Semester – III

18CAU302 **OPERATING SYSTEMS** **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:

- Understand the principles of concurrency and synchronization,.
- Understand basic resource management techniques.
- Understand the implementation of fundamental OS structures, including Threads, processes, system calls, scheduling, virtual memory, and file systems

Course Outcome:

- Appreciate the role of operating system as System software.
- Compare the various algorithms and comment about performance of various algorithms used for management of memory, CPU scheduling, File handling and I/O operations.
- Apply various concept related with Deadlock to solve problems related with Resources allocation, after checking system in Safe state or not.
- To appreciate role of Process synchronization towards increasing throughput of system.
- Describe the various Data Structures and algorithms used by Different Oss like Windows XP , Linux and Unix pertaining with Process , File , I/O management.
- To control the behavior of OS by writing Shell scripts.

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 aCSUess Authorization.

Suggested readings

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

Websites

1. www.cs.columbia.edu/~nieh/teaching/e6118_s00/
2. www.clarkson.edu/~jnm/cs644pages.cs.wisc.edu/~remzi/Classes/736/Fall2002/

Question Paper Pattern:

CIA	**Max.Marks : 50**
Part A	Objective type questions : 20 x 1 = 20 Marks
Part B	Answer all the questions Either/Or : 3 x 2 = 6 Marks
Part C	Answer all the questions : 3 x 8 = 24 Marks

ESE	**Max.Marks : 60**
Part A	Objective type questions : 20 x 1 = 20 Marks
Part B	Answer all the questions Either/Or : 5 x 6 = 30 Marks
Part C	Answer all the questions : 1 x 10 = 10 Marks

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Established Under Section 3 of UGC Act 1956)

EachanariPost, Coimbatore – 641 021. INDIA

Phone: 0422-2611146, 2611082 Fax No: 0422 -2611043

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

16CAU302 OPERATING SYSTEMS

LECTURER PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT I	
1	1	Basic OS Function	W1
2	1	Resource Abstraction	T1:4-6,W1
3	1	Types of Operating System	T1:19
4	1	Multiprogramming System	T1:32-35, W1
5	1	Batch Systems	T1:19-20
6	1	Time Sharing Systems	W1
7	1	Operating System for personal computers and Workstations	W1,W2
			W3
8	1	Process Control & Real time Systems	T2:34, W1
9	1	Discussion of Important Questions	
Total No. of Hours planned for Unit I			**9 hrs**

TEXT BOOK:

T1: Silberschatz,A., Galvin, P.B., Gagne,G.,(2008). *Operating Systems Concepts*, (8th ed.), John Wiley Publications.

T2: Tanenbaum, A.S., (2007). *Modern Operating Systems*, (3rd ed.), Pearson Education.

WEBSITES

W1:www.tutorialspoint.com>os_overview.htm

W2: www.comptechdoc.org>basictut>osintro

W3: www.dtic.mil>dtic>fulltext

W4:www.tutorialspoint.com/operating_system/os_processes.htm

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Established Under Section 3 of UGC Act 1956)

EachanariPost, Coimbatore – 641 021. INDIA

Phone: 0422-2611146, 2611082 Fax No : 0422 -2611043

16CAU302 OPERATING SYSTEMS

LECTURER PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT II	
1	1	Operating System Organization : Processes and user modes , kernels .	T1:20-23
2	1	System calls and System Programs	T1:55-58, 66-68,
3	1	Process Management: System view of the process and resources	T1:101-110, W3
4	1	Process Abstraction and process Hierarchy	T2:81-93, W3
5	1	Threads – Threading Issues	T1:153-157, T1:165-170,T2:93-115,W2
		Thread Libraries	T1:159-164,T2:93-115,W2
6	1	Process Scheduling – Non pre emptive and Pre emptive Scheduling algorithms , concurrent and processes	T1:105-110, T2:143-161
7	1	Critical section & Semaphore	T1:227-229, T2:117,866,
8	1	Methods for Inter-process communication	T1:897-898, T2: 115-126
		Deadlocks	T1:294-305, T2:435-439,W2
9	1	Discussion of Important Questions	
Total No. of Hours planned for Unit II			**9 hrs**

TEXT BOOK:

T1: Silberschatz,A., Galvin, P.B., Gagne,G.,(2008). *Operating Systems Concepts*, (8th ed.), John Wiley Publications.

T2: Tanenbaum, A.S., (2007). *Modern Operating Systems*, (3rd ed.), Pearson Education.

WEBSITES

W1:www2.cs.uic.edu>~jbell>4_Threads

W2: www.studytonight.com>operating-system

W3: www.geeksforgeeks.org>gate-notes-operating system-process-management

16CAU302 OPERATING SYSTEMS

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Established Under Section 3 of UGC Act 1956)
EachanariPost, Coimbatore – 641 021. INDIA
Phone: 0422-2611146, 2611082 Fax No : 0422 -2611043

LECTURER PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT III	
1	1	Memory management – Physical address space	T1:315-320, T2: 173-185
2	1	Virtual address space	T1:359-360, T2: 185
3	1	Memory Allocation Strategies	T1:315-327,T2:173-182
4	1	Fixed Partitions and Variable Partitions	T1:325-326,W1
5	1	Paging	T1:328-337, T2:186-187,214,225
		Structure of the Page Table	T1:337-341,T2:187-195
		Segmentation	T1:342-345,W2
6	1	Virtual memory – Demand Paging, Copy on Write	T1:357-368,T2:186-196
7	1	Virtual memory – Page Replacement and Allocation of frames	T1:369-381,T2:197-207
8	1	Virtual memory- Thrashing and Memory Mapped files	T1:386-395
9	1	Recapitulation and discussion of Important Questions	
Total No. of Hours planned for Unit III			**9 hrs**

TEXT BOOK:

T1: Silberschatz,A., Galvin, P.B., Gagne,G.,(2008). *Operating Systems Concepts*, (8th ed.), John Wiley Publications.

T2: Tanenbaum, A.S., (2007). *Modern Operating Systems*, (3rd ed.), Pearson Education.

WEBSITES

W1:u.cs.biu.ac.il>~ariel>ppts>os7-2-rea

W2: www.tutorialspoint.com>operating-system/os-memory-management-using-segmentation

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

16CAU302 OPERATING SYSTEMS
LECTURER PLAN

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Established Under Section 3 of UGC Act 1956)
EachanariPost, Coimbatore – 641 021. INDIA
Phone: 0422-2611146, 2611082 Fax No : 0422 -2611043

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT IV	
1	1	File and I/O Management	T1:64-67,421
2	1	Directory Structure and File Operations	T1:423-425, T2:266-270
3	1	File Operations - Example	T2:262-263
4	1	File Allocation Methods Contiguous and linked	T1:471-476 , W3
5	1	File Allocation Methods Management	T1:476-479,T1:64,W1,W3
6	1	File Allocation Methods – Indexed and Device –	T1:64 ,W1
7	1	Device type , Storage access	T2:262-263
8	1	Device management – Device drivers detection,	W1
		Device management – IPC , driver interface	W2
9	1	Recapitulation and discussion of Important Questions	
Total No. of Hours planned for Unit IV			**9 hrs**

TEXT BOOK:

T1: Silberschatz,A., Galvin, P.B., Gagne,G.,(2008). *Operating Systems Concepts*, (8th ed.), John Wiley Publications.

T2: Tanenbaum, A.S., (2007). *Modern Operating Systems*, (3rd ed.), Pearson Education.

WEBSITES

W1:wiki.ordev.org/Device-Management

W2: www.dauniv.ac.in>EmbsysRevEd_PPTs

W3: www.geeksforgeeks.org>File-allocation

16CAU302 OPERATING SYSTEMS

LECTURER PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT V	
1	1	Protection and Security Introduction	T1:629,W3
2	1	Policy mechanism	T1:620-623
3	1	Authentication –,	T1:639-640 , T2:651-654
4	1	Password authentication	T1:639-640 , T2:651-654
5	1	biometric Authentication	T1:639-640 , T2:651-654
6	1	Encrypted and one time passwords	T1:661-662
7	1	Program threats	T1:663-664
8	1	Internal Access Authorization	W1,W2
9	1	Recapitulation and discussion of Important Questions	
10	1	Discussion of previous year ESE Question Paper	
11	1	Discussion of previous year ESE Question Paper	
12	1	Discussion of previous year ESE Question Paper	
Total No. of Hours planned for Unit V			**12 hrs**

TEXT BOOK:

T1: Silberschatz,A., Galvin, P.B., Gagne,G.,(2008). *Operating Systems Concepts*, (8th ed.), John Wiley Publications.

T2: Tanenbaum, A.S., (2007). *Modern Operating Systems*, (3rd ed.), Pearson Education.

WEBSITES

W1:https://en,m.wkipedia.org>wiki>Authorization

W2: www.federia.urina.it

W3: www.tutorialspoint.com>os_security

UNIT – I

SYLLABUS

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.

INTRODUCTION TO OPERATING SYSTEMS

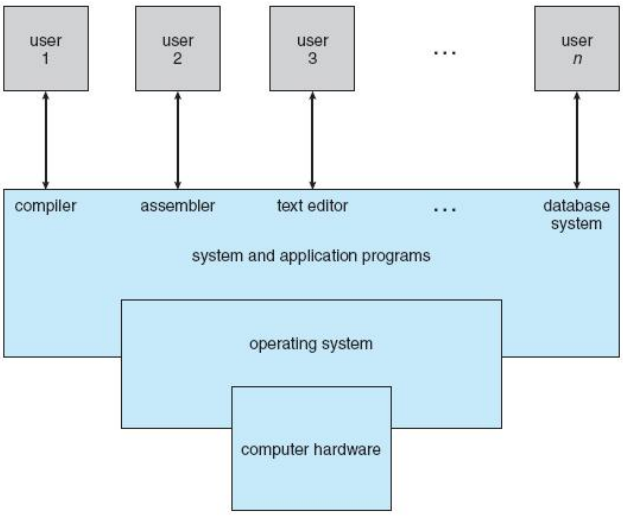
An Operating System (OS) is an interface between computer user and computer hardware. An operating system is software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers. Some popular Operating Systems include Linux, Windows, OS X, VMS, OS/400, AIX, z/OS, etc.

- **Definition**

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

What is an OS?

A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and the users. The hardware provides the basic computing resources for the system. The application programs define the ways in which these resources are used to solve users' computing problems.



System Components

The operating system controls the hardware and coordinates its use among the various application programs for the various users. OS cannot be defined exactly because, it differs in perspective.

- **User View**

The user's view of the computer varies according to the interface being used. In a personal Computing environment the goal of OS is "ease to use" with some attention paid for "resource-sharing ". In Computing environment like mainframes and minicomputer "Resource utilization" is maximized for computer availability and prevent user from sharing other's fair time. In environment like client server "Individual Usability" and "Resource sharing" are compromised in designing. In latest technologies like mobile and touch-pads, lap-tops the work of OS is to improve "battery-life" for better efficiency. In some systems like embedded system user's interaction is needed at initial phases only. The design principles of user view differ, so defining the work of OS cannot be made on their perspective.

- **System View**

In system (Computer) point of view, the work of OS is involved with the efficiency of handling hardware or software resources. In context, an OS can be viewed as a "Resource allocator". A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on. The operating system acts as the

manager of these resources. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly.

An operating system can be viewed as a "Control Program" that manages the execution of user programs to prevent errors and improper use of the computer.

BASIC OS FUNCTION

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. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. 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

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

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.

Other Important Activities

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

- Security − By means of password and similar other techniques, it prevents unauthorized access to programs and data.

- Control over system performance − Recording delays between request for a service and response from the system.
- Job accounting − Keeping track of time and resources used by various jobs and users.
- Error detecting aids − Production of dumps, traces, error messages, and other debugging and error detecting aids.
- Coordination between other softwares and users − Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

The operating system is the core software component of your computer. It performs many functions and is, in very basic terms, an interface between your computer and the outside world. In the section about hardware, a computer is described as consisting of several component parts including your monitor, keyboard, mouse, and other parts. The operating system provides an interface to these parts using what is referred to as "drivers". This is why sometimes when you install a new printer or other piece of hardware, your system will ask you to install more software called a driver.

An operating system has three main functions: (1) manage the computer's resources, such as the central processing unit, memory, disk drives, and printers, (2) establish a user interface, and (3) execute and provide services for applications software.

- System tools (programs) used to monitor computer performance, debug problems, or maintain parts of the system.
- A set of libraries or functions which programs may use to perform specific tasks especially relating to interfacing with computer system components.
- The operating system makes these interfacing functions along with its other functions operate smoothly and these functions are mostly transparent to the user.
- The operating system underpins the entire operation of the modern computer.

RESOURCE ABSTRACTION

- Resource abstraction is the process of "hiding the details of how the hardware operates, thereby making computer hardware relatively easy for an application programmer to use"

OPERATING SYSTEM TYPES

- There are many types of operating systems. The most common is the Microsoft suite of operating systems. They include from most recent to the oldest:
- Windows XP Professional Edition - A version used by many businesses on workstations. It has the ability to become a member of a corporate domain.
- Windows XP Home Edition - A lower cost version of Windows XP which is for home use only and should not be used at a business.
- Windows 2000 - A better version of the Windows NT operating system which works well both at home and as a workstation at a business. It includes technologies which allow hardware to be automatically detected and other enhancements over Windows NT.
- Windows ME - A upgraded version from windows 98 but it has been historically plagued with programming errors which may be frustrating for home users.
- Windows 98 - This was produced in two main versions. The first Windows 98 version was plagued with programming errors but the Windows 98 Second Edition which came out later was much better with many errors resolved.
- Windows NT - A version of Windows made specifically for businesses offering better control over workstation capabilities to help network administrators.
- Windows 95 - The first version of Windows after the older Windows 3.x versions offering a better interface and better library functions for programs.

There are other worthwhile types of operating systems not made by Microsoft. The greatest problem with these operating systems lies in the fact that not as many application programs are written for them. However if you can get the type of application programs you are looking for, one of the systems listed below may be a good choice.

- Unix - A system that has been around for many years and it is very stable. It is primary used to be a server rather than a workstation and should not be used by anyone who does

not understand the system. It can be difficult to learn. Unix must normally run an a computer made by the same company that produces the software.

- Linux - Linux is similar to Unix in operation but it is free. It also should not be used by anyone who does not understand the system and can be difficult to learn.
- Apple MacIntosh - Most recent versions are based on Unix but it has a good graphical interface so it is both stable (does not crash often or have as many software problems as other systems may have) and easy to learn. One drawback to this system is that it can only be run on Apple produced hardware.

TYPES OF OPERATING SYSTEM

 Types of operating system which are commonly used

MULTI-PROGRAMMING SYSTEM

- The work of the server is to execute the job in sequence assigned by the users at their fair intervals. This is the first time the OS are programmed (Control Program or Handler) to handle the users with the required resources. The switching between the users and the allocation of same resources to multiple processes was the difficult task. There was plenty of algorithm design for this by various research sectors in this time which paved a new way for multi-processing.
- Multiprogramming is a rudimentary form of parallel processing in which several programs are run at the same time on a uniprocessor. Since there is only one processor, there can be no true simultaneous execution of different programs.

BATCH OPERATING SYSTEM

- The tasks are grouped as batch based on the priority specified by the user. Once the tasks are grouped they are executed as a batch by the machine. The duration of execution may be a week or even months. The tasks are grouped manually by a person and after proper execution the results are given to them by that person. The processing of OS is to just execute the task and not on scheduling.

- The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

TIME-SHARING OPERATING SYSTEMS

- Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.
- 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.
- Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation.
- That is, if n users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most. The operating system uses CPU scheduling and multiprogramming to provide each

user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows −

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

REAL TIME OPERATING 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 likes undersea exploration and planetary rovers, etc.

DISTRIBUTED OPERATING SYSTEM

- Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.
- The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as loosely coupled systems or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Distributed Systems

- A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the system maintains. Access to a shared resource increases computation speed, functionality, data availability, and reliability. Some operating systems generalize network access as a form of file access, with the details of networking contained in the

network interface's device driver. Distributed systems depend on networking for their functionality.

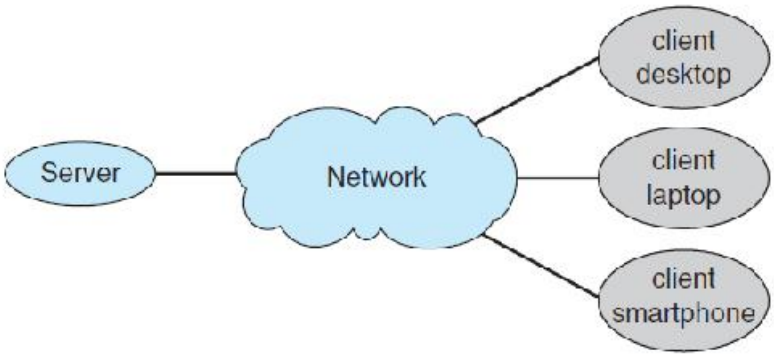
- Networks vary by the protocols used, the distances between nodes, and the transport media. TCP/IP is the most common network protocol, and it provides the fundamental architecture of the Internet. Most operating systems support TCP/IP, including all general-purpose ones. The media to carry networks are equally varied. They include copper wires, fiber strands, and wireless transmissions between satellites, microwave dishes, and radios.
- A network operating system is an operating system that provides features such as file sharing across the network, along with a communication scheme that allows different processes on different computers to exchange messages. A computer running a network operating system acts autonomously from all other computers on the network, although it is aware of the network and is able to communicate with other networked computers.

Client-Server Computing: As PCs have become faster, more powerful, and cheaper, designers have shifted away from centralized system architecture. Terminals connected to centralized systems are now being supplanted by PCs and mobile devices. Correspondingly, user-interface functionality once handled directly by centralized systems is increasingly being handled by PCs, quite often through a web interface. As a result, many of today's systems act as server systems to satisfy requests generated by client systems. This form of specialized distributed system, called a client–server system

Server systems can be broadly categorized as compute servers and file servers:

• The compute-server system provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client. A server running a database that responds to client requests for data is an example of such a system.

• The file-server system provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers.

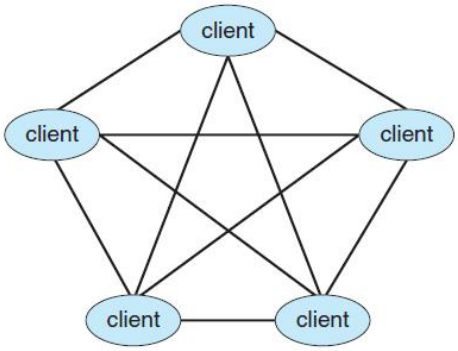


Client-Server Model

Peer to peer Systems

Another structure for a distributed system is the peer-to-peer (P2P) system model. In this model, clients and servers are not distinguished from one another. Instead, all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service. Peer-to-peer systems offer an advantage over traditional client-server systems. In a client-server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network. Determining what services are available is accomplished in one of two general ways:

- When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service. The remainder of the communication takes place between the client and the service provider.
- An alternative scheme uses no centralized lookup service. Instead, a peer acting as a client must discover what node provides a desired service by broadcasting a request for the service to all other nodes in the network. The node (or nodes) providing that service responds to the peer making the request. To support this approach, a discovery protocol must be provided that allows peers to discover services provided by other peers in the network.



Peer-Peer with no-centralized Machine

Skype is another example of peer-to-peer computing. It allows clients to make voice calls and video calls and to send text messages over the Internet using a technology known as voice over IP (VoIP). Skype uses a hybrid peer- to- peer approach. It includes a centralized login server, but it also incorporates decentralized peers and allows two peers to communicate.

Network operating System

- A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.
- Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

POSSIBLE QUESTIONS

UNIT – I

PART – A (20 MARKS)

(Q.NO 1 TO 20 Online Examinations)

PART – B (2 MARKS)

1. What is an Operating System?
2. List the Basic OS Functions
3. What is meant by Process Control?
4. What is Process?
5. List the types of Operating System
6. What is meant by Resource abstraction

PART – C (6 MARKS)

1. Discuss in detail about Real time System
2. Explain the Types of Operating System.
3. Explain Basic OS Function
4. Discuss in detail about Process Control Block
5. Explain about Distributed Systems
6. Explain the Components of Operating Systems
7. Explain about Batch System
8. Explain about Multiprogramming Systems
9. Discuss in detail about Resource Abstraction
10. Explain about Time Sharing Systems

KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore – 641 021.

