal data, contact information, work-related data, digital certificates, cognitive passwords, etc.
- The corporate environment is continually changing: business environment needs, resource access needs, employee roles, current employees, etc.

Identity management (cont'd)

The traditional identity management process has been manual, using directory services with permissions and profiles.

This approach has proven incapable of keeping up with complex demands and thus has been replaced with the use of newly arrived automated applications that are rich in functionality, including enterprise-wide products and single sign-on solutions.

Identity management (cont'd)

The following are some of the services that these types of products supply:

- User provisioning
- Password synchronization and resetting
- Self service for users on specific types of activities
- Delegation of administrative tasks
- Centralized auditing and reporting
- Integrated workflow and increase in business productivity
- Decrease in network access points
- Regulatory compliance

Authentication mechanisms

Biometrics:

- Fingerprint
- Palm Scan
- Hand Geometry
- Retina Scan
- Iris Scan
- Signature Dynamics
- Keyboard Dynamics
- Voice Print
- Facial Scan

- Hand Topography

Authentication mechanisms (cont'd)

Passwords….. weakness:

If an attacker is after a password, he can try different techniques:

- **Electronic monitoring** Listening to network traffic to capture information, especially when a user is sending her password to an authentication server. The password can be copied and reused by the attacker at another time, which is called a replay attack.
- **Access the password file** Usually done on the authentication server. The password file contains many users' passwords and, if compromised, can be the source of a lot of damage. This file should be protected with access control mechanisms and encryption.
- **Brute force attacks** Performed with tools that cycle through many possible character, number, and symbol combinations to uncover a password.
- **Dictionary attacks** Files of thousands of words are used to compare to the user's password until a match is found.
- **Social engineering** An attacker falsely convinces an individual that she has the necessary authorization to access specific resources.

Authentication mechanisms (cont'd)

- Password Checkers;
- Password Hashing and Encryption;
- Password Aging;
- Limit Logon Attempts;
- Cognitive Passwords;
- One-Time Passwords:
- Synchronous token device,
- Asynchronous token device;
- Cryptographic keys;
- Smart Cards.

Authorization mechanisms (cont'd)

After successful authentication, the system must establish whether the user is authorized to access the particular resource and what actions he is permitted to perform on that resource.

Authorization is a core component of every operating system, but applications, security add-on packages, and resources themselves can also provide this functionality.

The decision of whether or not to allow users to access some resource was based on access criteria.

Access criteria is the crux of authentication.

Authorization: Access Criteria

This subject can get very granular in its level of detail when it comes to dictating what a subject can or cannot do to an object or resource.

This is a good thing for network administrators and security professionals, because they want to have as much control as possible over the resources they have been put in charge of protecting, and a fine level of detail enables them to give individuals just the precise level of access that they need.

It would be frustrating if access control permissions were based only on full control or no access. These choices are very limiting, and an administrator would end up giving everyone full control, which would provide no protection.

There are different ways of limiting access to resources and, if they are understood and used properly, they can give just the right level of access desired.

Authorization: Access Criteria (cont'd)

Granting access rights to subjects should be based on the level of trust a company has in a subject and the subject's need to know. Just because a company completely trusts Alice with its files and resources does not mean she fulfills the need-to-know criteria to access the company's tax returns and profit margins.

These issues need to be identified and integrated into the access criteria.

The different access criteria can be broken up into different types:

- roles,
- groups,
- location,
- time,
- transaction types.

Access Criteria

- Using ***roles*** is an efficient way to assign rights to a type of user who performs a certain task. The role is based on a job assignment or function.
- Using ***groups*** is another effective way of assigning access control rights. If several users require the same type of access to information and resources, putting them into a group and then assigning rights and permissions to that group is easier to manage than assigning rights and permissions to each and every individual separately.

Access Criteria (cont'd)

- **Physical or logical location** can also be used to restrict access to resources. Some files may be available only to users who can log on interactively to a computer. Logical location restrictions are usually done through network address restrictions.
- **Time of day**, or temporal isolation, is another access control mechanism that can be used.
- **Transaction-type** restrictions can be used to control what data is accessed during certain types of functions and what commands can be carried out on the data. An online banking program may allow a customer to view his account balance, but may not allow the customer to transfer money until he has a certain security level or access right. (A database administrator may be able to build a database for the human resources department, but may not be able to read certain confidential files within that database).

Authorization Creep

As employees work at a company over time and move from one department to another, they often are assigned more and more access rights and permissions.