(For the Candidates admitted from 2018 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

UNIT - I : (Objective Type Multiple choice Questions each Question carries one Mark)

OPERATING SYSTEMS

PART - A (Online Examination)

Questions	Opt1	opt2	opt3	opt4	KEY
computer hardware and acts as an intermediary	acceleration	Operating Syster	compiler	logical transcation	System
_____manages the execution of user programs to	resource all	work station	main frame	control program	control program
____were the first computers used to tackle many	Mainframe computer system	Mainframe computer service	multiframe computer system	multiframe computer service	Mainframe computer system
_____contains the address of an instruction to be fetched from memory	Program cou	Instruction regist	Control registers	Status registers	Instruction regis
_____ is also known as parallel systems	Multiproces sor systems	desktop systems	Time sharing systems	Multiprogrammed systems	Multiprocesso r systems
_____operating systems are even more co	Time-sharin	desktop systems	Multiprogrammed systems	Multiprocessor systems	Time-sharing
_____ operating system keeps several jobs in memory simultaneously.	Time-sharin	desktop systems	Multiprogrammed systems	Multiprocessor systems	Multiprogram med systems
_____can save more money than multiple sin	Multiproces sor systems	desktop systems	Time sharing systems	Multiprogrammed systems	Multiprocesso r systems
The most common multiple-processor systems now u	symmetric multiproces sing	asymmetric multiprocessing	multithreading	multiprogramming	symmetric multiprocessin g

Another form of a special-purpose operating system i	real-time system	distributed operating system	Process states	multiframe computer system	real-time system
The assignment of the CPU to the first process on the	graceful degradation	Time-sharing	dispatching	Multiprocessor systems	dispatching
The manifestation of a process in an operating system	Process state transitions	process control block	child process	cooperating processes	process control block
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
Which operating system reacts in the actual time	Batch system	Quick response system	Real time system	Time sharing system	Real time system
The primary job of an OS is to _____	command resource	manage resource	provide utilities	Be user friendly	manage resource
The term " Operating System " means _____	A set of programs which controls computer working	The way a computer operator works	Conversion of high-level language in to machine level language	The way a floppy disk drive operates	A set of programs which controls computer working
With ..... more than one process can be running simultaneously each on a different processer.	Multiprogramming	Uniprocessing	Multiprocessing	Uniprogramming	Multiprogramming

The two central themes of modern operating system are	Multiprogramming and Distributed processing	Multiprogramming and Central Processing	Single Programming and Distributed processing	None of above	Multiprogramming and Distributed processing
…….. is a example of an operating system that support single user process and single thread	UNIX	MS-DOS	OS/2	Windows 2000	MS-DOS
The operating system of a computer serves as a software interface between the user and the _____.	Hardware	Peripheral	Memory	Screen	Hardware
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
The main function of the command interpreter is:	to get and execute the next user-specified command	to provide the interface between the API and application program	to handle the files in operating system	none of the mentioned	to get and execute the next user-specified command
As OS that has strict time constraints	Sensor Node OS	Real Time OS	Mainframe OS	Timesharing OS	Real Time OS
The OS that groups similar jobs is called as	Network OS	Distributed OS	Mainframe OS	Batch OS	Batch OS
_____ 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
A system program that combines the separately compiled modules of a program into a form suitable for execution	assembler	linking loader	cross compiler	load and go	linking loader
A _____manages the execution of user programs to prevent errors and improper use of the computer.	Control program	Managing Program	allocating program	User program	Control program

_____ is a program associated with the operating system but are not part of the kernel,	System Program	User program	System calls	Functions	System Program
General-purpose computers run most of their programs from rewriteable memory, called as _____	Floppy disk	ROM	Random access Memory	Hard disk	Random access Memory
On systems with multiple command interpreters to choose from, the interpreters are known as _____	GUI	shells	Signal	Command	shells
The term PDA is _____	Personal Digital Assistant	Personal Data Assistant	Personal Data Accountant	Private Digital Assistant	Personal Digital Assistant
_____ handle large numbers of small requests	Batch systems	Time sharing	Transaction-processing systems	Distributed systems	Transaction-processing systems
The occurrence of an event is usually signaled by an _____from either the hardware or the software.	interrupt	signal	service	routine	interrupt
Operating systems have a _____for each device controller	Process	device driver	controller	allocator	device driver
CPU design that includes multiple computing cores on a single chip. Such multiprocessor systems are termed _____	multicore	uniprocessor	singlecore	multichips	multicore
logical storage unit is called as _____	folder	file	RAM	ROM	file
_____ is any mechanism for controlling the access of processes or users to the resources defined by a computer system.	Protection	authorization	policy	privacy	Protection
A _____is an operating system that provides features such as file sharing across the network.	network operating system	Distributed OS	Parallel OS	Sensor OS	network operating system
_____operating systems are even more complex than multi programmed operating systems.	Time-sharing	desktop systems	Multiprogrammed systems	Multiprocessor systems	Time-sharing

_____can save more money than multiple single-processor systems	Multiprocessor systems	desktop systems	Time sharing systems	Multiprogrammed systems	Multiprocessor systems
_____ is also known as parallel systems or tightly coupled systems	Multiprocessor systems	desktop systems	Time sharing systems	Multiprogrammed systems	Multiprocessor systems
Another form of a special-purpose operating system is the	real-time system	distributed operating system	Process states	multiframe computer system	real-time system
The message-passing facility in Windows 2000 is called	MUTUAL EXCLUSION	Buffering	local procedure call facility	CRITICAL SECTIONS	local procedure call facility
Which process is known for initializing a microcomputer with its OS	cold booting	boot recording	booting	warm booting	booting
A series of statements explaining how the data is to be processed is called	instruction	compiler	program	interpretor	program
Distributed systems should	high security	have better resource sharing	better system utilization	low system overhead	have better resource sharing
Which of the following is always there in a computer	Batch system	Operating system	Time sharing system	Controlling system	Operating system
When did IBM released the first version of its disk operating system DOS version 1.0	1981	1982	1983	1984	1981
The kernel is a_____	memory manager	resource manager	file manager	directory manager	resource manager
_____contains the address of an instruction to be fetched from memory	Program counter (PC)	Instruction register (IR)	Control registers	Status registers	Instruction register (IR)
_____contains the instruction most recently fetched.	Program counter (PC)	Instruction register (IR)	Control registers	Status registers	Program counter (PC)
If a process fails, most operating system write the error information to a	log file	another running process	new file	none of the mentioned	log file

The OS X has _____	monolithic kernel	hybrid kernel	microkernel	monolithic kernel with modules	hybrid kernel
Which Operating system does not support long file names	OS/2	Windows 95	MS-DOS	Windows NT	MS-DOS
Which Operating system does not support networking between computers	Windows 3.1	Windows 95	Windows 2000	Windows NT	Windows 3.1
Which Operating system is better for implementing client server network	MS DOS	Windows 95	Windows 98	Windows 2000	Windows 2000
_____is the commercial UNIX-based operating system of Sun Microsystems.	Solaris	UNIX	Linux	Macintosh	Solaris
_____ is an example of an open-source operating system	GNU/Linux	Windows 3.1	Windows NT	Macintosh	GNU/Linux
In Operating System_____ hides the details of hov	Resource manager	Resource Abstraction	Resource Hiding	Information Hiding	Resource Abstraction

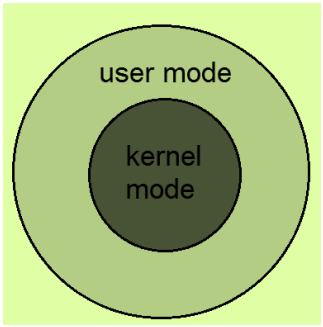
UNIT – II

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.

PROCESSOR AND USER MODES

• Mode bit: Supervisor or User mode • Supervisor mode – Can execute all machine instructions – Can reference all memory locations • User mode – Can only execute a subset of instructions – Can only reference a subset of memory locations



Modes supported by the operating system

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.

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.

SYSTEM CALLS AND SYSTEM PROGRAMS

- 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 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.

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

SYSTEM CALL

When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a **system call**.

When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **context switch**.

Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Generally, system calls are made by the user level programs in the following situations:

- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer.

In a typical UNIX system, there are around 300 system calls. Some of them which are important ones in this context are described below.

SYSTEM PROGRAMS

These programs are not usually part of the OS kernel, but are part of the overall operating system.

File Management

These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.

Status Information

Some programs simply request the date and time, and other simple requests. Others provide detailed performance, logging, and debugging information. The output of these files is often sent to a terminal window or GUI window

File modification

Programs such as text editors are used to create, and modify files.

Communications

These programs provide the mechanism for creating a virtual connect among processes, users, and other computers. Email and web browsers are a couple examples.

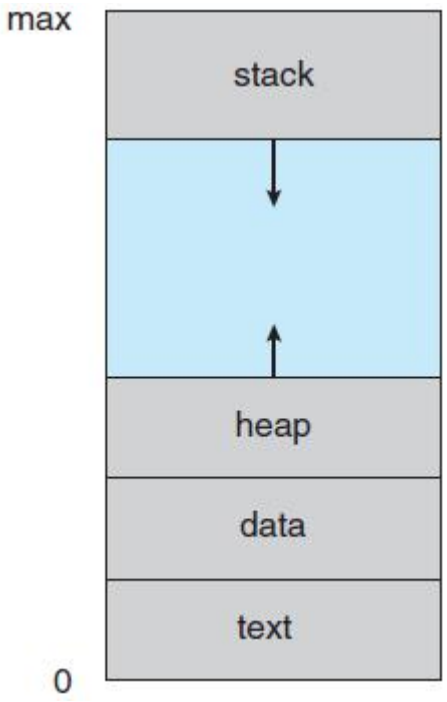
PROCESS MANAGEMENT

Process Concept

Process is a program that is in execution. It is defined as unit of work in modern systems. A batch system executes jobs, whereas a time-shared system has user programs, or tasks. Even on a single-user system, a user may be able to run several programs at one time: a word processor, a Web browser, and an e-mail package. And even if a user can execute only one program at a time, such as on an embedded device that does not support multitasking, the operating system may need to support its own internal programmed activities, such as memory management. In many respects, all these activities are similar, so we call all of them processes.

Process in memory

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.



Process in Memory

A program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file). In contrast, 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.

Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

• New. The process is being created.

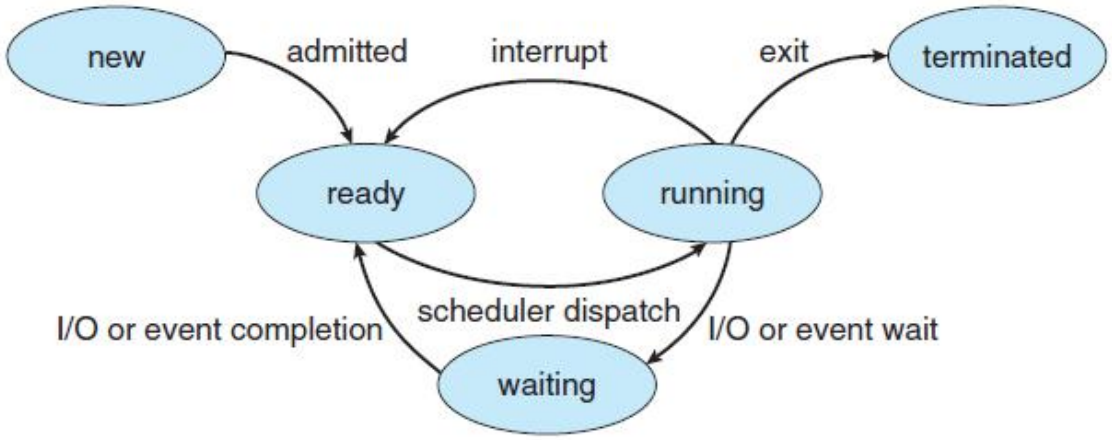
• Running. Instructions are being executed.

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

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

• 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 the following Figure.



Process State Diagram

Process Control Block (PCB)

Each process is represented in the operating system by a process control block (PCB)—also called a task control block. It contains many pieces of information associated with a specific process, including these: Process state. The state may be new, ready, running, and waiting, halted, and so on.

- **Program counter.** The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers.** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-

purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

- **CPU-scheduling information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

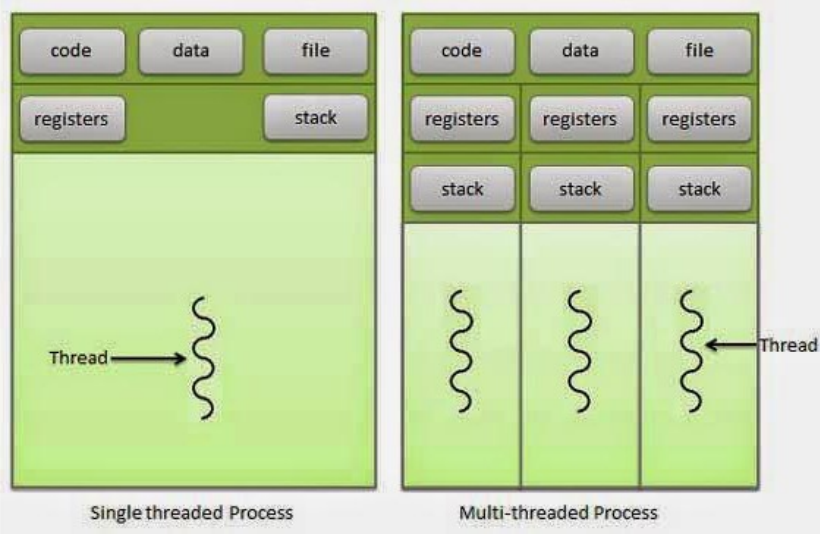
Process Control Block (PCB)

- **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system
- **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

THREAD

- A thread is a flow of execution through the process code, with its own program counter, system registers and stack. A thread is also called a light weight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. Following figure shows the working of the single and multithreaded processes.



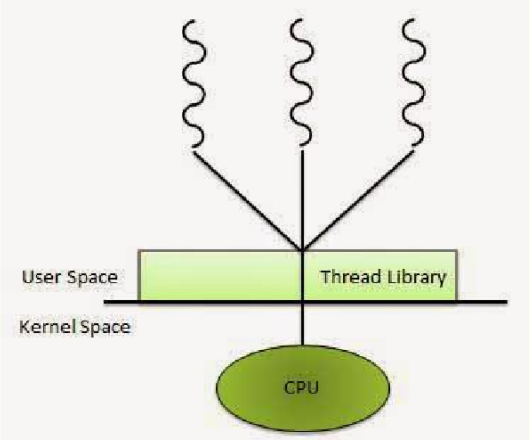
Advantages of Thread

- Thread minimizes context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- Economy- It is more economical to create and context switch threads.
- Utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

- Threads are implemented in following two ways
- User Level Threads -- User managed threads
- Kernel Level Threads -- Operating System managed threads acting on kernel, an operating system core.
- User Level Threads

- In this case, application manages thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application begins with a single thread and begins running in that thread.



ADVANTAGES

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

DISADVANTAGES

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

Kernel Level Threads

In this case, thread management done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any

application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

ADVANTAGES

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

DISADVANTAGES

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

Multithreading Models

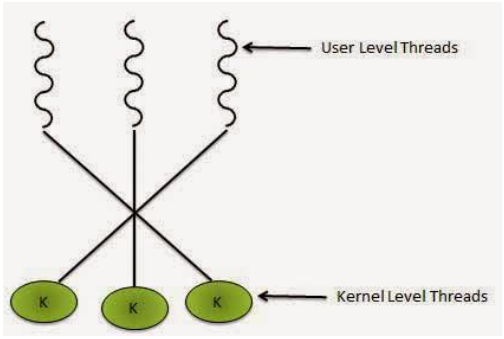
Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many too many relationships.
- Many to one relationship.
- One to one relationship.

Many to Many Model

- In this model, many user level threads multiplex to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine.

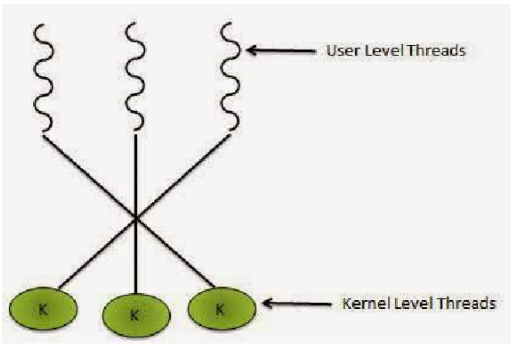
- Following diagram shows the many to many models. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.



Many to One Model

- Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocks. Only one thread can access the Kernel at a time,so multiple threads are unable to run in parallel on multiprocessors.

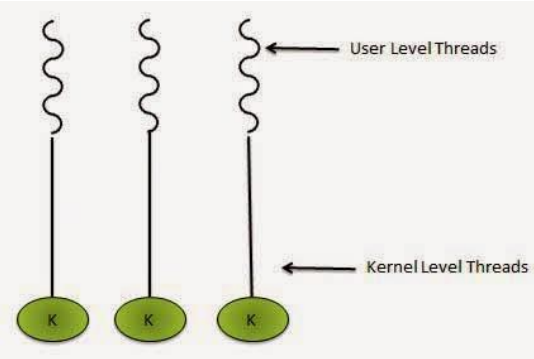
- If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.



One to One Model

- There is one to one relationship of user level thread to the kernel level thread. This model provides more concurrency than the many to one model. It is also another thread to run when a thread makes a blocking system call. It supports multiple thread to execute in parallel on microprocessors.

- Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



THREADING ISSUES OPERATING SYSTEMS

1. The fork() and exec() System Calls

- If one thread in a program calls fork(), does the new process duplicate all threads, or is the new process single-threaded? Some UNIX systems have chosen to have two versions of fork(), one that duplicates all threads and another that duplicates only the thread that invoked the fork() system call.

- If a thread invokes the exec() system call, the program specified in the parameter to exec () will replace the entire process-including all threads.

2. Cancellation

- Thread cancellation is the task of terminating a thread before it has completed. For example, if multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be canceled.
- A thread that is to be canceled is often referred to as the **target thread**.
- Cancellation of a target thread may occur in two different scenarios:
  - **Asynchronous cancellation**. One thread immediately terminates the target thread.
  - **Deferred cancellation.** The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.
- The difficulty with cancellation occurs in situations where resources have been allocated to a canceled thread or where a thread is canceled while in the midst of updating data it is sharing with other threads.

3. Signal Handling

- A signal is used in UNIX systems to notify a process that a particular event has occurred. All signals, whether synchronous or asynchronous, follow the same pattern:
- A signal is generated by the occurrence of a particular event.
- A generated signal is delivered to a process.
- Once delivered, the signal must be handled.
- Examples of synchronous signals include illegal memory access and division by 0. If a running program performs either of these actions, a signal is generated.
- Every signal has a **default signal handler** that is run by the kernel when handling that signal. This default action can be overridden by a **user defined signal handler** that is called to handle the signal.
- Handling signals in single-threaded programs is straightforward: signals are always delivered to a process. However, delivering signals is more complicated in

multithreaded programs, where a process may have several threads. Where, then, should a signal be delivered?

- In general the following options exist:
  - Deliver the signal to the thread to which the signal applies.
  - Deliver the signal to every thread in the process.
  - Deliver the signal to certain threads in the process.
  - Assign a specific thread to receive all signals for the process.

4. Thread Pools

- The first **issue** concerns the amount of time required to create the thread prior to servicing the request, together with the fact that this thread will be discarded once it has completed its work.
- The second **issue** is more troublesome: if we allow all concurrent requests to be serviced in a new thread, we have not placed a bound on the number of threads concurrently active in the system. Unlimited threads could exhaust system resources, such as CPU time or memory. One solution to this problem is to use a **thread pool**.
- The general idea behind a thread pool is to create a number of threads at process startup and place them into a pool, where they sit and wait for work.
- Thread pools offer these benefits:
  - Servicing a request with an existing thread is usually faster than waiting to create a thread.
  - A thread pool limits the number of threads that exist at any one point. This is particularly important on systems that cannot support a large number of concurrent threads.

5. Thread-Specific Data

- Threads belonging to a process share the data of the process. Indeed, this sharing of data provides one of the benefits of multithreaded programming.

- However, in some circumstances, each thread might need its own copy of certain data. We will call such data **thread specific data**.
- For example, in a transaction-processing system, we might service each transaction in a separate thread. Furthermore, each transaction might be assigned a unique identifier. To associate each thread with its unique identifier, we could use thread-specific data.

PROCESS SCHEDULING

Definition

- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Process scheduling is an essential part of a Multiprogramming operating system. Such operating systems allow more than one process to be loaded into the executable memory at a time and loaded process shares the CPU using time multiplexing.

Scheduling Queues

- Scheduling queues refers to queues of processes or devices. When the process enters into the system, then this process is put into a job queue. This queue consists of all processes in the system. The operating system also maintains other queues such as device queue. Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has its own device queue.

This figure shows the queuing diagram of process scheduling.

- Queue is represented by rectangular box.
- The circles represent the resources that serve the queues.
- The arrows indicate the process flow in the system.

Queues are of two types

- Ready queue
- Device queue

A newly arrived process is put in the ready queue. Processes waits in ready queue for allocating the CPU. Once the CPU is assigned to a process, then that process will execute. While executing the process, any one of the following events can occur.

- The process could issue an I/O request and then it would be placed in an I/O queue.
- The process could create new sub process and will wait for its termination.
- The process could be removed forcibly from the CPU, as a result of interrupt and put back in the ready queue.

Two State Process Model

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

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

Schedulers

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

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

Long Term Scheduler

- It is also called job scheduler. Long term scheduler determines which programs are admitted to the system for processing. Job scheduler selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling. The primary objective of the job scheduler is to provide a balanced mix of

jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

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

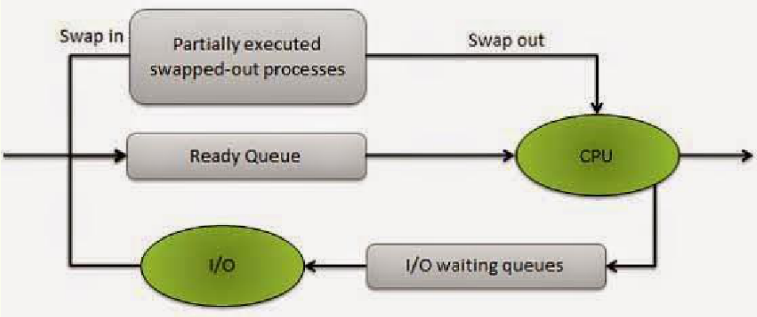
Short Term Scheduler

- It is also called CPU scheduler. Main objective is increasing system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects process among the processes that are ready to execute and allocates CPU to one of them.

- Short term scheduler also known as dispatcher, execute most frequently and makes the fine grained decision of which process to execute next. Short term scheduler is faster than long term scheduler.

Medium Term Scheduler

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



Running process may become suspended if it makes an I/O request. Suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other process, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Comparison between Scheduler

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

SCHEDULING (PREEMPTIVE AND NONPREEMPTIVE)

- *A* final issue to be considered with multithreaded programs concerns communication between the kernel and the thread library, which may be required by the many-to-many and two-level models.
- Such coordination allows the number of kernel threads to be dynamically adjusted to help ensure the best performance.

General Goals

Fairness

Fairness is important under all circumstances. A scheduler makes sure that each process gets its fair share of the CPU and no process can suffer indefinite postponement. Note that giving equivalent or equal time is not fair. Think of *safety control* and *payroll* at a nuclear plant.

Policy Enforcement

The scheduler has to make sure that system's policy is enforced. For example, if the local policy is safety then the *safety control processes* must be able to run whenever they want to, even if it means delay in *payroll processes*.

Efficiency

Scheduler should keep the system (or in particular CPU) busy cent percent of the time when possible. If the CPU and all the Input/Output devices can be kept running all the time, more work gets done per second than if some components are idle.

Response Time

A scheduler should minimize the response time for interactive user.

Turnaround

A scheduler should minimize the time batch users must wait for an output.

Throughput

A scheduler should maximize the number of jobs processed per unit time.

Preemptive Vs Non preemptive Scheduling

The Scheduling algorithms can be divided into two categories with respect to how they deal with clock interrupts.

Non preemptive Scheduling

- A scheduling discipline is non preemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process.