This is commonly referred to as authorization creep. It can be a large risk for a company, because too many users have too much privileged access to company assets.

Users' access needs and rights should be periodically reviewed to ensure that the principle of least privilege is being properly enforced.

Notes on Authorization

It is important to understand that it is management's job to determine the security requirements of individuals and how access is authorized.

The security administrator configures the security mechanisms to fulfill these requirements, but it is not her job to determine security requirements of users.

Access Control Models

An **access control model** is a framework that dictates how subjects access objects. It uses access control technologies and security mechanisms to enforce the rules and objectives of the model.

There are three main types of access control models:

- Discretionary (DAC),
- Mandatory (MAC),
- Nondiscretionary (also called role-based RBAC).

Each model type uses different methods to control how subjects access objects.

For every access attempt, before a subject can communicate with an object, the security monitor reviews the rules of the access control model to determine whether the request is allowed.

Discretionary Access Control

If a user creates a file, he is the owner of that file. An identifier for this user is placed in the file header.

A system that uses ***discretionary access control (DAC)*** enables the owner of the resource to specify which subjects can access specific resources.

This model is called discretionary because the control of access is based on the discretion of the owner.

The most common implementation of DAC is through ACLs, which are dictated and set by the owners and enforced by the operating system.

Discretionary Access Control (cont'd)

Most of the operating systems are based on DAC models, such as all Windows, Linux, and Macintosh systems and most flavors of Unix.

When you look at the properties of a file or directory and you see the choices that allow you to control which users can have access to this resource and to what degree, you are witnessing an instance of ACLs enforcing a DAC model.

DACs can be applied to both the directory tree structure and the files it contains.

The PC world has access permissions of No Access, Read (r), Write (w), Execute (x), Delete (d), Change (c), and Full Control.

CLASS: II B.Sc. (CT) **COURSE NAME: OPERATING SYSTEMS**

COURSE CODE: 18CTU401 **UNIT: V (Protection and Security)**

SEMESTER IV **BATCH (2018-2021)**

Notes on DAC

Identity-Based Access Control

- DAC systems grant or deny access based on the identity of the subject. The identity can be a user identity or group membership. So, for example, a data owner can choose to allow Bob (user identity) and the Accounting group (group membership identity) to access his file.

Mandatory Access Control

In a ***mandatory access control (MAC)*** model, users and data owners cannot determine who can access files.

This model is much more structured and strict and is based on a security label system. Users are given a security clearance (secret, top secret, confidential, and so on), and data is classified in the same way. The clearance and classification data is stored in the security labels, which are bound to the specific subjects and objects.

When the system makes a decision about fulfilling a request to access an object, it is based on the clearance of the subject, the classification of the object, and the security policy of the system.

The rules for how subjects access objects are made by the security officer, configured by the administrator, enforced by the operating system, and supported by security technologies.

DAC and MAC limitations

Each organization has unique security requirements, many of which are difficult to meet using traditional DAC and MAC controls.

DAC is an access control mechanisms that permits system users to allow or disallow other users access to objects under their control without the intercession of a system administrator.

In many organizations, the end users do not “own” the information for which they are allowed access; the actual “owner” is the corporation -> control has to be based on employee functions rather than data ownership.

Role-based Access Control

A user has access to an object based on the assigned role.

Roles are defined based on job functions.

Permissions are defined based on job authority and responsibilities within a job function.

Operations on an object are invocated based on the permissions.

The object is concerned with the user's role and not the user.

POSSIBLE QUESTIONS

Part B (2 Marks)

1. What is Authentication?.
2. What is Access Control Lists?
3. What is internal access authorization
4. What is security in OS?
5. What are authentication mechanisms

Part C (6 Marks)

1. Discuss about Policy mechanism with example.
2. Explain about Internal access Authorization
3. Discuss in detail about the different types of Access criteria
4. Explain in detail the Authentication mechanisms used in operating systems
5. Explain about Authentication in detail
6. Discuss about the Access control Models in detail.
7. Explain about Identity management in detail
8. Discuss about the method to authenticate a system using a physical object.
9. Explain in detail the method used to authenticate using passwords.
10. Illustrate in detail about Access control

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021

(For the candidates admitted in 2018 onwards)