Following are some characteristics of non preemptive scheduling

1. In non preemptive system, short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
2. In non preemptive system, response times are more predictable because incoming high priority jobs cannot displace waiting jobs.
3. In non preemptive scheduling, a scheduler executes jobs in the following two situations.
    a. When a process switches from running state to the waiting state.
    b. When a process terminates.

Preemptive Scheduling

- A scheduling discipline is preemptive if, once a process has been given the CPU can taken away.
- The strategy of allowing processes that are logically runable to be temporarily suspended is called Preemptive Scheduling and it is contrast to the "run to completion" method.

Schedule:

- A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

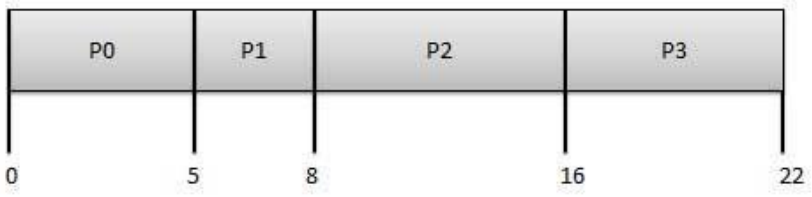
- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

- These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0= 0
P1	5 - 1= 4
P2	8 - 2= 6
P3	16 - 3= 13

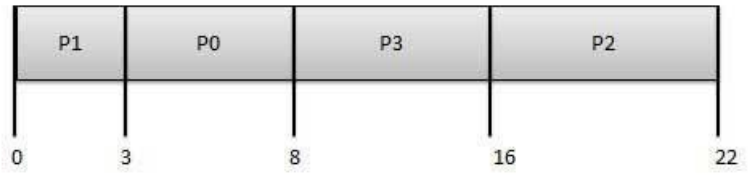
Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.

- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

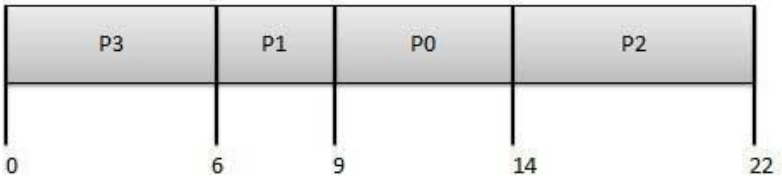
Average Wait Time: (3+0+14+5) / 4 = 5.50

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.

- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

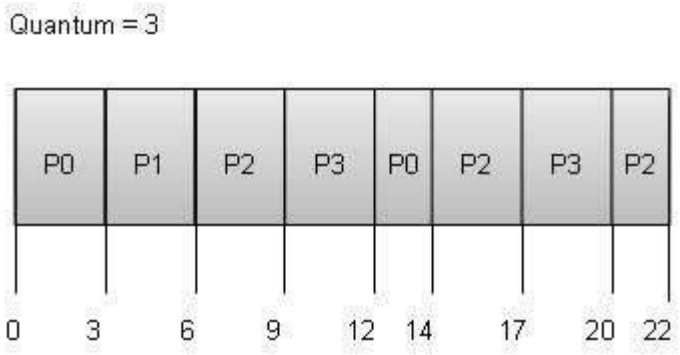
Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.

- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Context Switch

- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.
- When the scheduler switches the CPU from executing one process to execute another, the context switcher saves the content of all processor registers for the process being removed from the CPU, in its process descriptor. The context of a process is represented in the process control block of a process.
- Context switch time is pure overhead. Context switching can significantly affect performance as modern computers have a lot of general and status registers to be saved. Content switching times are highly dependent on hardware support. Context switch requires (n + m) bxK time units to save the state of the processor with n general registers, assuming b are the store operations are required to save n and m registers of two process control blocks and each store instruction requires K time units.

Some hardware systems employ two or more sets of processor registers to reduce the amount of context switching time. When the process is switched, the following information is stored.

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

CONCURRENT AND PROCESSES

Co-operation of Process

Processes executing concurrently in the operating system may be either independent processes or cooperating processes. A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other

processes is a cooperating process. There are several reasons for providing an environment that allows process cooperation:

• **Information sharing**. Since several users may be interested in the same piece of information (for instance, a shared file).It must provide an environment to allow concurrent access to such information.

• **Computation speedup**. If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing cores.

• **Modularity.** We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads,

• **Convenience.** Even an individual user may work on many tasks at the same time. For instance, a user may be editing, listening to music, and compiling in parallel.

Cooperating processes require an inter-process communication (IPC) mechanism that will allow them to exchange data and information. There are two fundamental models of inter-process communication: shared memory and message passing. In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region. In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes.

CRITICAL SECTION PROBLEM

Consider a system consisting of n processes {P0, P1, ..., Pn−1}. Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time.

The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section may be followed by an exit section. The remaining code is the remainder section.

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

1. **Mutual exclusion**. If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

2. **Progress**. If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. **Bounded waiting**. There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Two general approaches are used to handle critical sections in operating systems: preemptive kernels and non-preemptive kernels. A preemptive kernel allows a process to be preempted while it is running in kernel mode. A non-preemptive kernel does not allow a process running in kernel mode to be preempted; a kernel-mode process will run until it exits kernel mode, blocks, or voluntarily yields control of the CPU.

SEMAPHORE

Mutex locks, as we mentioned earlier, are generally considered the simplest of synchronization tools. In this section, we examine a more robust tool that can behave similarly to a mutex lock but can also provide more sophisticated ways for processes to synchronize their activities. A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait() and signal(). The wait() operation was originally termed P (from the Dutch proberen, —to test‖); signal() was originally called V (from verhogen, —to increment‖). The definition of wait() is as follows:

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

The definition of signal() is as follows: signal(S) {

```
S++;
}
```

All modifications to the integer value of the semaphore in the wait () and signal () operations must be executed indivisibly. That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value. In addition, in the case of wait(S), the testing of the integer value of S (S ≤ 0), as well as its possible modification (S--), must be executed without interruption.

Semaphore Usage

Operating systems often distinguish between **counting and binary semaphores**. The value of a **counting semaphore** can range over an unrestricted domain. The value of a binary semaphore can range only between 0 and 1. Thus, **binary semaphores** behave similarly to mutex locks. Counting semaphores can be used to control access to a given resource consisting of a finite number of instances. The semaphore is initialized to the number of resources available. Each process that wishes to use a resource performs a wait() operation on the semaphore (thereby decrementing the count). When a process releases a resource, it performs a signal () operation (incrementing the count). When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.

Deadlock and Starvation

The implementation of a semaphore with a waiting queue may result in a situation where two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes. The event in question is the execution of a signal() operation. When such a state is reached, these processes are said to be deadlocked. To illustrate this, consider a system consisting of two processes, P0 and P1, each accessing two semaphores, S and Q, set to the value 1: P0 P1

```
wait(S); wait(Q);
wait(Q); wait(S);
. .
. .
```

. .

signal(S); signal(Q);

signal(Q); signal(S);

Suppose that P0 executes wait(S) and then P1 executes wait(Q).When P0 executes wait(Q), it must wait until P1 executes signal(Q). Similarly, when P1 executes wait(S), it must wait until P0 executes signal(S). Since these signal() operations cannot be executed, P0 and P1 are deadlocked. We say that a set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.

The events with which we are mainly concerned here are resource acquisition and release Another problem related to deadlocks is indefinite blocking or starvation, a situation in which processes wait indefinitely within the semaphore. Indefinite blocking may occur if we remove processes from the list associated with a semaphore in LIFO (last-in, first-out) order.

METHODS OF INTER-PROCESS COMMUNICATION (IPC)

Inter-process communication using shared memory requires communicating processes to establish a region of shared memory. Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by these processes and are not under the operating system's control.

The processes are also responsible for ensuring that they are not writing to the same location simultaneously. Two types of buffers can be used. The unbounded buffer places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items. The bounded buffer assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Message-Passing Systems

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. It is particularly useful in a

distributed environment, where the communicating processes may reside on different computers connected by a network. For example, an Internet chat program could be designed so that chat participants communicate with one another by exchanging messages. A message-passing facility provides at least two operations:

send(message) receive(message)

Messages sent by a process can be either fixed or variable in size. If only fixed-sized messages can be sent, the system-level implementation is straightforward. This restriction, however, makes the task of programming more difficult. Conversely, variable-sized messages require a more complex system common kind of tradeoff seen throughout operating-system design. If processes P and Q want to communicate, they must send messages to and receive messages from each other: a communication link must exist between them. This link can be implemented in a variety of ways. We are concerned here not with the link's physical implementation (such as shared memory, hardware bus, or network, but rather with its logical implementation. Here are several methods for logically implementing a link and the send()/receive() operations:

- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

Naming

Processes that want to communicate must have a way to refer to each other. They can use either direct or indirect communication. Under direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send() and receive() primitives are defined as:

- send(P, message)—Send a message to process P.
- receive(Q, message)—Receive a message from process Q.

A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.

• A link is associated with exactly two processes.

• Between each pair of processes, there exists exactly one link.

This scheme exhibits symmetry in addressing; that is, both the sender process and the receiver process must name the other to communicate. A variant of this scheme employs asymmetry in addressing. Here, only the sender names the recipient; the recipient is not required to name the sender. In this scheme, the send () and receive () primitives are defined as follows:

• send (P, message)—Send a message to process P.

• receive (id, message)—Receive a message from any process. The variable id is set to the name of the process with which communication has taken place.

The disadvantage in both of these schemes (symmetric and asymmetric) is the limited modularity of the resulting process definitions. Changing the identifier of a process may necessitate examining all other process definitions. All references to the old identifier must be found, so that they can be modified to the new identifier. In general, any such hard-coding techniques, where identifiers must be explicitly stated, are less desirable than techniques involving indirection.

With indirect communication, the messages are sent to and received from mailboxes, or ports. A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed. Each mailbox has a unique identification. For example, POSIX message queues use an integer value to identify a mailbox. A process can communicate with another process via a number of different mailboxes, but two processes can communicate only if they have a shared mailbox. The send () and receive () primitives are defined as follows:

• send (A, message)—Send a message to mailbox A.

• receive (A, message)—Receive a message from mailbox A.

In this scheme, a communication link has the following properties:

• A link is established between a pair of processes only if both members of the pair have a shared mailbox.

• A link may be associated with more than two processes.

• Between each pair of communicating processes, a number of different links may exist, with each link corresponding to one mailbox.

Now suppose that processes P1, P2, and P3 all share mailbox A. Process P1 sends a message to A, while both P2 and P3 execute a receive() from A. Which process will receive the message sent by P1? The answer depends on which of the following methods we choose:

• Allow a link to be associated with two processes at most.

• Allow at most one process at a time to execute a receive () operation.

• Allow the system to select arbitrarily which process will receive the message (that is, either P2 or P3, but not both, will receive the message).

The system may define an algorithm for selecting which process will receive the message (for example, round robin, where processes take turns receiving messages). The system may identify the receiver to the sender. A mailbox may be owned either by a process or by the operating system.

If the mailbox is owned by a process (that is, the mailbox is part of the address space of the process), then we distinguish between the owner (which can only receive messages through this mailbox) and the user (which can only send messages to the mailbox). Since each mailbox has a unique owner, there can be no confusion about which process should receive a message sent to this mailbox. When a process that owns a mailbox terminates, the mailbox disappears.

Any process that subsequently sends a message to this mailbox must be notified that the mailbox no longer exists. In contrast, a mailbox that is owned by the operating system has an existence of its own. It is independent and is not attached to any particular process. The operating system then must provide a mechanism that allows a process to do the following:

• Create a new mailbox.

• Send and receive messages through the mailbox.

• Delete a mailbox.

The process that creates a new mailbox is that mailbox's owner by default. Initially, the owner is the only process that can receive messages through this mailbox. However, the ownership and receiving privilege may be passed to other processes through appropriate system calls. Of course, this provision could result in multiple receivers for each mailbox.

Synchronization

Communication between processes takes place through calls to send() and receive() primitives. There are different design options for implementing each primitive. Message passing may be either blocking or non blocking— also known as synchronous and asynchronous. (Throughout this text, you will encounter the concepts of synchronous and asynchronous behavior in relation to various operating-system algorithms.)

• Blocking send. The sending process is blocked until the message is received by the receiving process or by the mailbox.

• Non-blocking send. The sending process sends the message and resumes operation.

• Blocking receive. The receiver blocks until a message is available.

• Non-blocking receive. The receiver retrieves either a valid message or a null.

Different combinations of send () and receive () are possible. When both send () and receive () are blocking, we have a rendezvous between the sender and the receiver. The solution to the producer–consumer problem becomes trivial when we use blocking send () and receive () statements. The producer merely invokes the blocking send () call and waits until the message is delivered to either the receiver or the mailbox. Likewise, when the consumer invokes receive (), it blocks until a message is available.

Buffering

Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue. Basically, such queues can be implemented in three ways:

• Zero capacity. The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.

• Bounded capacity. The queue has finite length n; thus, at most n messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue (either the message is copied or a pointer to the message is kept), and the sender can continue execution without waiting. The link's capacity is finite, however. If the link is full, the sender must block until space is available in the queue.

• Unbounded capacity. The queue's length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

The zero-capacity case is sometimes referred to as a message system with no buffering. The other cases are referred to as systems with automatic buffering.

DEADLOCK

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. Request. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.

2. Use. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).

3. Release. The process releases the resource.

- For each use of a kernel-managed resource by a process or thread, the operating system checks to make sure that the process has requested and has been allocated the resource. A system table records whether each resource is free or allocated. For each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.
- A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set. The events with which we are mainly concerned here are resource acquisition and release. The resources may be either physical resources (for example, printers, tape drives, memory space, and CPU cycles) or logical resources (for example, semaphores, mutex locks, and files).

Deadlock Prevention

Mutual Exclusion

- The mutual exclusion condition must hold. That is, at least one resource must be non-sharable. Sharable resources, in contrast, do not require mutually exclusive

access and thus cannot be involved in a deadlock. Read-only files are a good example of a sharable resource. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file.

- A process never needs to wait for a sharable resource. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable. For example, a mutex lock cannot be simultaneously shared by several processes.

Hold and Wait

- To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.
- One protocol that we can use requires each process to request and be allocated all its resources before it begins execution. We can implement this provision by requiring that system calls requesting resources for a process precede all other system calls. An alternative protocol allows a process to request resources only when it has none.
- A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.
- Both these protocols have two main disadvantages. First, resource utilization may be low, since resources may be allocated but unused for a long period. In the example given, for instance, we can release the DVD drive and disk file, and then request the disk file and printer, only if we can be sure that our data will remain on the disk file. Otherwise, we must request all resources at the beginning for both protocols.
- Second, starvation is possible. A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

No Preemption

- The third necessary condition for deadlocks is that there be no preemption of resources that have already been allocated. To ensure that this condition does not hold, we can use the following protocol.

- If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources the process is currently holding are preempted. In other words, these resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting.

- The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting. Alternatively, if a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process and allocate them to the requesting process. If the resources are neither available nor held by a waiting process, the requesting process must wait.

- While it is waiting, some of its resources may be preempted, but only if another process requests them. A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were preempted while it was waiting.

Circular Wait

- The fourth and final condition for deadlocks is the circular-wait condition. One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration.

- To illustrate, we let R = {R1, R2, ..., Rm} be the set of resource types. We assign to each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering. Formally, we define a one-to-one function F: R→N, where N is the set of natural numbers. For example, if the set of resource types R includes tape drives, disk drives, and printers, then the function F might be defined as follows:

F(tape drive) = 1

F(disk drive) = 5

F(printer) = 12

- We can now consider the following protocol to prevent deadlocks: Each process can request resources only in an increasing order of enumeration. That is, a process can initially request any number of instances of a resource type —say, Ri . After that, the process can request instances of resource type Rj if and only if F(Rj) > F(Ri). For example, using the function defined previously, a process that wants to use the tape drive and printer at the same time must first request the tape drive and then request the printer.

- Alternatively, we can require that a process requesting an instance of resource type Rj must have released any resources Ri such that F(Ri) ≥ F(Rj). Note also that if several instances of the same resource type are needed, a single request for all of them must be issued. If these two protocols are used, then the circular-wait condition cannot hold. We can demonstrate this fact by assuming that a circular wait exists (proof by contradiction). Let the set of processes involved in the circular wait be {P0, P1, ..., Pn}, where Pi is waiting for a resource Ri , which is held by process Pi+1. (Modulo arithmetic is used on the indexes, so that Pn is waiting for a resource Rn held by P0.) Then, since process Pi+1 is holding resource Ri while requesting resource Ri+1, we must have F(Ri) < F(Ri+1) for all i. But this condition means that F(R0) < F(R1) < ... < F(Rn) < F(R0). By transitivity, F(R0) < F(R0), which is impossible. Therefore, there can be no circular wait.

Deadlock Avoidance:

- For avoiding deadlocks, it is to require additional information about how resources are to be requested. For example, in a system with one tape drive and one printer, the system might need to know that process P will request first the tape drive and then the printer before releasing both resources, whereas process Q will request first the printer and then the tape drive.

- With this knowledge of the complete sequence of requests and releases for each process, the system can decide for each request whether or not the process should wait in order to avoid a possible future deadlock. Each request requires that in making this decision the system consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

- The various algorithms that use this approach differ in the amount and type of information required. The simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.

Safe State:

- A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence.

- A sequence of processes <P1, P2, ... , Pn> is a safe sequence for the current allocation state if, for each Pi, the resource requests that Pi can still make can be satisfied by the currently available resources plus the resources held by all Pj, with j < i. In this situation, if the resources that Pi needs are not immediately available, then Pi can wait until all Pj have finished. When they have finished, Pi can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate. When Pi terminates, Pi+l can obtain its needed resources, and so on. If no such sequence exists, then the system state is said to be unsafe. A safe state is not a deadlocked state. Conversely, a deadlocked state is an unsafe state.

Resource-Allocation-Graph Algorithm

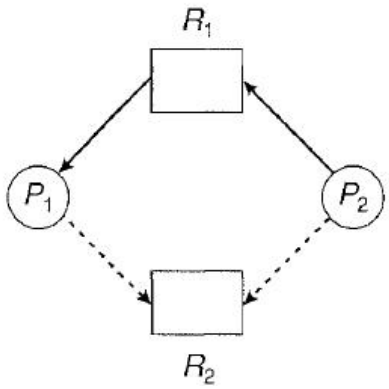
- If we have a resource-allocation system with only one instance of each resource type, we can use a variant of the resource-allocation graph defined for deadlock avoidance. In addition to the request and assignment edges already described, we introduce a new type of edge, called a claim edge.

- A claim edge Pi ~ Rj indicates that process Pi may request resource Rj at some time in the future. This edge resembles a request edge in direction but is represented in the graph by a dashed line. When process Pi requests resource R1, the claim edge P; -+ R1 is converted to a request edge. Similarly, when a resource R1 is released by P;, the assignment edge Rj -+ P; is reconverted to a claim edge P; -+ Rj.

- We note that the resources must be claimed a priori in the system. That is, before process P; starts executing, all its claim edges must already appear in the resource-allocation

graph. We can relax this condition by allowing a claim edge P; -+ R1 to be added to the graph only if all the edges associated with process P; are claim edges. Now suppose that process P; requests resource Rj. The request can be granted only if converting the request edge P; -+ Rj to an assignment edge R1 -+ P; does not result in the formation of a cycle in the resource-allocation graph. We check for safety by using a cycle-detection algorithm.

- An algorithm for detecting a cycle in this graph requires an order of n2 operations, where n is the number of processes in the system. If no cycle exists, then the allocation of the resource will leave the system in a safe state. If a cycle is found, then the allocation will put the system in an unsafe state. In that case, process P; will have to wait for its requests to be satisfied. To illustrate this algorithm, we consider the following resource-allocation graph.



- Suppose that P2 requests R2. Although R2 is currently free, we cannot allocate it to P2, since this action will create a cycle in the graph .A cycle, as mentioned, indicates that the system is in an unsafe state. If P1 requests R2, and P2 requests R1, then a deadlock will occur.

Banker's algorithm:

- The resource-allocation-graph algorithm is not applicable to a resource allocation system with multiple instances of each resource type. The Banker's algorithm is less efficient than the resource-allocation graph scheme. This algorithm is commonly known as the banker's algorithm. When a new process enters the system, it must declare the maximum

number of instances of each resource type that it may need. This nun1.ber may not exceed the total number of resources in the system.

- When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources. Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. We need the following data structures, where n is the number of processes in the system and m is the number of resource types:

Available: A vector of length m indicates the number of available resources of each type. If Available[j] equals k, then k instances of resource type Ri are available.

Max: An n x m matrix defines the maximum demand of each process. If Max[i] [j] equals k, then process P; may request at most k instances of resource type Ri.

Allocation: An 11 x m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] equals lc, then process P; is currently allocated lc instances of resource type Rj.

Need: An n x m matrix indicates the remaining resource need of each process. If Need[i][j] equals k, then process P; may need k more instances of resource type Ri to complete its task. Note that Need[i][j] equals Max[i][j] - Allocation [i][j].

These data structures vary over time in both size and value. To simplify the presentation of the banker's algorithm, we next establish some notation. Let X andY be vectors of length 11. We say that X::= Y if and only if X[i] ::= Y[i] for all i = 1, 2, ... , n. For example, if X = (1,7,3,2) and Y = (0,3,2,1), then Y ::=X. In addition, Y < X if Y ::=X and Y# X. We can treat each row in the matrices Allocation and Need as vectors and refer to them as Allocation; and Need;. The vector Allocation; specifies the resources currently allocated to process P;; the vector Need; specifies the additional resources that process P; may still request to complete its task.

Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initialize Work= Available and Finish[i] =false for i = 0, 1, ... , n - 1.

2. Find an index i such that both
    a. Finish[i] ==false
    b. $Need_i$ Work. If no such i exists, go to step 4.
3. Work = Work + Allocation; Finish[i] = true. Go to step 2.
4. If Finish[i] ==true for all i, then the system is in a safe state.

Resource-Request Algorithm

Let Request; be the request vector for process P;. If Request; [j] == k, then process P; wants k instances of resource type Rj. When a request for resources is made by process P;, the following actions are taken:

1. If Request; <= Need; go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If Request< = Available, go to step 3. Otherwise, P; must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process P; by modifyil1.g the state as follows: Available=

    Available- $Request_i$

Deadlock

Allocation; =Allocation; $+Request_i$

Need; $=Need_i - Request_i$

If the resulting resource-allocation state is safe, the transaction is completed, and process P; is allocated its resources. However, if the new state is unsafe, then P; must wait for $Request_i$, and the old resource-allocation state is restored.

Deadlock Detection

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system may provide:

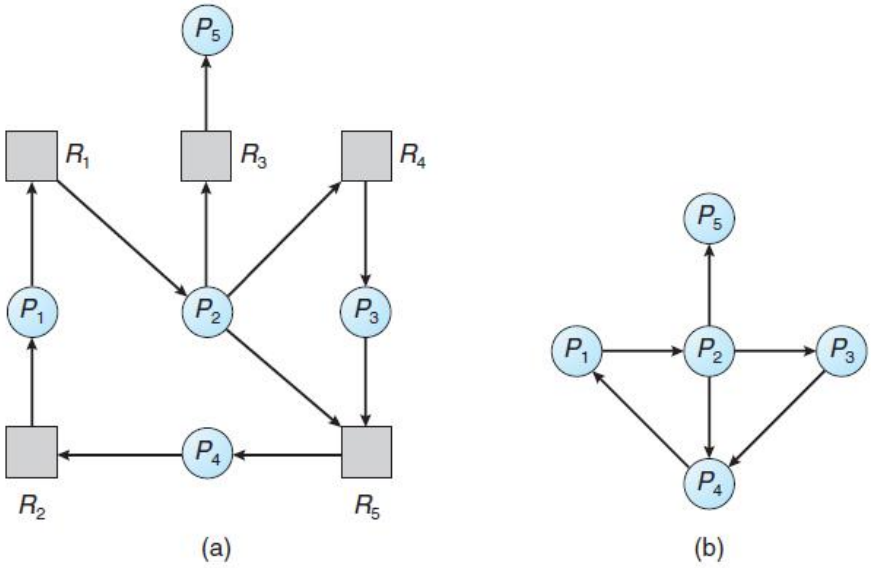
• An algorithm that examines the state of the system to determine whether a deadlock has occurred

• An algorithm to recover from the deadlock

Single Instance of Each Resource Type

If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph. We obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.

More precisely, an edge from Pi to Pj in a wait-for graph implies that process Pi is waiting for process Pj to release a resource that Pi needs.



An edge Pi → Pj exists in a wait-for graph if and only if the corresponding resource allocation graph contains two edges Pi → Rq and Rq → Pj for some resource Rq . In Figure, we present a resource-allocation graph and the corresponding wait-for graph. As before, a deadlock exists in the system if and only if the wait-for graph contains a cycle.

To detect deadlocks, the system needs to maintain the wait for graph and periodically invoke an algorithm that searches for a cycle in the graph. An algorithm to detect a cycle in a graph requires an order of n2 operations, where n is the number of vertices in the graph.

Several Instances of a Resource Type

The wait-for graph scheme is not applicable to a resource-allocation system with multiple instances of each resource type. We turn now to a deadlock detection algorithm that is applicable

to such a system. The algorithm employs several time-varying data structures that are similar to those used in the banker's algorithm

• Available. A vector of length m indicates the number of available resources of each type.

• Allocation. An n × m matrix defines the number of resources of each type currently allocated to each process.

• Request. An n × m matrix indicates the current request of each process. If Request[i][j] equals k, then process Pi is requesting k more instances of resource type Rj . To simplify notation, we again treat the rows in the matrices Allocation and Request as vectors; we refer to them as Allocationi and Requesti . The detection algorithm described here simply investigates every possible allocation sequence for the processes that remain to be completed.

1. Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available.

For i = 0, 1, ..., n–1, if Allocationi _= 0, then Finish[i] = false. Otherwise, Finish[i] = true.

2. Find an index i such that both

a. Finish[i] == false

b. Requesti ≤Work

3. Work =Work +

Allocationi Finish[i] = true

Go to step 2.

4. If Finish[i] ==false for some i, 0≤i<n, then the system is in a deadlocked state. Moreover, if Finish[i] == false, then process Pi is deadlocked. This algorithm requires an order of m × n2 operations to detect whether the system is in a deadlocked state. You may wonder why we reclaim the resources of process Pi (in step 3) as soon as we determine that Requesti ≤ Work (in step 2b). We know that Pi is currently not involved in a deadlock (since Requesti ≤ Work). Thus, we take an optimistic attitude and assume that Pi will require no more resources to complete its task; it will thus soon return all currently allocated resources to the system. If our assumption is incorrect, a deadlock may occur later. That deadlock will be detected the next time the deadlock-detection algorithm is invoked.

To illustrate this algorithm, we consider a system with five processes P0 through P4 and three resource types A, B, and C. Resource type A has seven instances, resource type B has two instances, and resource type C has six instances. Suppose that, at time T0, we have the following resource-allocation state:

	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

We claim that the system is not in a deadlocked state. Indeed, if we execute our algorithm, we will find that the sequence <P0, P2, P3, P1, P4> results in Finish[i] == true for all i. Suppose now that process P2 makes one additional request for an instance of type C. The Request matrix is modified as follows:

	Request
	A B C
P0	0 0 0
P1	2 0 2
P2	0 0 1
P3	1 0 0
P4	0 0 2

We claim that the system is now deadlocked. Although we can reclaim the resources held by process P0, the number of available resources is not sufficient to fulfill the requests of the other processes. Thus, a deadlock exists, consisting of processes P1, P2, P3, and P4.

Recovery from Deadlock

When a detection algorithm determines that a deadlock exists, several alternatives are available. One possibility is to inform the operator that a deadlock has occurred and to let the

operator deal with the deadlock manually. Another possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock.

Process Termination

To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.

• Abort all deadlocked processes. This method clearly will break the deadlock cycle, but at great expense. The deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.

• Abort one process at a time until the deadlock cycle is eliminated. This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

Aborting a process may not be easy. If the process was in the midst of updating a file, terminating it will leave that file in an incorrect state. Similarly, if the process was in the midst of printing data on a printer, the system must reset the printer to a correct state before printing the next job.

Resource Preemption

To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. If preemption is required to deal with deadlocks, then three issues need to be addressed:

1. Selecting a victim. Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed.

2. Rollback. If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must roll back the process to some safe state and restart it from that state. Since, in general, it is difficult to determine what a safe state is, the simplest solution is a total rollback: abort the process and then restart it. Although it is more effective to roll back the process only as far as

necessary to break the deadlock, this method requires the system to keep more information about the state of all running processes.

3. Starvation. How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process?

In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task, a starvation situation any practical system must address. Clearly, we must ensure that a process can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

POSSIBLE QUESTIONS

UNIT – II

PART – A (20 MARKS)

(Q.NO 1 TO 20 Online Examinations)

PART – B (2 MARKS)

1. Define System Call
2. What is meant by Kernel?
3. Write a short notes on System Program
4. Define Semaphore
5. List the types of Scheduling
6. Define Deadlock
7. List some Threading issues

PART – C (6 MARKS)

1. Discuss in detail about System Calls and System Programs.
2. Explain about FCFS Scheduling Algorithm with an example
3. Explain about Deadlock and its process
4. Explain about process scheduling
5. Discuss in detail about System view of the process and resources
6. Explain about Thread and threading issues
7. Discuss in detail about Preemptive and Non preemptive Scheduling Algorithms with suitable examples
8. Explain the methods of inter-process communication
9. Explain the Round Robin Scheduling Algorithm with an example
10. Explain the Shortest Job First (SJF) Scheduling Algorithm with an example

KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore – 641 021.

(For the Candidates admitted from 2018 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

UNIT - II : (Objective Type Multiple choice Questions each Question carries one Mark)

OPERATING SYSTEMS

PART - A (Online Examination)

Questions	Opt1	opt2	opt3	opt4	KEY
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
Four necessary conditions for deadlock are non pre-emption, circular wait, hold and wait and	mutual exclusion	race condition	buffer overflow	multiprocessing	mutual exclusion
A series of statements explaining how the data is to be processed is called	instruction	compiler	program	interpretor	program
Banker's algorithm deals with	deadlock prevention	deadlock avoidance	deadlock recovery	mutual exclusion	deadlock avoidance
Which is non pre-emptive	Round robin	FIFO	MQS	MQSF	FIFO
A hardware device which is capable of executing a sequence of instructions, is known as	CPU	ALU	CU	Processor	Processor
Distributed systems should	high security	have better resource sharing	better system utilization	low system overhead	have better resource sharing
Which of the following is always there in a computer	Batch system	Operating system	Time sharing system	Controlling system	Operating system

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
Banker's algorithm for resource allocation deals with	deadlock prevention	deadlock aviodance	deadlock recovery	circular wait	deadlock aviodance
_____ is the basis of multiprogrammed operating system.	RR scheduling	Self Scheduling	CPU sceduling	throughput	CPU sceduling
A _____ is executed until it must wait, typically for the completion of some i/o request	reverse	deadlock avoidance	deadlock	process	process
_____ is a fundamental operating system function.	RR	CPU	Scheduling	nonpreemptive	Scheduling
Process execution begins with a_____	CPU burst	RR scheduling	SJF scheduling	SRT scheduling	CPU burst
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
When scheduling takes place only under circumstances 1 and 4 called _____	variable class	real time class	priority class	nonpreemptive	nonpreemptive
Another component involved in the CPU scheduling function is the _____	central edge	dispatcher	claim edge	graph edge	dispatcher
One measure of work is the number of processes completed per time unit called _____	throughput	variable class	real time class	priority class	throughput
Which of the following is the simplest scheduling discipline?	FCFS scheduling	RR scheduling	SJF scheduling	SRT scheduling	FCFS scheduling
In which scheduling, processes are dispatched according to their arrival time on the ready queue?	FCFS scheduling	RR scheduling	SJF scheduling	SRT scheduling	FCFS scheduling
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
Which scheduling is effective in time sharing environments	FIFO scheduling	RR scheduling	SJF scheduling	SRT scheduling	RR scheduling

Variable size blocks are called	Pages	Segments	Tables	None	Segments
Which scheduling is effective in time sharing environments	FIFO scheduling	RR scheduling	SJF scheduling	SRT scheduling	RR scheduling
Which of the following is non-preemptive scheduling?	RR scheduling	SJF scheduling	SRT scheduling	None	SJF scheduling
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
The simplest CPU sceduling algorithm is the	FCS	SJS	FCFS	DFG	FCFS
The SJF algorithm is a special case of the general _____ algorithm	FCS	SJS	Roundrobin	FCSC	Roundrobin
_____ scheduling algorithm is designed especially for time sharing systems.	CFS	FSCS	priority	Round Robin	Round Robin
The seek optimization strategy in which there is no reordering of the queue is called _____.	FCFS	SSTF	SCAN	C-SCAN	FCFS
A major problem with priority scheduling algorithms is _____	tail	Starvation	time first	time quantum	Starvation
If the time quantum is very small the RR aproach is called _____	Queues	Processor Sharing	Sharing resources	Context switching	Processor Sharing
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
If several identical processors are available then _____ can occur.	heterogeneous	homogeneous	load sharing	UMA	load sharing
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
_____ 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
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

_____ uses the the given algorithm and the system workload to produce a formula.	deterministic modelling	scheduling process	Analaytic evaluation	Queuing model	Analaytic evaluation
If no thread is found the dispatcher will execute a special thread called_____	variable class	real time class	priority class	idle thread	idle thread
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
_____ 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
_____ 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
A system is in a safe state only if there exists a _____	Safe state	unsafe state	normal	deadlock	Safe state
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
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
CPU Scheduling is the basis of _____ operating system.	single programmed	multi programmed	multi system	multi disks	multi programmed
Scheduling is a fundamental _____ function.	computer	operating system	system resource	disk	operating system
Another component involved in the CPU _____ function is the dispatcher	processing	mathematical	arithmetic	scheduling	scheduling
A major problem for priority _____ is starvation.	sort algorithms	scheduling algoriuthms	search algorithms	manage algorithms	scheduling algoriuthms

The seek _____ strategy in which there is no reordering of the queue is called SSTF	processing	scheduling	optimization	implementation	optimization
The high _____ would be waiting for a lower priority one to finish is called priority inversion	performance	priority	patent	graph edge	priority
A _____ is a program segment where shared resources are accessed.	critical section	sub section	cross section	class section	critical section
If no thread is found, the _____ will execute a special thread called idle thread.	degrader	scheduler	dispatcher	redeemer	dispatcher
_____ execution begins with a CPU Burst.	Process	Performance	Purge	Put	Process
SJF Scheduling is an example for _____ scheduling.	non- preemptive	preemptive	emptive	prescheduling	non-preemptive
_____ is the simplest CPU sceduling algorithm.	FCFS	LCFS	FCLS	LCFS	FCFS
Segments are called _____ blocks.	equal size	variable class	variable size	big size	variable size
_____ can be described more precisely in terms of a directed graph.	semaphore	deadlocks	dumplocks	starvation	deadlocks
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

UNIT – III

SYLLABUS

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

MEMORY MANAGEMENT

- The operating system, executing in kernel mode, is given unrestricted access to both operating-system memory and users' memory. This provision allows the operating system to load users' programs into users' memory, to dump out those programs in case of errors, to access and modify parameters of system calls, to perform I/O to and from user memory, and to provide many other services.
- Consider, for example, that an operating system for a multiprocessing system must execute context switches, storing the state of one process from the registers into main memory before loading the next process's context from main memory into the registers.

  This scheme allows the operating system to change the value of the registers but prevents user programs from changing the registers' contents.

Address Binding

- Memory management is the functionality of an operating system which handles or manages primary memory. Memory management keeps track of each and every memory location 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 management provides protection by using two registers, a base register and a limit register. The base register holds the smallest legal physical memory address and the limit register specifies the size of the range. For example, if the base register holds 300000 and the limit register is 1209000, then the program can legally access all addresses from 300000 through 411999.

Instructions and data to memory addresses can be done in following ways

- Compile time -- When it is known at compile time where the process will reside, compile time binding is used to generate the absolute code.
- Load time -- When it is not known at compile time where the process will reside in memory, then the compiler generates re-locatable code.
- Execution time -- If the process can be moved during its execution from one memory segment to another, then binding must be delayed to be done at run time

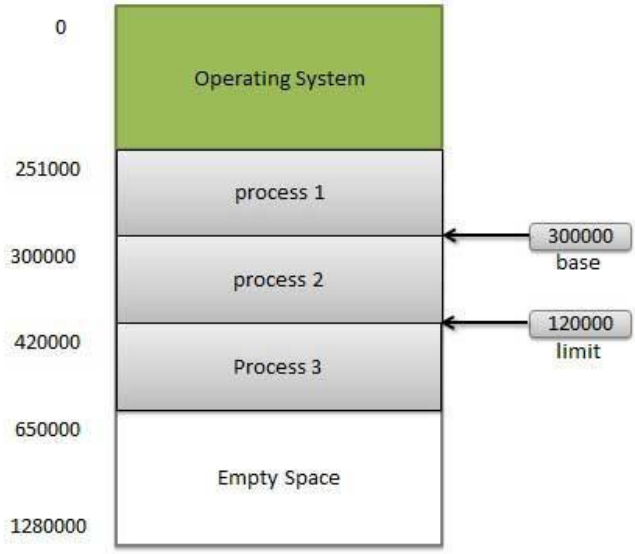
Dynamic Loading

- In dynamic loading, a routine of a program is not loaded until it is called by the program. All routines are kept on disk in a re-locatable load format. The main program is loaded into memory and is executed. Other routines methods or modules are loaded on request. Dynamic loading makes better memory space utilization and unused routines are never loaded.
- The advantage of dynamic loading is that a routine is loaded only when it is needed. This method is particularly useful when large amounts of code are needed to handle infrequently occurring cases, such as error routines. In this case, although the total program size may be large, the portion that is used (and hence loaded) may be much smaller.
- Dynamic loading does not require special support from the operating system. It is the responsibility of the users to design their programs to take advantage of such a method. Operating systems may help the programmer, however, by providing library routines to implement dynamic loading.

Dynamic Linking

- Linking is the process of collecting and combining various modules of code and data into a executable file that can be loaded into memory and executed. Operating system can link system level libraries to a program. When it combines the libraries at load time, the linking is called static linking and when this linking is done at the time of execution, it is called as dynamic linking.

- In static linking, libraries linked at compile time, so program code size becomes bigger whereas in dynamic linking libraries linked at execution time so program code size remains smaller.
- Unlike dynamic loading, dynamic linking and shared libraries generally require help from the operating system. If the processes in memory are protected from one another, then the operating system is the only entity that can check to see whether the needed routine is in another process's memory space or that can allow multiple processes to access the same memory addresses.



PHYSICAL AND VIRTUAL ADDRESS SPACE

Logical (Virtual) 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.

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.

MEMORY ALLOCATION STRATEGIES

Contiguous Memory Allocation

- The main memory must accommodate both the operating system and the various user processes. We therefore need to allocate main memory in the most efficient way possible. The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.
- We can place the operating system in either low memory or high memory. The major factor affecting this decision is the location of the interrupt vector. Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well.
- We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In contiguous memory allocation, each process is contained in a single section of memory that is contiguous to the section containing the next process.

Memory Protection

- Before discussing memory allocation further, we must discuss the issue of memory protection. If we have a system with a relocation register, together with a limit, we accomplish our goal. The relocation register contains the value of the smallest physical

address; the limit register contains the range of logical addresses (for example, relocation = 100040 and limit = 74600).

- Each logical address must fall within the range specified by the limit register. The MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory. When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch. Because every address generated by a CPU is checked against these registers, we can protect both the operating system and the other users' programs and data from being modified by this running process.

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 multiple **partition 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. This method was originally used by the IBM OS/360 operating system (called MFT) but is no longer in use. The method described next is a generalization of the fixed-partition scheme (called MVT); it is used primarily in batch environments. Many of the ideas presented here are also applicable to a time-sharing environment in which pure segmentation is used for memory management.
- 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.
- As processes enter the system, they are put into an input queue. The operating system takes into account the memory requirements of each process and the amount of available memory space in determining which processes are allocated memory.

- When a process is allocated space, it is loaded into memory, and it can then compete for CPU time. When a process terminates, it releases its memory, which the operating system may then fill with another process from the input queue.
- In general, the memory blocks available comprise a set of holes of various sizes scattered throughout memory. When a process arrives and needs memory, the system searches the set for a hole that is large enough for this process. If the hole is too large, it is split into two parts. One part is allocated to the arriving process; the other is returned to the set of holes. When a process terminates, it releases its block of memory, which is then placed back in the set of holes.
- If the new hole is adjacent to other holes, these adjacent holes are merged to form one larger hole. At this point, the system may need to check whether there are processes waiting for memory and whether this newly freed and recombined memory could satisfy the demands of any of these waiting processes. This procedure is a particular instance of the general dynamic storage allocation 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.

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

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

• **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-fit approach.

Fragmentation

- Both the first-fit and best-fit strategies for memory allocation suffer from **external fragmentation.** As processes are loaded and removed from memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory

space to satisfy a request but the available spaces are not contiguous: storage is fragmented into a large number of small holes. This fragmentation problem can be severe. In the worst case, we could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, we might be able to run several more processes.

- Whether we are using the first-fit or best-fit strategy can affect the amount of fragmentation. (First fit is better for some systems, whereas best fit is better for others.) Another factor is which end of a free block is allocated. (Which is the leftover piece—the one on the top or the one on the bottom?) Memory fragmentation can be internal as well as external. Consider a multiple-partition allocation scheme with a hole of 18,464 bytes. Suppose that the next process requests 18,462 bytes. If we allocate exactly the requested block, we are left with a hole of 2 bytes. The overhead to keep track of this hole will be substantially larger than the hole itself.
- The general approach to avoiding this problem is to break the physical memory into fixed-sized blocks and allocate memory in units based on block size. With this approach, the memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is **internal fragmentation**—unused memory that is internal to a partition.
- One solution to the problem of external fragmentation is compaction. The goal is to shuffle the memory contents so as to place all free memory together in one large block. Compaction is not always possible, however. If relocation is static and is done at assembly or load time, compaction cannot be done. It is possible only if relocation is dynamic and is done at execution time. Another possible solution to the external-fragmentation problem is to permit the logical address space of the processes to be noncontiguous, thus allowing a process to be allocated physical memory wherever such memory is available. Two complementary techniques achieve this solution: segmentation and paging

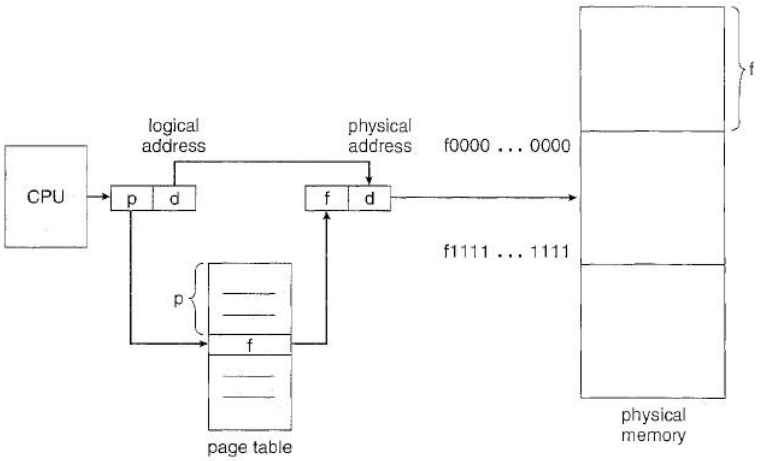
PAGING

- It is a memory-management scheme that permits the physical address space a process to be noncontiguous. Paging avoids external fragmentation and the need for compaction. It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store; most memory management schemes used before the introduction of paging suffered from this problem. The problem arises because, when some code fragments or data residing in main memory need to be swapped out, space must be framed on the backing store.
- The backing store has the same fragmentation problems discussed in connection with main memory, but access is much slower, so compaction is impossible. Because of its advantages over earlier methods, paging in its various forms is used in most operating systems.