Unit I

CLASS : II B.Sc CT Semester:IV

SUBJECT : Operating Systems Batch:2018-2021

	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	File management function of the operating system includes	File creation and deletion	Disk scheduling	Process scheduling	Multiprogramming	File creation and deletion
2	What is a shell	It is a hardware component	It is a command interpreter	It is a part in compiler	It is a tool in CPU scheduling	It is a command interpreter
3	The queue has maximum length 0; thus, the link cannot have any	Zero capacity	Bounded capacity	Unbounded capacity	synchronous	Zero capacity
4	Let S and Q be two semaphores initialized to 1,where Po and p1 processes the following statements wait(S);wait(Q);--:signal(S) and signal(Q) and wait(Q);wait(S);_---;signal(Q);signal(S);respectively.The above sitaution depicts a	semaphore	deadlock	signal	interrupt	deadlock
5	The OS that groups similar jobs is called as	Network OS	Distributed OS	Distributed OS	Batch OS	Batch OS
6	_____is a program that manages the computer hardware and acts as an intermediary between the computer user and the computer hardware.	hardware acceleration	Operating System	compiler	logical transcation	Operating System

7	_____manages the execution of user programs to prevent errors and improper use of the computer	resource allocator	work station	main frame	control program	control program
8	_____ is a program associated with the operating system but are not part of the	System Program	User program	System calls	Functions	System Program
10	A more common definition is that the operating system is the one program running at all times on the computer usually called_____	bootstrap	firmware	kernel	read-only memory	kernel
11	The simplest CPU scheduling algorithm is the	FCS	SJS	FCFS	DFG	FCFS
12	computer system can be divided into _____ components	2	3	4	5	2
13	____were the first computers used to tackle many commercial & scientific application.	Mainframe computer system	Mainframe computer service	multiframe computer system	multiframe computer service	Mainframe computer system
14	_____system is the collection of computer that act,work and appear as one large computer size of a distributed system.	distributed	symmetric	asymmetric	multiple	distributed
15	_____contains the address of an instruction to be fetched from memory	Program counter (PC)	Instruction register (IR)	Control registers	Status registers	Instruction register (IR)
16	_____contains the instruction most recently fetched.	Program counter (PC)	Instruction register (IR)	Control registers	Status registers	Program counter (PC)
17	The kernel is a_____	memory manager	resource manager	file manager	directory manager	resource manager

18	Main function of shared memory is_____	to use primary memory efficently	to do intra process communication	to do inter process communication	to do other process communication	to do inter process communicatio n
19	Disk scheduling includes deciding_____	which should be accessed next	order in which disk access requests must be serviced	the physical location of the file	the logical location of the file	order in which disk access requests must be serviced
20	Memory protection is normally done by _____	the processor and the associated hardware	the operating system	the compiler	the user program	the processor and the associated hardware
21	With _____ more than one process can be running simultaneously each on a different processer.	Multiprogrammin g	Uniprocessing	Multiprocessing	Uniprogrammin g	Multiprocessin g
22	In Banker's Algorithm _____ conditions are allowed	mutual-exclusion	Time-sharing	race condition	cooperating processes	mutual-exclusion
23	_____is the process of actually determining that a deadlock exists	Deadlock detection	Deadlock prevention	fault tolerant	process synchronization	Deadlock detection
24	Once a system has become _____ the deadlock must be broken by removing one or more of the necessary conditions	deadlocked	race condition	mutual-exclusion	cooperating processes	deadlocked
25	_____ is also known as parallel systems or tightly coupled systems)	Multiprocessor systems	desktop systems	Time sharing systems	Multiprogramme d systems	Multiprocessor systems

26	_____operating systems are even more complex than multi programmed operating systems.	Time-sharing	desktop systems	Multiprogrammed systems	Multiprocessor systems	Time-sharing
27	_____ operating system keeps several jobs in memory simultaneously.	Time-sharing	desktop systems	Multiprogrammed systems	Multiprocessor systems	Multiprogrammed systems
28	_____can save more money than multiple single-processor systems	Multiprocessor systems	desktop systems	Time sharing systems	Multiprogrammed systems	Multiprocessor systems
29	This ability to continue providing service proportional to the level of surviving hardware is called	fault tolerant.	graceful degradation	Economy of scale	Increased throughput	graceful degradation
30	Systems designed for graceful degradation are also called	graceful degradation	Economy of scale	fault tolerant	Increased throughput	fault tolerant
31	The most common multiple-processor systems now use	symmetric multiprocessing	asymmetric multiprocessing	multithreading	multiprogramming	symmetric multiprocessing
32	Process states are _____	Submit, Ready	Submit, Run	Ready, Run	Submit, block	Ready, Run
33	The assignment of the CPU to the first process on the ready list is called	graceful degradation	Time-sharing	dispatching	Multiprocessor systems	dispatching
34	The manifestation of a process in an operating system is a	Process state transitions	process control block	child process	cooperating processes	process control block
35	A process may spawn a new process. If it does, the creating process is called the parent process and the created process is called the	child process	Process state transitions	Process state transitions	process control block	child process
36	The communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue known as	Buffering	synchronization	asynchronization	communication link	Buffering