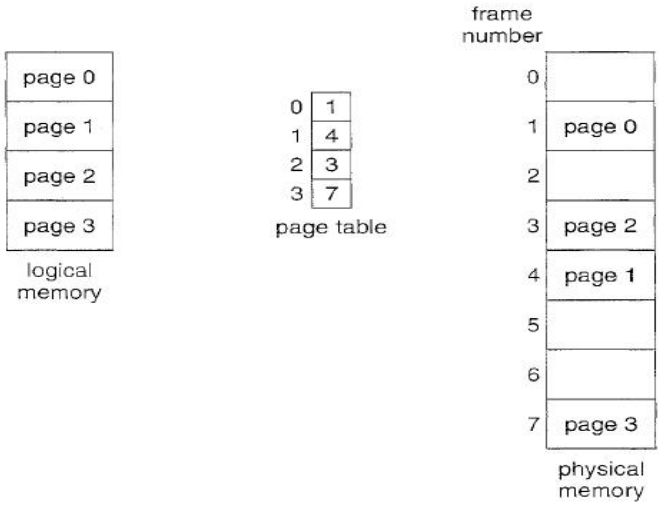
  Traditionally, support for paging has been handled by hardware. However, recent designs have implemented paging by closely integrating the hardware and operating system, especially on 64-bit microprocessors.

Basic Method

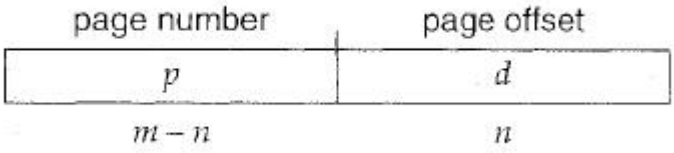
- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages.
- When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store). The backing store is divided into fixed-sized blocks that are of the san1.e size as the memory frames. The hardware support for paging is illustrated in the following figure



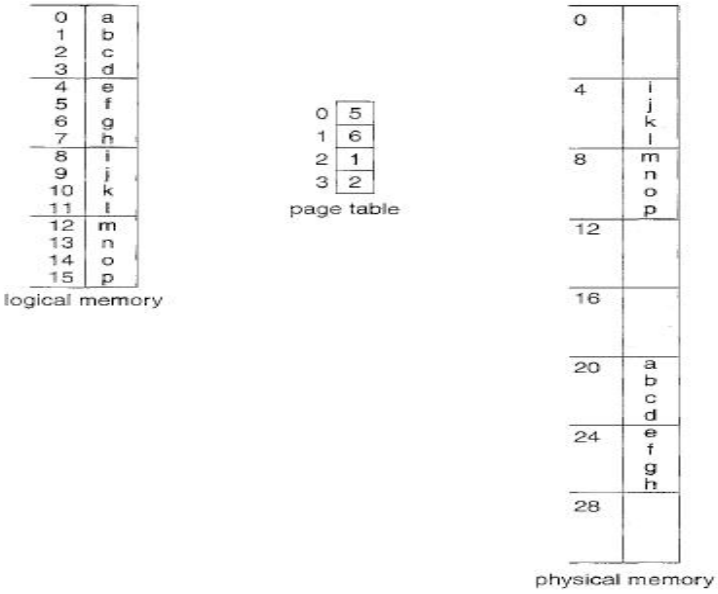
- Every address generated the CPU is divided into two parts: a {p) and a . The page number is used as an index into a page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit. The paging model of memory is shown in the following diagram



- The page size (like the frame size) is defined by the hardware. The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.



- If the size of the logical address space is 2m, and a page size is 271 addressing units (bytes or wordst then the high-order m- n bits of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows: where p is an index into the page table and d is the displacement within the page. As a concrete (although minuscule) example, consider the memory in the following diagram



- Here, in the logical address, n= 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages), we show how the user's view of memory can be mapped into physical memory. Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 [= (5 x 4) + 0]. Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 x 4) + 3].
- Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 [= (6 x 4) + O]. Logical address 13 maps to physical address 9. You may have noticed that paging itself is a form of dynamic relocation. Every logical address is bound by the paging hardware to some physical address.

Using paging is similar to using a table of base (or relocation) registers, one for each frame of memory. When we use a paging scheme, we have no external fragmentation: any free frame can be allocated to a process that needs it. However, we may have some internal fragmentation. Notice that frames are allocated as units.

- If the memory requirements of a process do not happen to coincide with page boundaries, the last frame allocated may not be completely full. For example, if page size is 2,048 bytes, a process of 72,766 bytes will need 35 pages plus 1,086 bytes. It will be allocated 36 frames, resulting in internal fragmentation of 2,048 - 1,086 = 962 bytes. In the worst case, a process would need 11 pages plus 1 byte. It would be allocated 11 + 1 frames, resulting in internal fragmentation of almost an entire frame. If process size is independent of page size, we expect internal fragmentation to average one-half page per process. This consideration suggests that small page sizes are desirable. Generally, page sizes have grown over time as processes, data sets, and main memory have become larger.
- Today, pages typically are between 4 KB and 8 KB in size and some systems support even larger page sizes. Some CPUs and kernels even support multiple page sizes. For instance, Solaris uses page sizes of 8 KB and 4 MB, depending on the data stored by the pages. Researchers are now developing support for variable on-the-fly page size. Usually, each page-table entry is 4 bytes long, but that size can vary as well. A 32-bit entry can point to one of 232 physical page frames. If frame size is 4 KB, then a system with 4-byte entries can address 244 bytes (or 16 TB) of physical memory. When a process arrives in the system to be executed, its size, expressed in pages, is examined. Each page of the process needs one frame. Thus, if the process requires 11 pages, at least 11 frames must be available in memory.
- If n frames are available, they are allocated to this arriving process. The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process. The next page is loaded into another frame, its frame number is put into the page table, and so on. An important aspect of paging is the clear separation between the user's view of memory and the actual physical memory. The user program views

memory as one single space, containing only this one program. In fact, the user program is scattered throughout physical memory, which also holds other programs.

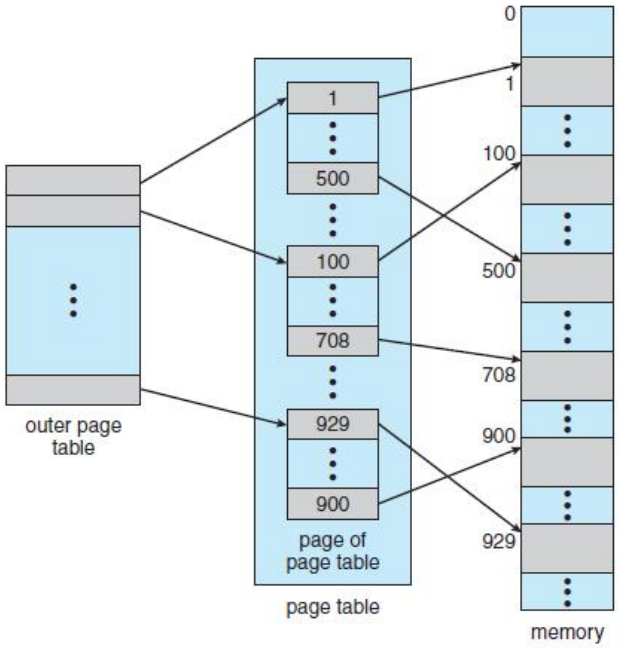
- The difference between the user's view of memory and the actual physical memory is reconciled by the address-translation hardware. The logical addresses are translated into physical addresses. This mapping is hidden from the user and is controlled by the operating system. Notice that the user process by definition is unable to access memory it does not own.
- It has no way of addressing memory outside of its page table, and the table includes only those pages that the process owns. Since the operating system is managing physical memory, it must be aware of the allocation details of physical memory-which frames are allocated, which frames are available, how many total frames there are, and so on. This information is generally kept in a data structure called a frame the frame-table has one entry for each physical page frame, indicating whether the latter is free or allocated and, if it is allocated, to which page of which process or processes.
- In addition, the operating system must be aware that user processes operate in user space, and all logical addresses must be mapped to produce physical addresses.If a user makes a system call (to do I/0, for example) and provides an address as a parameter (a buffe1~ for instance), that address must be mapped to produce the correct physical address.
- The operating system maintains a copy of the page table for each process, just as it maintains a copy of the instruction counter and register contents. This copy is used to translate logical addresses to physical addresses whenever the operating system must map a logical address to a physical address manually. It is also used by the CPU dispatcher to define the hardware page table when a process is to be allocated the CPU. Paging therefore increases the context-switch time.

STRUCTURE OF PAGE TABLE

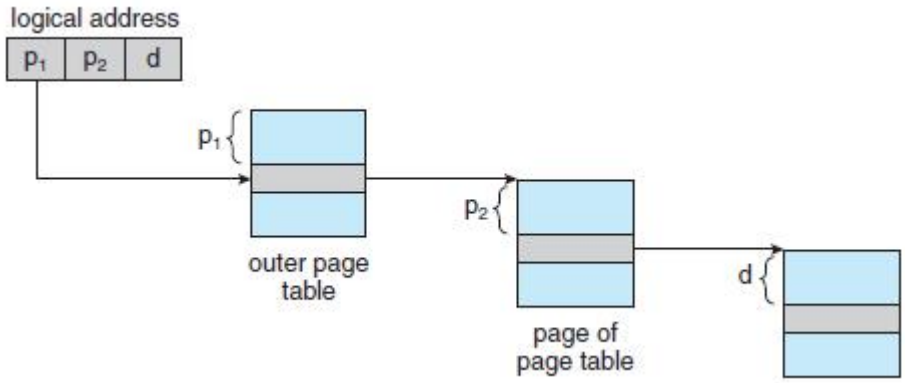
In this section, we explore some of the most common techniques for structuring the page table, including hierarchical paging, hashed page tables, and inverted page tables.

Hierarchical Paging

- Most modern computer systems support a large logical address space (232 to 264). In such an environment, the page table itself becomes excessively large. For example, consider a system with a 32-bit logical address space. If the page size in such a system is 4 KB (212), then a page table may consist of up to 1 million entries (232/212). Assuming that each entry consists of 4 bytes, each process may need up to 4 MB of physical address space for the page table alone. Clearly, we would not want to allocate the page table contiguously in main memory. One simple solution to this problem is to divide the page table into smaller pieces.



- We can accomplish this division in several ways. One way is to use a two-level paging algorithm, in which the page table itself is also paged. For example, consider again the system with a 32-bit logical address space and a page size of 4 KB. A logical address is divided into a page number

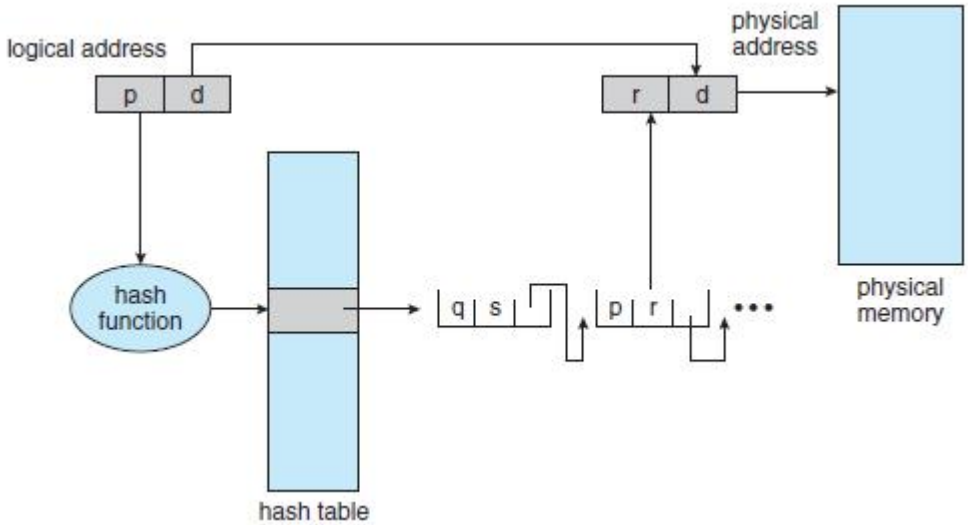


consisting of 20 bits and a page offset consisting of 12 bits. Because we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset. Thus, a logical address is as follows:

Where p1 is an index into the outer page table and p2 is the displacement within the page of the inner page table. The address-translation method for this architecture is shown in Figure. Because address translation works from the outer page table inward, this scheme is also known as a forward-mapped page table.

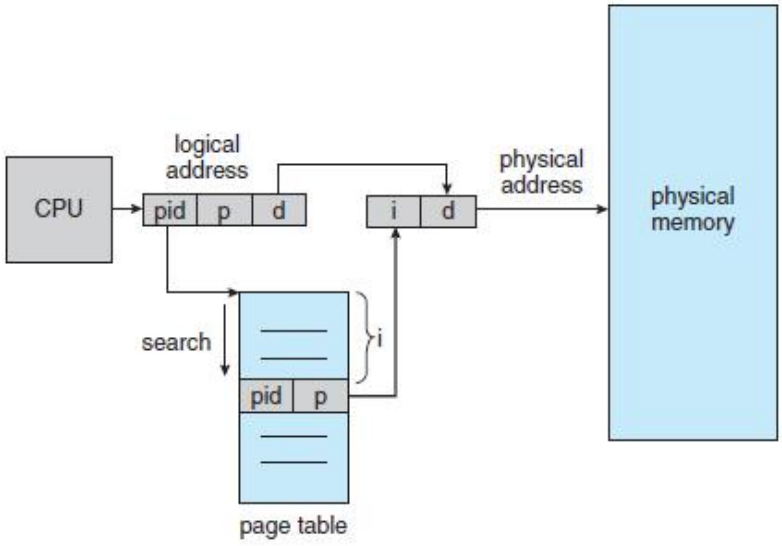
Hashed Page Tables

- A common approach for handling address spaces larger than 32 bits is to use a hashed page table, with the hash value being the virtual page number. Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions). Each element consists of three fields: (1) the virtual page number, (2) the value of the mapped page frame, and (3) a pointer to the next element in the linked list. The algorithm works as follows:
- The virtual page number in the virtual address is hashed into the hash table. The virtual page number is compared with field 1 in the first element in the linked list. If there is a match, the corresponding page frame (field 2) is used to form the desired physical address. If there is no match, subsequent entries in the linked list are searched for a matching virtual page number.



Inverted Page Tables

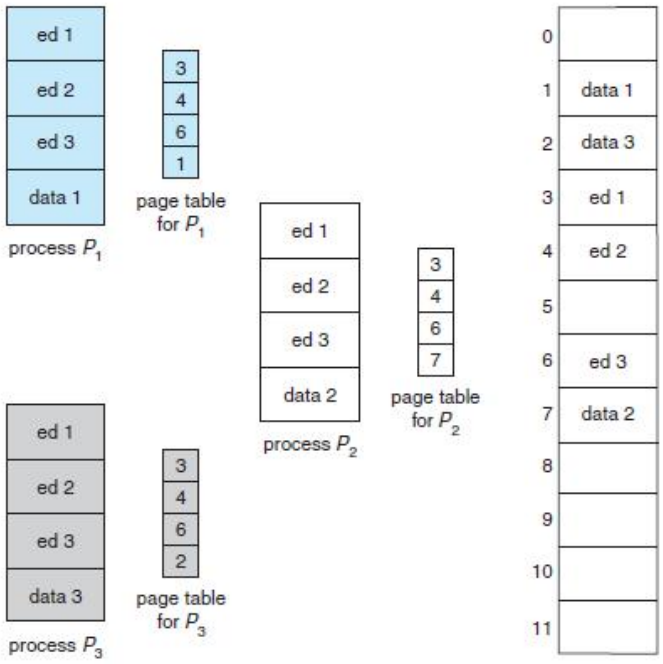
- Usually, each process has an associated page table. The page table has one entry for each page that the process is using (or one slot for each virtual address, regardless of the latter's validity). This table representation is a natural one, since processes reference pages through the pages' virtual addresses.
- The operating system must then translate this reference into a physical memory address. Since the table is sorted by virtual address, the operating system is able to calculate where in the table the associated physical address entry is located and to use that value directly. One of the drawbacks of this method is that each page table may consist of millions of entries. These tables may consume large amounts of physical memory just to keep track of how other physical memory is being used.



- To solve this problem, we can use an inverted page table. An inverted page table has one entry for each real page (or frame) of memory. Each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns the page. Thus, only one page table is in the system, and it has only one entry for each page of physical memory.

Shared Pages

- An advantage of paging is the possibility of sharing common code. This consideration is particularly important in a time-sharing environment. Consider a system that supports 40 users, each of whom executes a text editor. If the text editor consists of 150 KB of code and 50 KB of data space, we need 8,000 KB to support the 40 users. If the code is reentrant code (or pure code), it can be shared, as shown in Figure. Here, we see three processes sharing a three-page editor—each page 50 KB in size (the large page size is used to simplify the figure). Each process has its own data page. Reentrant code is non-self-modifying code: it never changes during execution. Thus, two or more processes can execute the same code at the same time.



- Each process has its own copy of registers and data storage to hold the data for the process's execution. The data for two different processes will, of course, be different. Only one copy of the editor need be kept in physical memory. Each user's page table maps onto the same physical copy of the editor, but data pages are mapped onto different frames. Thus, to support 40 users, we need only one copy of the editor (150 KB), plus 40 copies of the 50 KB of data space per user. The total space required is now 2,150 KB instead of 8,000 KB—a significant savings. Other heavily used programs can also be shared—compilers, window systems, run-time libraries, database systems, and so on. To be sharable, the code must be reentrant. The read-only nature of shared code should not be left to the correctness of the code; the operating system should enforce this property.
- The sharing of memory among processes on a system is similar to the sharing of the address space of a task by threads.

SEGMENTATION

- An important aspect of memory management that became unavoidable with paging is the separation of the user's view of memory from the actual physical memory. As we have already seen, the user's view of memory is not the same as the actual physical memory. The user's view is mapped onto physical memory. This mapping allows differentiation between logical memory and physical memory.

Basic Methods

- It is a memory-management scheme that supports this user view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: a segment name and an offset. (Contrast this scheme with the paging scheme, in which the user specifies only a single address, which is partitioned by the hardware into a page number and an offset, all invisible to the programmer.) For simplicity of implementation, segments are numbered and are referred to by a segn"lent number, rather than by a segment name. Thus, a logical address consists of a two tuple:

  <segment-number, offset>.

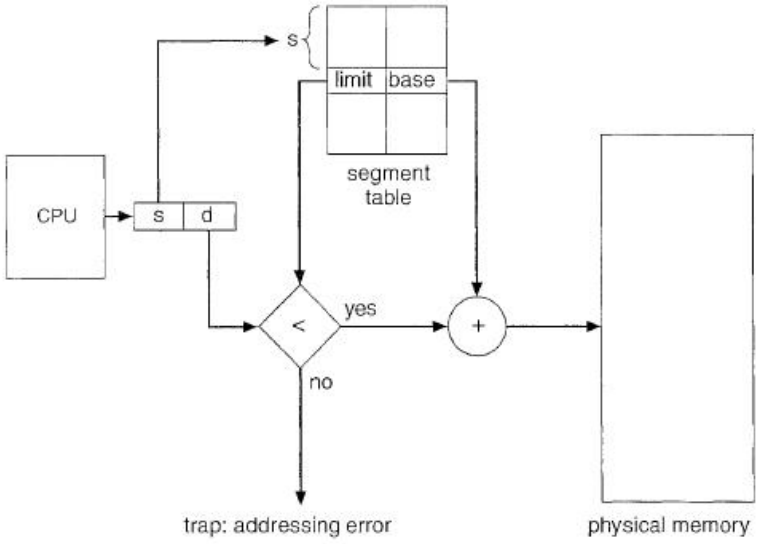
- Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program. A C compiler might create separate segments for the following:

  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

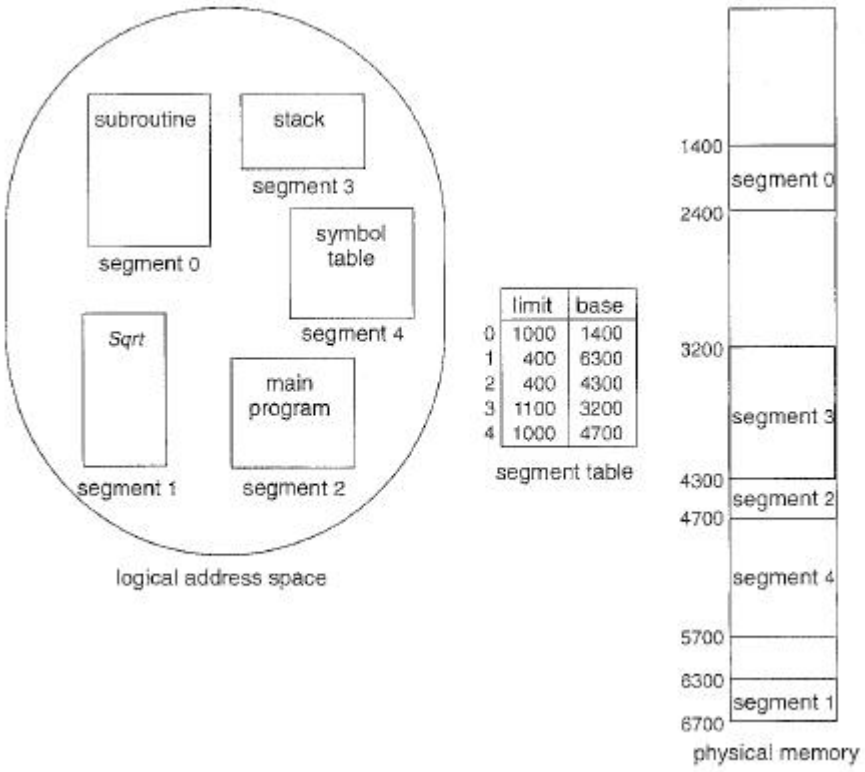
- Libraries that are linked in during compile time might be assigned separate segments. The loader would take all these segments and assign them segment numbers.

Hardware

- Although the user can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, of course, a one-dimensional sequence of bytes. Thus, we must define an implementation to map two dimensional user-defined addresses into one-dimensional physical addresses. This mapping is affected by each entry in the segment table has a segment base and a segment limit. The segment base contains the start physical address where the segment resides in memory, and the segment limit specifies the length of the segment. The use of a segment table is illustrated in Figure



- A logical address consists of two parts: a segment number, s, and an offset into that segment, d. the segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment). When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs. As an example, consider the situation shown in Figure



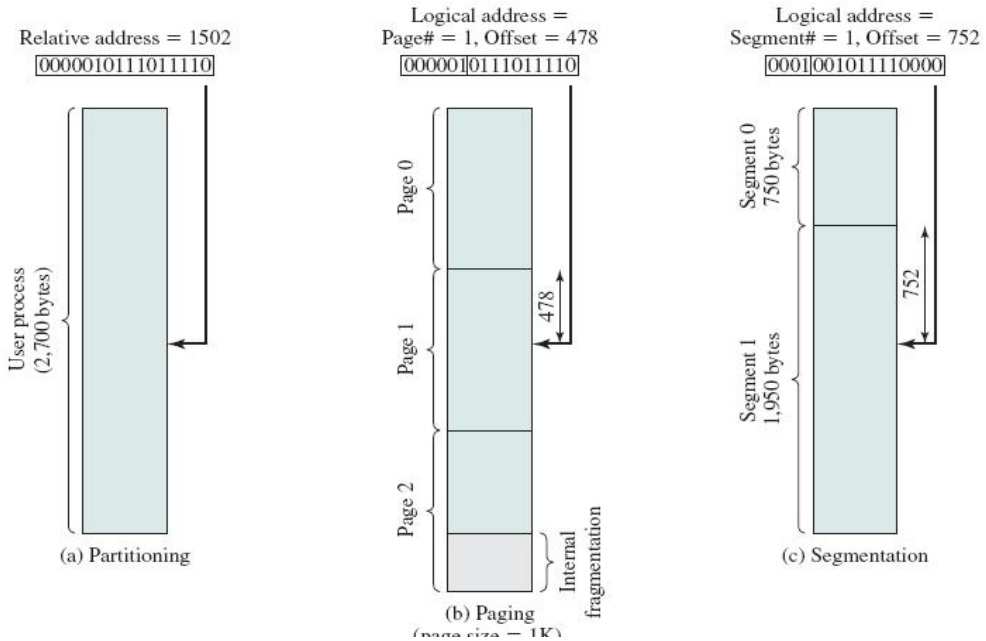
- We have five segments numbered from 0 through 4. The segments are stored in physical memory as shown. The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit). For example, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 +53= 4353. A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1000 bytes long.

Segmentation and Paging

- A user program can be subdivided using segmentation, in which the program and its associated data are divided into a number of **segments.** It is not required that all segments of all programs be of the same length, although there is a maximum segment length. As with paging, a logical address using segmentation consists of two parts, in this case a

segment number and an offset. Because of the use of unequal-size segments, segmentation is similar to dynamic partitioning.

- In the absence of an overlay scheme or the use of virtual memory, it would be required that all of a program's segments be loaded into memory for execution. The difference, compared to dynamic partitioning, is that with segmentation a program may occupy more than one partition, and these partitions need not be contiguous. Segmentation eliminates internal fragmentation but, like dynamic partitioning, it suffers from external fragmentation.
- However, because a process is broken up into a number of smaller pieces, the external fragmentation should be less. Whereas paging is invisible to the programmer, segmentation is usually visible and is provided as a convenience for organizing programs and data. STypically, the programmer or compiler will assign programs and data to different segments. For purposes of modular programming, the program or data may be further broken down into multiple segments.
- The principal inconvenience of this service is that the programmer must be aware of the maximum segment size limitation. Another consequence of unequal-size segments is that there is no simple relationship between logical addresses and physical addresses.
- Each segment table entry would have to give the starting address in main memory of the corresponding segment. The entry should also provide the length of the segment, to assure that invalid addresses are not used. When a process enters the Running state, the address of its segment table is loaded into a special register used by the memory management hardware. Consider an address of n_m bits, where the leftmost n bits are the segment number and the rightmost m bits are the offset. In our example (Figure C), n _ 4 and m _ 12. Thus the maximum segment size is 2 12 _ 4096.



The following steps are needed for address translation:

• Extract the segment number as the leftmost n bits of the logical address.

• Use the segment number as an index into the process segment table to find the starting physical address of the segment.

• Compare the offset, expressed in the rightmost m bits, to the length of the segment.

If the offset is greater than or equal to the length, the address is invalid. The desired physical address is the sum of the starting physical address of the segment plus the offset.

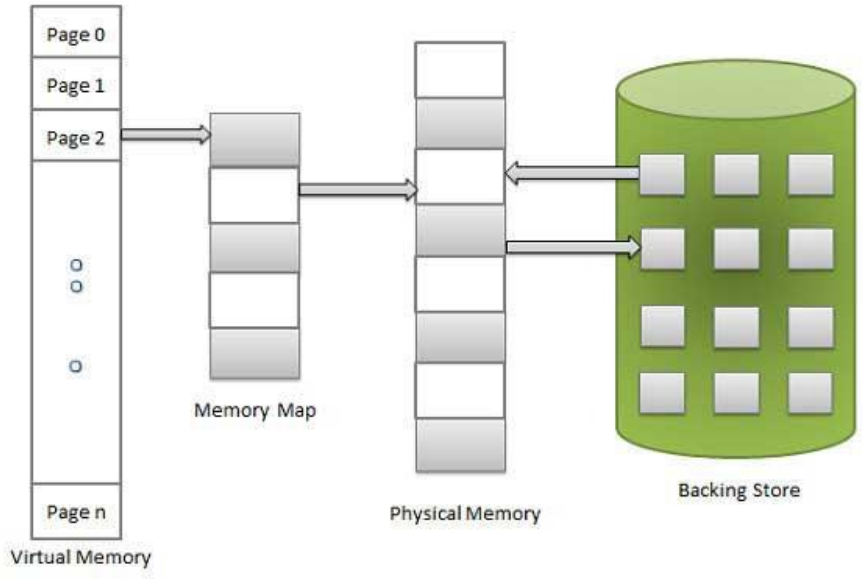
In our example, we have the logical address 0001001011110000, which is segment number 1, offset 752. Suppose that this segment is residing in main memory starting at physical address 0010000000100000. Then the physical address is 0010000000100000 + 001011110000 _ 0010001100010000.

To summarize, with simple segmentation, a process is divided into a number of segments that need not be of equal size. When a process is brought in, all of its segments are loaded into available regions of memory, and a segment table is set up.

VIRTUAL MEMORY

Virtual Memory and its Organization

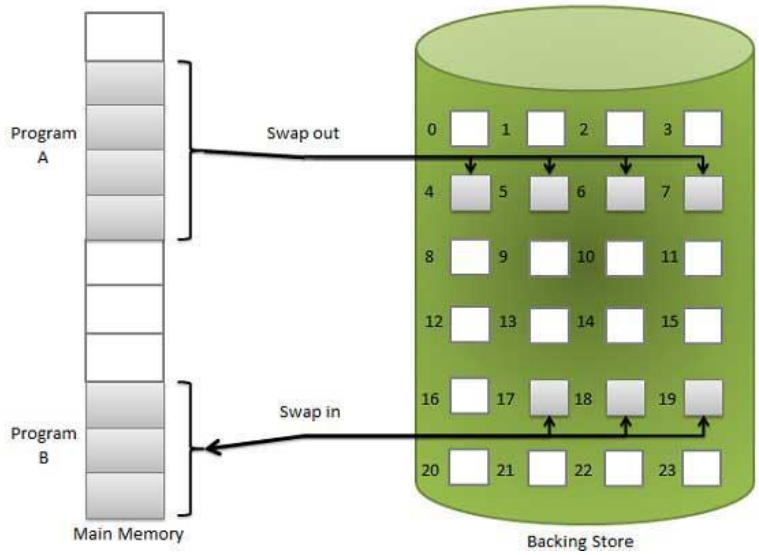
- Virtual memory is a technique that allows the execution of processes which are not completely available in memory. The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory is the separation of user logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. Following are the situations, when entire program is not required to be loaded fully in main memory.
- User written error handling routines are used only when an error occured in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.



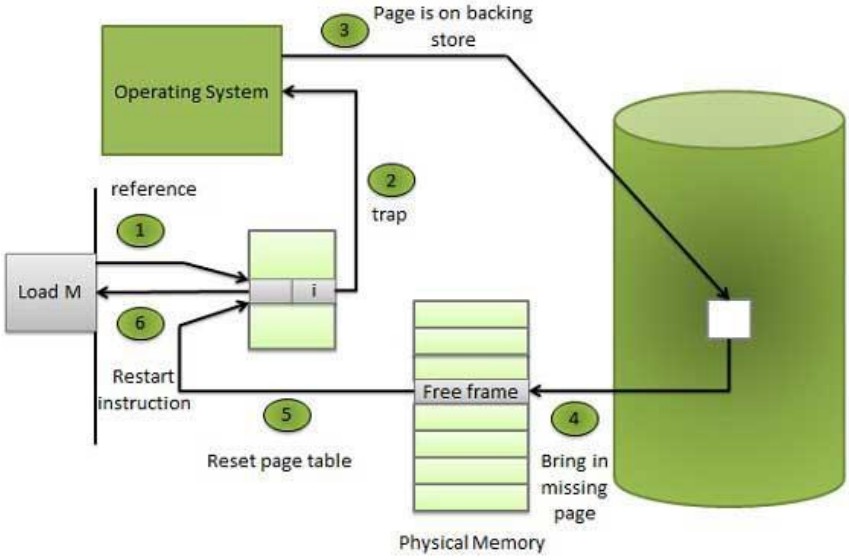
- Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Demand Paging

- A demand paging system is quite similar to a paging system with swapping. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper called pager.
- When a process is to be swapped in, the pager, guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.
- Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme, where valid and invalid pages can be checked by checking the bit. Marking a page will have no effect if the process never attempts to access the page. While the process executes and accesses pages that are memory resident, execution proceeds normally.



- Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory. But page fault can be handled as following



Step	Description
Step 1	Check an internal table for this process, to determine whether the reference was a valid or it was an invalid memory access.
Step 2	If the reference was invalid, terminate the process. If it was valid, but page have not yet brought in, page in the latter.
Step 3	Find a free frame.
Step 4	Schedule a disk operation to read the desired page into the newly allocated frame.
Step 5	When the disk read is complete, modify the internal table kept with the process and the page table to indicate that the page is now in memory.
Step 6	Restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory. Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory.

Advantages

Following are the advantages of Demand Paging

- Large virtual memory.
- More efficient use of memory.
- Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

Disadvantages

Following are the disadvantages of Demand Paging

- Number of tables and amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to the lack of an explicit constraint on a job address space size.

POSSIBLE QUESTIONS

UNIT – III

PART – A (20 MARKS)

(Q.NO 1 TO 20 Online Examinations)

PART – B (2 MARKS)

1. What is paging?
2. What is meant by Segmentation?
3. What is Fragmentation?
4. Define Page table
5. List some Memory Allocation Strategies.
6. Define Virtual Address Space

PART – C (6 MARKS)

1. Explain about Memory Allocation Strategies.
2. Explain the process of Fixed and Variable partition.
3. Explain about Virtual address space.
4. Discuss in detail about Paging in detail
5. Explain the concept of Physical address space in detail.
6. Discuss in detail about Segmentation.
7. Explain the process of swapping
8. Difference between Paging and Segmentation
9. Comparison between paging and Fragmentation
10. Difference between Physical address space and Virtual Address space

KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore – 641 021.

(For the Candidates admitted from 2018 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

UNIT - III : (Objective Type Multiple choice Questions each Question carries one Mark)

OPERATING SYSTEMS

PART - A (Online Examination)

Questions	Opt1	opt2	opt3	opt4	KEY
Memory is array of _____	bytes	circuits	ics	ram	bytes
CPU fetches instructions from _____	memory	pendrive	dvd	cmos	memory
Program must be in _____	dvd	pendrive	memory	cmos	memory
Collection of process in disk forms_____	input queue	output queue	stack	circle	input queue
Address space of computer starts at _____	3333	4444	0000	2222	0000
If process location is found during compile time t	relative	absolute	approximate	more or less	absolute
Address generated by CPU is _____ a	logical	physical	direct	indirect	logical
Logical address can be also called as _____	physical	virtual	direct	indirect	virtual
Run time mapping is done using _____	MMU	CPU	CU	IU	MMU
In address binding base register is also called as _	relocation register	memory register	hard disk	pendrive	relocation regis
Better memory space is utilized using _____	dynamic loading	dynamic linking	registers	array of words	dynamic loadin
_____ routine is never loaded in d	unused	used	regular	recursive	unused
Some operating systems support only _____	static	dynamic	temporary	interruptive	static
_____ is a code that locates library rou	stub	dll	recursive routine	exe file	stub
_____ can be used to manage large mem	overlays	swapping	roll in and out	libraries	overlays
_____ error is raised in memory	addressing	swapping	dynamic	index	addressing
Set of _____ are scattered throughout the	holes	gaps	free space	words	holes

_____ can be internal and external	fragmentation	merging	grouping	fixing	fragmentation
_____ is used to divide a process into fix	paging	segmentation	sp	swapping	paging
In paging physical memory is divided into _____	frames	pages	segments	bytes	frames
In paging virtual memory is divided into _____	frames	pages	segments	bytes	pages
_____ is first of virtual address in pagi	page number	segment number	frame number	offset	page number
_____ is second part of virtual address	page number	segment number	frame number	offset	offset
Page mapping entries are found in _____	page table	segment table	hash table	pointing table	page table
Page size is defined by _____	hardware	software	os	kernel	hardware
_____ is first in mapping of virtual to p	direct	associate	direct & associative	pointing	direct
_____ is second in mapping of virtual t	direct	associate	direct & associative	pointing	associate
_____ is third in mapping of virtual to	direct	associate	direct & associative	pointing	direct & associa
_____ is used to divide a process into va	paging	segmentation	sp	swapping	segmentation
In segmentation virtual memory is divided into _	frames	pages	segments	bytes	segments
_____ view is supported in segmentation	user	system	cpu	manager	user
_____ is format for segmentation virtual	(s,d)	(p,d)	(v,d)	(k,d)	(s,d)
_____ is the first element in segment tab	limit	base	offset	page number	limit
_____ is the second element in segment	limit	base	offset	page number	base
Addressing in segmentation is similar as _____	direct	associate	direct & associative	pointing	direct
How many elements are there in segmentation ad	1	2	3	4	3
_____ is organization in physical memor	frames	pages	segments	bytes	frames
_____ memory is used to manage inc	virtual	physical	rom	eprom	virtual
virtual memory abstracts _____ memory	virtual	eerom	main	eprom	main
_____ reasons are there for existence	1	2	3	4	3
_____ benefits are there from virtual r	1	2	3	4	3
Virtual memory is commonly implemented by __	demand	bargain	quarrel	order	demand
_____ fault occurs when desired pag	page	segment	pages	segments	page
_____ table is used in demand paging	page	segment	pages	segments	page
_____ methods are there for process c	1	2	3	4	2
_____ method implements partial sha	copy on write	memory mapping	paging	segmentation	copy on write
_____ is done for page fault	replacement	swapping	logging	locking	replacement
_____ is unrealizable page replaceme	optimal	FIFO	LRU	NRU	optimal
_____ is first page replacement algori	optimal	FIFO	LRU	NRU	FIFO
_____ is second page replacement alg	optimal	FIFO	LRU	NRU	optimal

_____ is third page replacement algor	optimal	FIFO	LRU	NRU	NRU
_____ is associated with each page in	label	index	number	identity	label
_____ labelled page replaced in optim	highest	lowest	moderate	below average	highest
_____ end page is removed in fifo alg	rear	head	top	bottom	head
Modified version of fifo algorithm gives _____	1	2	3	4	2
_____ is called as high paging activity	thrashing	smashing	mocking	breaking	thrashing
_____ occurs frequently during thrashi	page fault	segment fault	memory fault	address fault	page fault
_____ strategy is used to solve thrashi	working set	pff	lpr algorithm	gpl algorithm	working set
_____ algorithm is used to solve thras	working set	pff	lpr algorithm	gpl algorithm	lpr algorithm
_____ is a basic solution for thrashing	working set	pff	lpr algorithm	gpl algorithm	pff

UNIT – IV

SYLLABUS

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

FILE AND I/O

MANAGEMENT DIRECTORY STRUCTURE

- 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. 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.

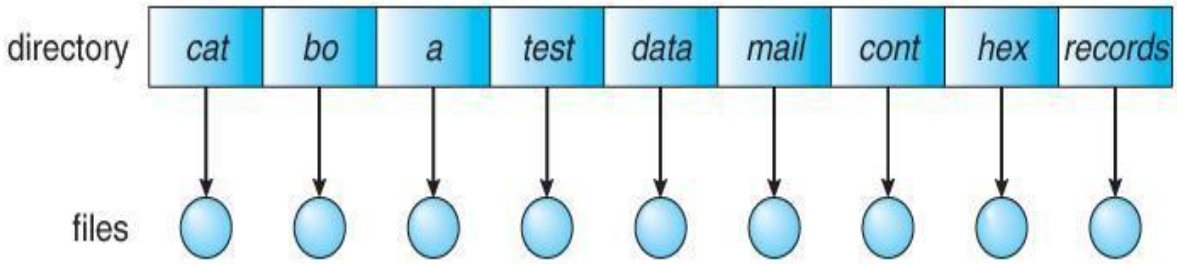
• **Rename a file**. Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.

• **Traverse the file system**. Wemay 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.

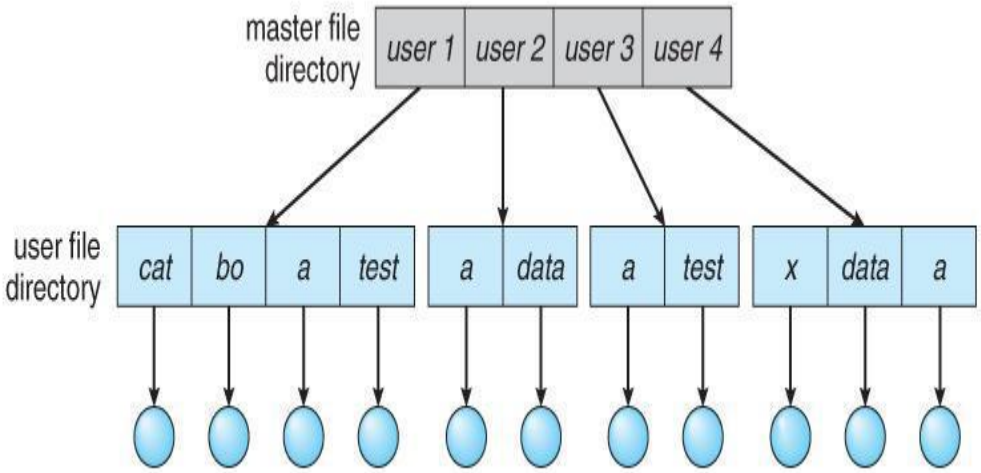
Single-Level Directory

It is Simple to implement, but each file must have a unique name.



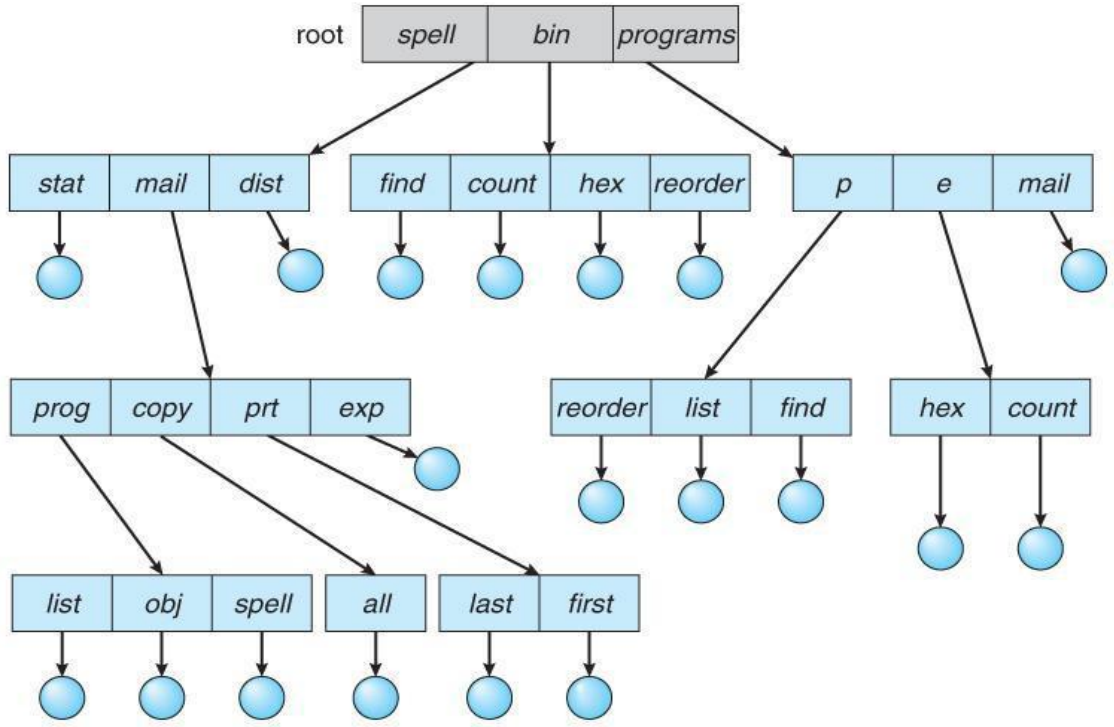
Two-Level Directory

- In this structure each user gets their own directory space. File names only need to be unique within a given user's directory. A master file directory is used to keep track of each user's directory, and must be maintained when users are added to or removed from the system.
- A separate directory is generally needed for system (executable) files. Systems may or may not allow users to access other directories besides their own If access to other directories is allowed, then provision must be made to specify the directory being accessed. If access is denied, then special consideration must be made for users to run programs located in system directories. A **search path** is the list of directories in which to search for executable programs, and can be set uniquely for each user.



Tree-Structured Directories

- An obvious extension to the two-tiered directory structure, and the one with which we are all most familiar. Each user / process has the concept of a **current directory** from which all (relative) searches take place. Files may be accessed using either absolute pathnames (relative to the root of the tree) or relative pathnames (relative to the current directory.) Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands. One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.