37	_____ is a program that includes all programs not associated with the operation of the system	Application Programs	Kernel	Thread Program	Process	Application Programs
38	_____can assume only the value 0 or the value 1	Binary semaphores	Counting semaphores	semaphore operations	normal semaphores	Binary semaphores
39	Semophores are used to solve the problem of	race condition	process synchronizatio n	mutual exclusion	belady problem	mutual exclusion
40	For multiprogramming operating system	special support from processor is essential	special support from processor is not essential	cache memory is essential	cache memory is not essential	special support from processor is not essential
41	Which is single user operating system	MS-DOS	UNIX	XENIX	LINUX	MS-DOS
42	Which operating system reacts in the actual time	Batch system	Quick response system	Real time system	Time sharing system	Real time system
43	In real time OS, which is most suitable scheduling scheme	round robin	FCFS	pre-emptive scheduling	random scheduling	pre-emptive scheduling
44	Dispatcher function is to	put tasks in I/O wait	schedule tasks in processor	change task priorities	Multitasking	put tasks in I/O wait
45	Multiprogramming systems	Are easier to develop than single programming systems	Execute each job faster	Execute more jobs in the same time	Are used only on large main frame computers	Execute more jobs in the same time
46	Operating system is	A collection of hardware components	A collection of input output devices	A collection of software routines	last entered the queue	A collection of software routines

47	Semaphores function is to	synchronize critical resources to prevent deadlock	synchronize processes for better CPU utilization	used for memory management	may cause a high I/O rate	synchronize critical resources to prevent deadlock
48	Which operating system use write through catches	UNIX	XENIX	ULTRIX	DOS	DOS
49	Which process is known for initializing a microcomputer with its OS	cold booting	boot recording	booting	warm booting	booting
50	Four necessary conditions for deadlock are non pre-emption, circular wait, hold and wait and	mutual exclusion	race condition	buffer overflow	multiprocessing	mutual exclusion
51	Remote computing services involves the use of timesharing and	multiprocessing	interactive processing	batch processing	real time processing	batch processing
52	A series of statements explaining how the data is to be processed is called	instruction	compiler	program	interpretor	program
53	Banker's algorithm deals with	deadlock prevention	deadlock avoidance	deadlock recovery	mutual exclusion	deadlock avoidance
54	Which is non pre-emptive	Round robin	FIFO	MQS	MQSF	FIFO
55	A hardware device which is capable of executing a sequence of instructions, is known as	CPU	ALU	CU	Processor	Processor
56	Distributed systems should	high security	have better resource sharing	better system utilization	low system overhead	have better resource sharing
57	Which of the following is always there in a computer	Batch system	Operating system	Time sharing system	Controlling system	Operating system

58	Which of following is not an advantage of multiprogramming	increased throughput	shorter response time	ability to assign priorities of jobs	decreased system overload	decreased system overload
59	In which of the following usually a front end processor is used	Virtual storage	Timesharing	Multiprogamming	Multithreading	Timesharing
60	Remote computing services involves the use of	multiprocessing	multiprogramming	batch processing	real time processing	batch processing
61	Banker's algorithm for resource allocation deals with	deadlock prevention	deadlock aviodance	deadlock recovery	circular wait	deadlock aviodance
62	In Unix, Which system call creates the new process?	Fork	Create	New	Old	Fork
63	The queue has finite length n; thus, at most n messages can reside in it is known as	Zero capacity	Bounded capacity	Unbounded capacity	multiprocessing	Bounded capacity
64	The queue has potentially infinite length; thus, any number of messages can wait in it is known as	Zero capacity	Bounded capacity	Unbounded capacity	multiprocessing	Unbounded capacity
65	Each process accessing the shared data excludes all others from doing so simultaneously and this is called	mutual exclusion	multiprocessing	real time processing	multiprogramming	mutual exclusion
	_____ OS runs on Apple produced hardware	Unix	Linux	Linux Mint	Apple	Apple
	PC _____ supports complex games	OS	PMT	PCB	Software	OS
66	The primary job of an OS is to _____	command resource	manage resource	provide utilities	Be user friendly	manage resource