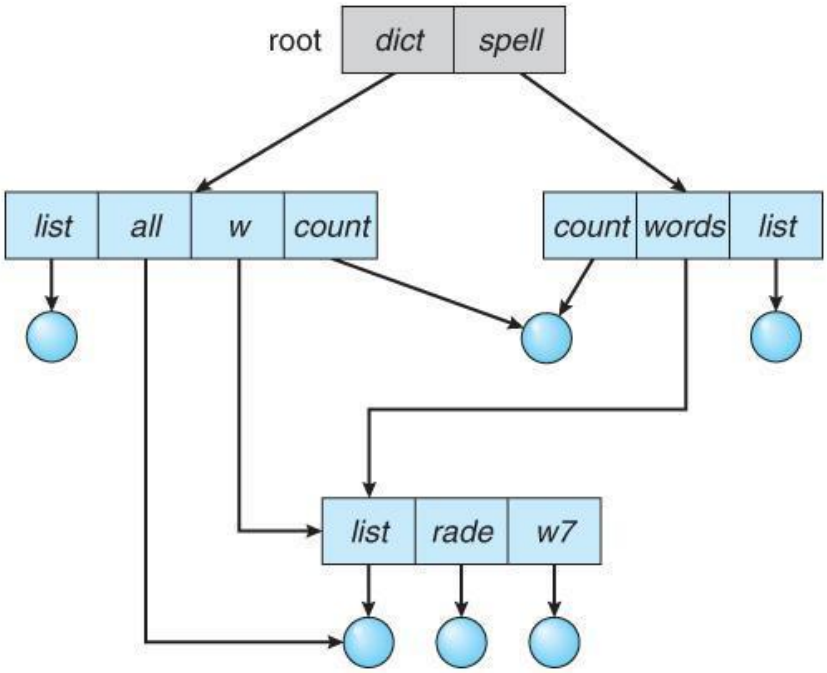


Acyclic-Graph Directories

- When the same files need to be accessed in more than one place in the directory structure (e.g. because they are being shared by more than one user / process), it can be useful to provide an acyclic-graph structure. (Note the **directed** arcs from parent to child.)

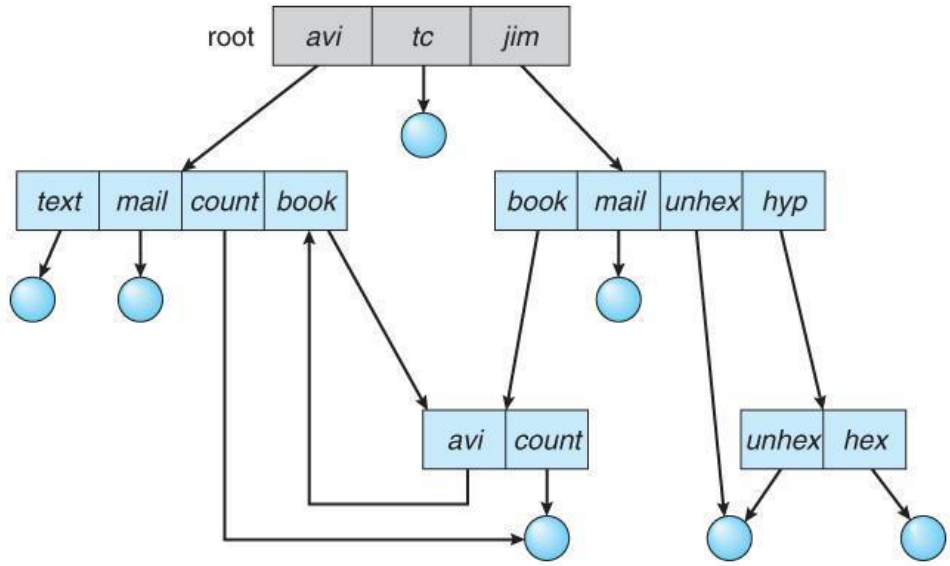
  UNIX provides two types of **links** for implementing the acyclic-graph structure. (See "man ln" for more details.)
- A **hard link** (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same filesystem.
- A **symbolic link**, that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other file systems, as well as ordinary files in the current file system.
- Windows only supports symbolic links, termed **shortcuts.** Hard links require a **reference count**, or **link count** for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed. For symbolic links

there is some question as to what to do with the symbolic links when the original file is moved or deleted: One option is to find all the symbolic links and adjust them also. Another is to leave the symbolic links dangling, and discover that they are no longer valid the next time they are used. What if the original file is removed, and replaced with another file having the same name before the symbolic link is next used?



General Graph Directory

- If cycles are allowed in the graphs, then several problems can arise: Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms. (Or not to follow symbolic links, and to only allow symbolic links to refer to directories.) Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero.
- Periodic garbage collection is required to detect and resolve this problem. (chkdsk in DOS and fsck in UNIX search for these problems, among others, even though cycles are not supposed to be allowed in either system. Disconnected disk blocks that are not marked as free are added back to the file systems with made-up file names, and can usually be safely deleted.)



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. Let's examine what the operating system must do to perform each of these six basic file operations.

  - **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.

FILE ALLOCATION METHODS

There are three major methods of storing files on disks: contiguous, linked, and indexed.

Contiguous Allocation

- **Contiguous Allocation** requires that all blocks of a file be kept together contiguously. The performance is very fast, because reading successive blocks of the same file generally requires no movement of the disk heads, or at most one small step to the next adjacent cylinder.

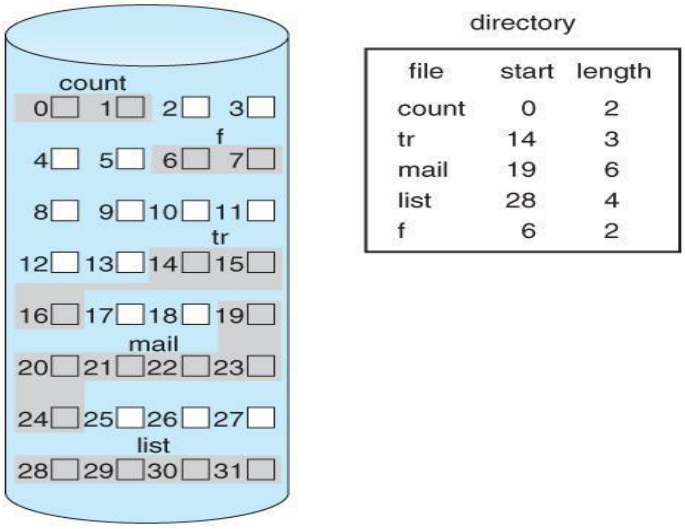
- Storage allocation involves the same issues discussed earlier for the allocation of contiguous blocks of memory (first fit, best fit, fragmentation problems, etc.) The distinction is that the high time penalty required for moving the disk heads from spot to spot may now justify the benefits of keeping files contiguously when possible.

Problems can arise when files grow, or if the exact size of a file is unknown at creation time:

- Over-estimation of the file's final size increases external fragmentation and wastes disk space.

- Under-estimation may require that a file be moved or a process aborted if the file grows beyond its originally allocated space.
- If a file grows slowly over a long time period and the total final space must be allocated initially, then a lot of space becomes unusable before the file fills the space.

A variation is to allocate file space in large contiguous chunks, called **extents.** When a file outgrows its original extent, then an additional one is allocated. (For example an extent may be the size of a complete track or even cylinder, aligned on an appropriate track or cylinder boundary.) The high-performance files system VERITAS uses extents to optimize performance.

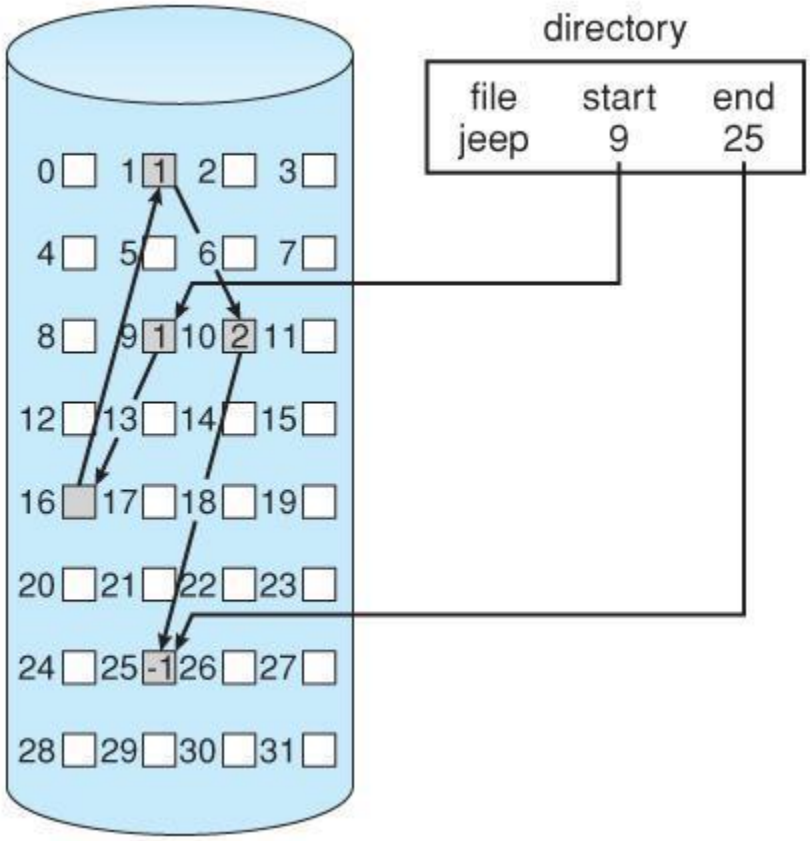


Contiguous allocation of disk space.

Linked Allocation

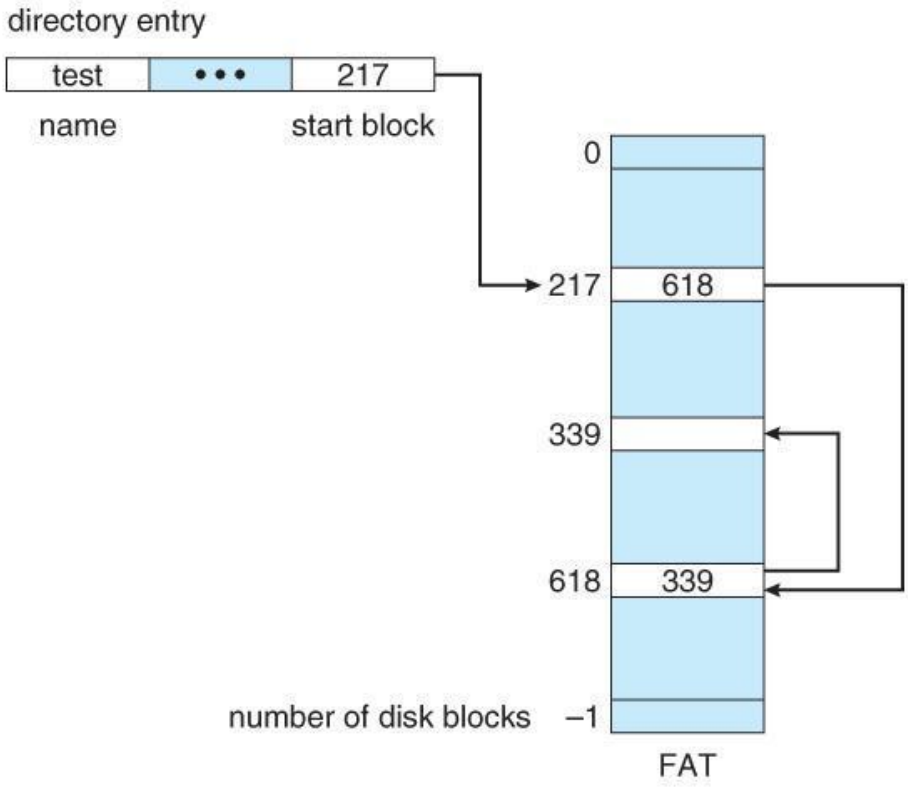
- Disk files can be stored as linked lists, with the expense of the storage space consumed by each link. (E.g. a block may be 508 bytes instead of 512.) Linked allocation involves no external fragmentation, does not require pre-known file sizes, and allows files to grow dynamically at any time. Unfortunately linked allocation is only efficient for sequential access files, as random access requires starting at the beginning of the list for each new location access. Allocating **clusters** of blocks reduces the space wasted by pointers, at the cost of internal fragmentation. Another big problem with linked allocation is reliability if

a pointer is lost or damaged. Doubly linked lists provide some protection, at the cost of additional overhead and wasted space.



Linked allocation of disk space.

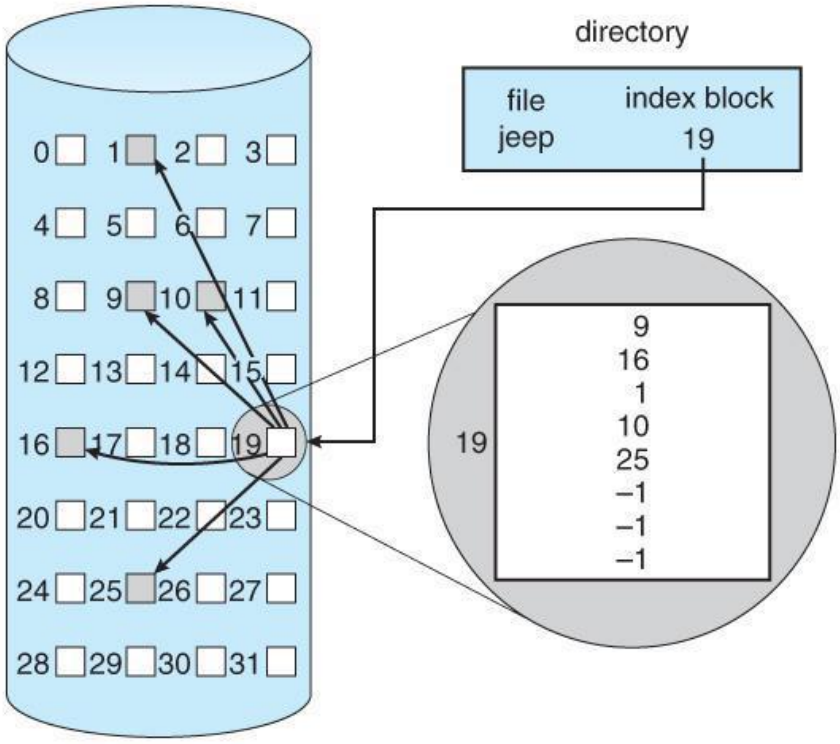
The **File Allocation Table, FAT,** used by DOS is a variation of linked allocation, where all the links are stored in a separate table at the beginning of the disk. The benefit of this approach is that the FAT table can be cached in memory, greatly improving random access speeds.



File-allocation table.

Indexed Allocation

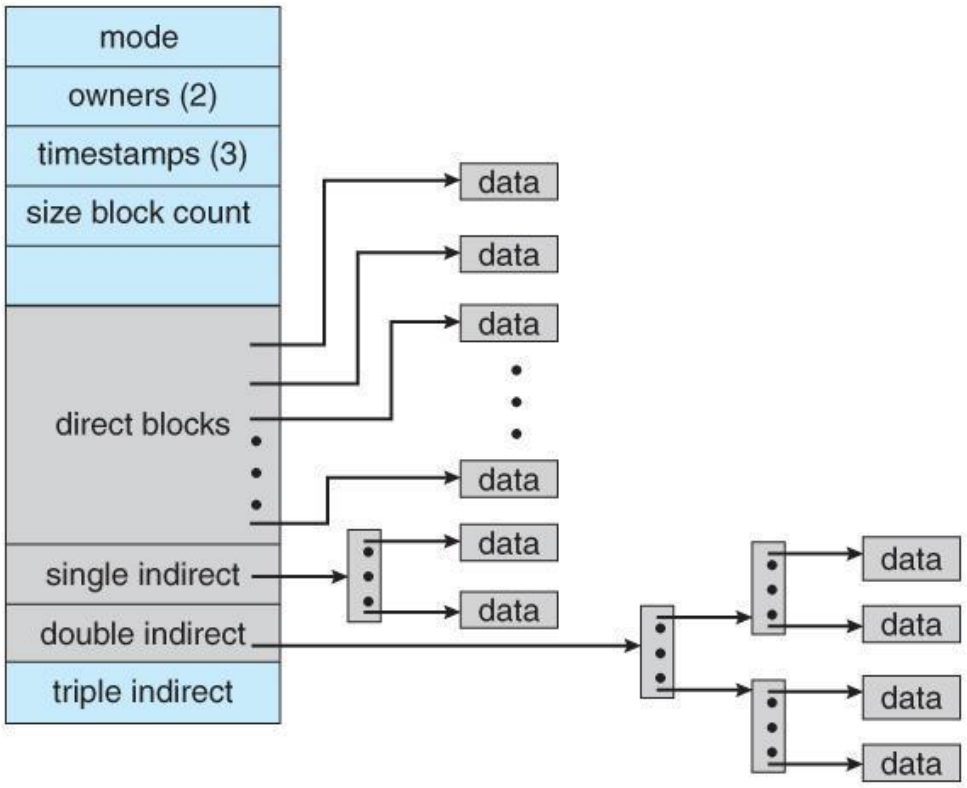
- **Indexed Allocation** combines all of the indexes for accessing each file into a common block (for that file), as opposed to spreading them all over the disk or storing them in a FAT table.
- Some disk space is wasted (relative to linked lists or FAT tables) because an entire index block must be allocated for each file, regardless of how many data blocks the file contains. This leads to questions of how big the index block should be, and how it should be implemented. There are several approaches:
- **Linked Scheme** - An index block is one disk block, which can be read and written in a single disk operation. The first index block contains some header information, the first N block addresses, and if necessary a pointer to additional linked index blocks.



Indexed allocation of disk space.

- **Multi-Level Index -** The first index block contains a set of pointers to secondary index blocks, which in turn contain pointers to the actual data blocks.

- **Combined Scheme -** This is the scheme used in UNIX inodes, in which the first 12 or so data block pointers are stored directly in the inode, and then singly, doubly, and triply indirect pointers provide access to more data blocks as needed. The advantage of this scheme is that for small files (which many are), the data blocks are readily accessible (up to 48K with 4K block sizes); files up to about 4144K (using 4K blocks) are accessible with only a single indirect block (which can be cached), and huge files are still accessible using a relatively small number of disk accesses (larger in theory than can be addressed by a 32-bit address, which is why some systems have moved to 64-bit file pointers.)



The UNIX inode.

Performance

- The optimal allocation method is different for sequential access files than for random access files, and is also different for small files than for large files. Some systems support more than one allocation method, which may require specifying how the file is to be used (sequential or random access) at the time it is allocated.
- Such systems also provide conversion utilities. Some systems have been known to use contiguous access for small files, and automatically switch to an indexed scheme when file sizes surpass a certain threshold. And of course some systems adjust their allocation schemes (e.g. block sizes) to best match the characteristics of the hardware for optimum performance.

DISK MANAGEMENT

Disk Formatting

- Before a disk can be used, it has to be low-level formatted, which means laying down all of the headers and trailers marking the beginning and ends of each sector. Included in the header and trailer are the linear sector numbers, and error-correcting codes, ECC, which allow damaged sectors to not only be detected, but in many cases for the damaged data to be recovered (depending on the extent of the damage.) Sector sizes are traditionally 512 bytes, but may be larger, particularly in larger drives.
- ECC calculation is performed with every disk read or write, and if damage is detected but the data is recoverable, then a soft error has occurred. Soft errors are generally handled by the on-board disk controller, and never seen by the OS.
- Once the disk is low-level formatted, the next step is to partition the drive into one or more separate partitions. This step must be completed even if the disk is to be used as a single large partition, so that the partition table can be written to the beginning of the disk.
- After partitioning, then the file-systems must be logically formatted, which involves laying down the master directory information (FAT table or inode structure), initializing free lists, and creating at least the root directory of the file-system. (Disk partitions which are to be used as raw devices are not logically formatted. This saves the overhead and disk space of the file-system structure, but requires that the application program manage its own disk storage requirements.

Boot Block

- Computer ROM contains a bootstrap program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it. (The ROM bootstrap program may look in floppy and/or CD drives before accessing the hard drive, and is smart enough to recognize whether it has found valid boot code or not.). The first sector on the hard drive

is known as the Master Boot Record, MBR, and contains a very small amount of code in addition to the partition table. The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the active or boot partition.

- The boot program then looks to the active partition to find an operating system, possibly loading up a slightly larger / more advanced boot program along the way. In a dual-boot (or larger multi-boot) system, the user may be given a choice of which operating system to boot, with a default action to be taken in the event of no response within some time frame.

- Once the kernel is found by the boot program, it is loaded into memory and then control is transferred over to the OS. The kernel will normally continue the boot process by initializing all important kernel data structures, launching important system services (e.g. network daemons, sheds, init, etc.), and finally providing one or more login prompts. Boot options at this stage may include single-user a.k.a. maintenance or safe modes, in which very few system services are started - These modes are designed for system administrators to repair problems or otherwise maintain the system.

POSSIBLE QUESTIONS

UNIT – IV

PART – A (20 MARKS)

(Q.NO 1 TO 20 Online Examinations)

PART – B (2 MARKS)

1. What is meant by Linked allocation?
2. What are the file allocation methods?
3. What is the process of Device management?
4. List the File Authentication Methods
5. List the various File operations

PART – C (6 MARKS)

1. Describe about files and explain the access methods for files.
2. Explain the various File operations.
3. Explain about Device management.
4. Discuss in detail about directory structure
5. Explain about contiguous file allocation method.
6. Discuss in detail about Linked file allocation method.
7. Explain about Indexed file allocation method.
8. Explain the process of file operation with suitable example
9. Discuss in detail about File structure and file access mechanisms
10. Describe the process of I/O Management in Operating System

KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore – 641 021.

(For the Candidates admitted from 2018 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

UNIT - IV : (Objective Type Multiple choice Questions each Question carries one Mark)

OPERATING SYSTEMS

PART - A (Online Examination)

Questions	Opt1	opt2	opt3	opt4	KEY
The file system consist of ---------- Distinct parts	2	3	4	5	2
A ---------- File is a sequence of character organized into lines	Source	Object	Text	Executable	Text
A ---------- File is a sequence of subroutines and functions	Source	Object	Text	Executable	Source
The operating system keeps a small table called the ---------- ,containing information about all open files	Show file table	Visible file table	Open file ta	Manage file table	Open file table
A file is executed in ---------- extension	External structure	.bat	.mdb	.in	.bat
The .bat file is a ----------containing in ANCII format,command to the operating system	Binary file	Batch file	Text file	Word file	Batch file
The file type is used to indicate the ---------- of the file	.txt	Internal structure	Block structure	Outer structure	Internal structure
Information in the file is processed in the order called- ----------	Direct access	Sequence access	Dynamic access	Random access	Sequence access
A file is made up of fixed length that allows the program to read and write record rapidly in no	Direct access	Sequence access	Dyanamic access	Random access	Direct access

Data cannot be written in secondary storage unless written with in a ----------	File	Swap space	Directory	Text format	File
File attribute consist of ----------	Name,Type,Content	Name,type,Size	Seperate directory system	Name,identifier	Name,Size,Type,identifier
The information about all files is kept in ----------	swap space	operating system	Name,Size,Type,Identifier	Hard disk	Seperate directory
A file is a ---------- type	Abstract	Primitive	Public	Private	Abstract
In UNIX Open system call returns ----------	pointer to the entry in the open	pointer to the entry in the	A file to the process calling it	pointer to the entry in	pointer to the entry in the
The open file table has a ---------- Associated with each file	File content	File permission	open count	Close count	open count
The file name is generaly split into which of the two parts ----------	Name and type	Name and identifier	Name and extension	Extension and type	Name and extension
In the sequential access method, information in the file is processed	One disk after the other	One record after the other	One text document after	One name after the	One record after the
Sequential access method ----------,on random access devices	Works well	Dosen't works well	Works slow	Works normal	Works well
The direct access method is based on a ---------- model of a file as---------- allow random access to	Magnetic tape,magnetic	Tape,Tapes	Disk,Disks	Tape,Disk	Disk,Disks
A relative block number is an index relative to ----------	The beginning of the file	The end of the file	The last written position in file	Middle of the file	The beginning of the file
The index contains ----------	Name of all content of file	Pointer to each page	Pointers to the various blocks	Pointer to same page	Pointers to the various
The directory can be viewed as a ----------,that translate the file name into their directory entries	Symbol table	Partition	Swap space	Cache	Symbol table
In the single level directory: ----------	All files are contain in	All files are contained in	Depend on the operating system	Depend on the file name	All files are contained in
In the single level directory ----------	All directory must have a unique	All files must have a unique	All files must have a unique owner	All files must have a	All files must have a unique
In the two level directory structure ----------	own user file directory	has its own master file	)Both a and b	has its different file	Both a and b

When a user refers to a particular file ----------	System MFD is searched	His own UFD is searched	Both MFD and UFD are searched	directory is searched	and UFD are searched
The disadvantage of the two level directory structure is that	the name collision problem	name collision	users from one another	users from one another	users from one another
In the tree structure directory ----------	The tree has the same directory	The tree has the leaf	The tree has the root directory	The tree has no directory	The tree has the root
The three major methods of allocating disk space that are in wide use are ----------	Contiguous,Linked ,Hashed	Contiguous,Lin ked,Indexed	Linked,Hashed,Ind exed	Contiguous ,Linked	Contiguous,Li nked,Indexed
In Contiguous allocation ----------	occupy a set of contiguous block	linked list of disk blocks	All the pointers to scattered	All the files are blocked	occupy a set of contiguous
In linked allocation ----------	Each file must occupy a set of	Each file is a linked list of	All the pointers to scattered	All the files are blocked	Each file is a linked list of
In indexed allocation ----------	Each file must occupy a set of	Each file is a linked list of	All the pointers to scattered blocks	All the files are blocked	All the pointers to
[illegible] system, the decision to load a particular one is done by ----------	Boot loader	Boot strap	Process control block	File control block	Boot loader
The VFS refers to ----------	Virtual File System	Valid File System	Virtual Font System	Virtual Function	Virtual File System
The disadvantage of a linear list of directory entries is the ----------	Size of the linear list in the memory	Linear search to find a file	It is not reliable	It is not valid	Linear search to find a file
One difficulty of contiguous allocation is ----------	Finding space for a new file	Ineffecient	Costly	Time taking	Finding space for a new file
To solve the problem of external fragmentation ---------- needs to be done periodically	Compaction	Check	Formatting	Replacing memory	Compaction
If too little space is allocated to a file ----------	The file will not work	There will not be any space	The file cannot be extended	file cannot be opened	cannot be extended
A system program such as fsck ---------- is a consistency checker	UNIX	Windows	Macintosh	Solaris	UNIX
Each set of operations for performing a specific task is a ----------	Program	Code	Transaction	Method	Transaction

Once the changes are written to the log, they are considered to be ----------	Committed	Aborted	Completed	Finished	Committed
When an entire command transaction is completed,----------	It is stored in the memory	from the log file	It is redone	from the memory	from the log file
In ---------- information is recorded magnetically on platters	Magnetic disk	Electrical disk	Assemblies	Cylinders	Magnetic disk
The head of the magnetic disk are attached to a ---------- that moves all the head as unit	Spindle	Disk arm	Track	Pointer	Disk arm
The set of tracks that are at one arm position make up a ----------	Magnetic disk	Electrical disk	Assemblies	Cylinders	Cylinders
The time taken to move a disk arm to the desired cylinder is called as----------	Positioning time	Random access ti	Seek time	Rotational latency	Seek time
When a head damages the magnetic surface, it is known as ----------	Disk crash	Head crash	Magnetic damage	All of these	Head crash
A flopy disk is designed to rotate ---------- as compared to a hard disk drive	Faster	Slower	At the same speed	Normal speed	Slower
The host controller is ----------	the end of each disk	the computer end of the bus	Both a and b	the system side	the computer end of the bus
The process of dividing a disk into sectors that the disk controller can read and write, before a disk can store data is known as----------	Partitioning	Swap space creation	Low-level formatting	Physical formatting	Low-level formatting ,Physical
the data structure for a sector typically contains ----------	Header	Data area	Trailer	Main section	Header ,Data area ,Trailer
The header and trailer of a sector contains information used by the disk controller such as _____.	Main section	Error corecting codes	Sector number	Disk identifier	Sector number
The two steps that the operating system takes to use a disk to hold its files are ---------- and ----------	partitioning	Swap space creation	Catching	Logical formatting	partitioning
The ---------- program initializes all aspects of the system, from CPU registers to device controllers and the content of main memory, and then starts the	Main	Boot loader	Boot strap	ROM	Boot strap

For most computers the boot strap is stored in----------	RAM	ROM	Cache	Tertiary storage	ROM
A disk that has a boot partition is called a ----------	Start disk	Destroyed blocks	Boot disk	Format disk	System disk,boot disk
Defective sectors on disks are often known as ----------	Good blocks	System disk	Bad blocks	Semi blocks	Bad blocks
Bad blocks are called as _____	Good Sectors	Defective Sectors	boot disks	boot strap	Defective Sectors
ROM got _____ file	boot strap	Data area	head data	random data	boot strap

UNIT – V

SYLLABUS

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

PROTECTION AND 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

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 can not 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 suggest 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 mis-used. 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.

POLICY MECHANISM

- **Protection:** mechanisms that prevent accidental or intentional misuse of a system.
  - Accidents: generally easier to solve (make them unlikely)
  - Malicious abuse: much more difficult to eliminate (can't leave any loopholes, can't use probabilities).

- **Three aspects to a protection mechanism:**
  - Authentication: identify a responsible party (*principal*) behind each action.
  - Authorization: determine which principals are allowed to perform which actions.
  - Access enforcement: combine authentication and authorization to control access.
- A tiny flaw in any of these areas can compromise the entire protection mechanism.

AUTHENTICATION

- Typically done with *passwords*:
  - A secret piece of information used to establish identity of a user.
  - Must not be stored in a directly-readable form: use one-way transformations.
  - Passwords should be relatively long and obscure.
- Alternate form of authentication: badge or key.
  - Does not have to be kept secret.
  - Should not be forgable or copyable.
  - Can be stolen, but owner should know if it is.
- Paradox: key must be cheap to make, hard to duplicate.
- Once authentication is complete, the identity of the principal must be protected from tampering, since other parts of the system will rely on it.
- Once you log in, your user id is associated with every process executed under that login: each process inherits the user id from its parent.

INTERNAL ACCESS AUTHORIZATION

Authorization:

- Goal: determine which principals can perform which operations on which objects.
- Logically, authorization information represented as an access matrix:
- One row per principal.
- One column per object.
- Each entry indicates what that principle can do to that object.

- In practice a full access matrix would be too bulky, so it gets stored in one of two compressed ways: access control lists or capabilities.
- Access Control Lists (ACLs): organize by columns.
- With each object, store information about which users are allowed to perform which operations.
- Most general form: list of <user, privilege> pairs.
- For simplicity, users can be organized into groups, with a single ACL for an entire group.
- ACLs can be very general (Windows) or simplified (Unix).
- UNIX: 9 bits per file:
- owner, group, anyone
- read, write, execute permissions for each of the above
- In addition, user "root" has all permissions for everything
- ACLs are simple and are used in almost all file systems.
- Capabilities: organize by rows.
- With each user, indicate which objects may be accessed, and in what ways.
- Store a list of <object, privilege> pairs with each user. This is called a capability list.
- Typically, capabilities also act as names for objects: can't even name objects not referred to in your capability list.
- Almost as if there were no root directory in Unix and no "..".
- Systems based on ACLs encourage visibility of objects: shared public namespace.

- Capability systems discourage visibility; namespaces are private by default.
- Capabilities have been used in experimental systems attempting to be very secure. However, they have proven to be clumsy to use (painful to share things), so they have mostly fallen out of favor for managing objects such as files.
- Example of a simple capability-based protection scheme: page tables.
-

Access Enforcement

- Some part of the system must be responsible for enforcing access controls and protecting authentication and authorization info.
- This portion of the system has total power, so it should be as small and simple as possible. Example: the portion of the system that sets up page tables.
- Security kernel: an inner layer of the operating system that enforces security; only this layer has total power.
- Most operating systems have no security kernel: the entire OS has unlimited power.
- Miscellaneous Issues
- There are many other things that need to be protected besides just file access
- In Unix, root access is used to control most of these things.

Some common problems:

- Account penetration
- Abuse of valid privileges.
- Trojan Horse: modify valid program to misbehave or steal information.
- Impersonation/phishing: create the appearance of a trusted application, trick users into divulging personal information
- Network attack: snoop on network traffic or other communications and steal unprotected information (e.g. passwords).
- Denial of service: create program that uses up all system resources to make system crash or prevent others from getting work done
- Worm or virus: a Trojan Horse that can spread itself from machine to machine (exploiting bugs and loopholes)

Examples of successful attacks:

- Tenex page-fault password attack
- Botnets and denial of service
- "Salami attack": checking account interest calculator that credited fractional cents to the account of the creator
- It may not be possible to tell that a system has been penetrated. Example of undetectable Trojan Horse:
- Modify login program to recognize special login and give root privilege without a password.
- But, people might notice the login code.
- So, modify the compiler to figure out when it's compiling the login code and insert the Trojan Horse automatically.
- But, people might notice the compiler code.
- Modify the compiler to insert the compiler Trojan Horse.
- Compile the compiler.
- Remove the Trojan Horse from the sources.
- The Trojan Horse is completely hidden in the binaries!
- Once penetrated, it may be difficult or impossible to secure it again: too many complex Trojan Horses.
- Any bug can result in a security loophole, and all systems have bugs.
- Security Solutions
- Logging: record important actions and uses of privilege
- Principle of minimum privilege: limit access to only what is absolutely needed.
- Involve humans more:
- Auditing code to catch bugs and Trojan Horses.
- Human approval for particularly dangerous operations (e.g., large funds transfers)

- Prove correctness of system (absence of bugs)
- Information flow control:
- Control not only who can access what, but what they can do with the information once they have it.
- Goal: prevent Trojan Horses
- Example: can my editor leak my files out onto the Internet?
- Many attempts at information flow control, none that both work and are convenient to use.
- Example: covert channels
- Use some observable property of the system as a channel for signaling to an accomplice.
- E.g. modulate CPU load, accomplice observes.

POSSIBLE QUESTIONS

UNIT – V

PART – A (20 MARKS)

(Q.NO 1 TO 20 Online Examinations)

PART – B (2 MARKS)

1. What is meant by Access Control List?
2. Define Internal access authorization
3. What are Authentication mechanisms?
4. Define Program threats
5. What is meant by OS Security?
6. Define Trojan horse

PART – C (6 MARKS)

1. Explain about OS Protection and Security.
2. Discuss in detail about Policy mechanism.
3. Explain the process of Internal access authorization.
4. Describe the Features of Security in Operating System.
5. Discuss in detail about Authentication.
6. Explain the protection for Unix Files and Directories
7. Describe the process of protection in Operating System.
8. Discuss about the Configuration of User Authentication
9. Describe the process of Threats in Operating System.
10. Discuss in detail about Policy versus Mechanism

KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore – 641 021.

(For the Candidates admitted from 2018 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

UNIT - V : (Objective Type Multiple choice Questions each Question carries one Mark)

OPERATING SYSTEMS

PART - A (Online Examination)

Questions	Opt1	opt2	opt3	opt4	KEY
computer system assets can be modified only by authorized parities.	Confidentiality	Integrity	Availability	Authenticity	Integrity
In computer security, …….. means that the information in a computer system only be accessible for reading by authorized parities.	Confidentiality	Integrity	Availability	Authenticity	Confidentiality
Which of the following is independent malicious program that need not any host program?	Trap doors	Trojan horse	Virus	Worm	Worm
The …….. is code that recognizes some special sequence of input or is triggered by being run from a certain user ID of by unlikely sequence of events.	Trap doors	Trojan horse	Logic Bomb	Virus	Trap doors
program that is set to “explode” when certain conditions are met.	Trap doors	Trojan horse	Logic Bomb	Virus	Trap doors
Which of the following malicious program do not replicate automatically?	Trojan Horse	Virus	Worm	Zombie	Trojan Horse
…… programs can be used to accomplish functions indirectly that an unauthorized user could not	Zombie	Worm	Trojan Horses	Logic Bomb	Trojan Horses
programs by modifying them, the modification includes a copy of the virus program, which can go on to infect	Worm	Virus	Zombie	Trap doors	Virus
the systems be given just enough privileges to perform their task?	principle of operating system	principle of least privilege	principle of process scheduling	none of the mentioned	principle of least privilege

_____ is an approach to restricting system access to authorized users.	Role-based access control	Process-based access control	Job-based access control	none of the mentioned	Role-based access control
For system protection, a process should access	all the resources	resources for which it has	authorization is not required	all of the mentioned	resources for which it has
The protection domain of a process contains	object name	rights-set	object name and rights-set	none of the mentioned	object name and rights-set
If the set of resources available to the process is fixed throughout the process's lifetime then its domain is	static	dynamic	neither static nor dynamic	none of the mentioned	static
Access matrix model for user authentication contains	a list of objects	a list of domains	a function which returns an object's	all the options	all the options
Global table implementation of matrix table contains	domain	object	right-set	all the options	all the options
For a domain _____ is a list of objects together with the operation allowed on these objects.	capability list	access list	authorization	none of the mentioned	capability list
Which one of the following is capability based protection system?	hydra	cambridge CAP system	hydra & cambridge CAP	none of the mentioned	hydra & cambridge
In UNIX, domain switch is accomplished via	file system	user	superuser	none of the mentioned	file system
_____ is an important property of an operating system that hopes to keep up with advancements in	Portability	Reliability	Extensibility	compatibility	Extensibility
_____ is the ability to handle error conditions, including the ability of the operating system to protect itself and its users from defective or malicious software	Portability	Reliability	Extensibility	compatibility	Reliability
_____ is the ability to move from one hardware architecture to another with relatively few changes.	Portability	Reliability	Extensibility	compatibility	Portability
Windows NT is designed to afford good _____.	Portability	Reliability	Extensibility	performance	performance
A _____ is created by the NT disk administrator utility, and is based on a logical disk	Volume	File	Directory	subdirectory	Volume
A _____ of a directory contains the top level of the B+ tree.	index root	file reference	attributes	metadata	index root
The _____ in NT may occupy a portion of a disk, may occupy an entire disk or may span across	Volume	File	Directory	subdirectory	Volume

The _____ of a directory contains the top level of the B+ tree.	Volume	File	index root	subdirectory	index root
The _____ attribute contains the access token of the owner of the file, and an access control list	Portability	Recovery	Reliability	Security	Security
To deal with disk sectors that go bad, _____ uses a hardware technique called sector spanning.	Ps	Valloc	Kmalloc	FtDisk	FtDisk
places where they have no business being are called _____	intruders	crackers	hackers	worms	intruders
computers (using viruses and other means) and turn them into _____	virus	worms	malware	zombies	zombies
who may read and write their files and other objects, This policy is called _____	mandatory access co	access matrix	discretionary access control.	access control lists	discretionary access control
Every secured computer system must require all users to be _____at login time	authenticated	authorized	transferred	scheduled	authenticated
The most widely used form of authentication is to require the user to type a _____and a _____`	mailid, PIN number	login name, password.	PIN number, Account number	Username, mailid	login name, password.
characteristics of the user that are hard to forge is called as _____	Biometrics	password	stegnography	access control	Biometrics
_____is the name given to hackers who break into computers for criminal gain	hackers	spoofing	phising	Crackers	Crackers
A typical biometrics system has two parts: _____	enrollment and identification	& authentication	authentication & confidentiality	and authentication	enrollment and identification
Any malware hidden in software or a Web page that people voluntarily download is called _____	worm	Trojan Horse	Virus	Backdoor	Trojan Horse
boot record or the boot sector, with devastating results, such viruses called as _____	device driver virus	source code virus	companion virus	boot sector viruses	boot sector viruses
The trick to infect a device driver leads to a _____	source code virus	device driver virus	companion virus	boot sector viruses	device driver virus
When an attempt is to make a machine or network resource unavailable to its intended users, the attack is called	denial-of-service attack	slow read attack	spoofed attack	starvation attack	denial-of-service attack

The code segment that misuses its environment is called a	internal thief	trojan horse	code stacker	none of the mentioned	trojan horse
malicious function when specified conditions are met, is called	logic bomb	trap door	code stacker	none of the mentioned	logic bomb
The pattern that can be used to identify a virus is known as	stealth	virus signature	armoured	multipartite	virus signature
Which one of the following is a process that uses the spawn mechanism to revage the system performance?	worm	trojen	threat	virus	worm
What is a trap door in a program?	a security hole, inserted at programming time	a type of antivirus	security hole in a network	none of the mentioned	a security hole, inserted at programming
Which one of the following is not an attack, but a search for vulnerabilities to attack?	denial of service	port scanning	memory access violation	dumpster diving	port scanning
File virus attaches itself to the	source file	object file	executable file	all of the mentioned	executable file
Multipartite viruses attack on	files	boot sector	memory	all of the mentioned	all of the mentioned
In asymmetric encryption	same key is used for encryption and decryption	different keys are used for encryption and	the key is required for encryption and decryption	none of the mentioned	different keys are used for encryption and
Which of the following are forms of malicious attack ?	Theft of information	Modification of data	Wiping of information	All of the mentioned	All of the mentioned
What are common security threats ?	File Shredding	File sharing and permission	File corrupting	File integrity	File sharing and permission
From the following, which is not a common file permission ?	Write	Execute	Stop	Read	Stop
Which of the following is a good practice ?	permission for remote	only permission	permission to specified account	read and write	permission to specified
What is not a good practice for user administration ?	Isolating a system after a compromise	random auditing procedures	Granting privileges on a per host basis	and FTP for remote access.	Using telnet and FTP for remote access.

Which of the following is least secure method of authentication ?	Key card	fingerprint	retina pattern	Password	Password
Which of the following is a strong password ?	19thAugust88	Delhi88	P@assw0rd	!augustdelhi	P@assw0rd
What does Light Directory Access Protocol (LDAP) doesn't store ?	Users	Address	Passwords	Security Keys	Address
Which happens first authorization or authentication ?	Authorization	Authentication	Both are same	None of the mentioned	Authorization
What is characteristics of Authorization ?	RADIUS and RSA	handshaking with syn and	protection for securing resources	privileges and rights	privileges and rights
What forces the user to change password at first logon ?	Default behavior of OS	Part of AES encryption practice	Devices being accessed forces the user	Account administrator	Account administrator

Register Number: _____

[18CAU302]

KARPAGAM ACADEMY OF HIGHER EDUCATION
COMPUTER APPLICATIONS
FIRST INTERNAL EXAMINATION- JULY 2019
Third Semester
OPERATING SYSTEMS

Date & Session: 22.07.19 AN
Class: II BCA

Duration : 2 Hours
Maximum : 50 Marks

PART-A (20 X 1 = 20 Marks)
Answer ALL the Questions

1. _____is a program that manages the computer hardware and acts as an intermediary between the computer user and the computer hardware.
a)hardware acceleration b)Operating System c)compiler d)logical transaction
2. _____ manages the execution of user programs to prevent errors and improper use of the computer
a)resource allocator b)work station c)main frame d)control program
3. ____were the first computers used to tackle many commercial & scientific application.
a)Mainframe computer system b)Mainframe computer service
c)multiframe computer system d)multiframe computer service
4. _____contains the address of an instruction to be fetched from memory
a)Program counter (PC) b)Instruction register (IR) c)Control registers d)Status registers
5. _____ is also known as parallel systems or tightly coupled systems a)Multiprocessor systems b)desktop systems c)Time sharing systems d)Multiprogrammed systems
6. _____operating systems are even more complex than multi programmed operating systems.
a)Time-sharing b)desktop systems c)Multiprogrammed systems d)Multiprocessor systems
7. _____ operating system keeps several jobs in memory simultaneously.
a)Time-sharing b)desktop systems c)Multiprogrammed systems d)Multiprocessor systems
8. _____can save more money than multiple single-processor systems
a)Multiprocessor systems b)desktop systems c)Time sharing systems
d)Multiprogrammed systems
9. The most common multiple-processor systems now use
a)symmetric multiprocessing b)asymmetric multiprocessing c)multithreading d)multiprogramming
10. Another form of a special-purpose operating system is the
a)real-time system b)distributed operating system c)Process states d)multiframe computer system
11. The assignment of the CPU to the first process on the ready list is called
a)graceful degradation b)Time-sharing c)dispatching d)Multiprocessor systems
12. The manifestation of a process in an operating system is a
a)Process state transitions b)process control block
c)child process d)cooperating processes
13. For multiprogramming operating system
a)special support from processor is essential b)special support from processor is not essential c)cache memory is essential d)cache memory is not essential
14. Which operating system reacts in the actual time
a)Batch system b)Quick response system
c)Real time system d)Time sharing system
15. The primary job of an OS is to _____
a)command resource b)manage resource c)provide utilities d)Be user friendly
16. The term " Operating System " means _____ a)A set of programs which controls computer working b)The way a computer operator works c)Conversion of high-level language in to machine level language d)The way a floppy disk drive operates
17. With ..... more than one process can be running simultaneously each on a different processer.
a)Multiprogramming b)Uniprocessing c)Multiprocessing d)Uniprogramming
18. The two central themes of modern operating system are
a)Multiprogramming and Distributed processing b)Multiprogramming and Central Processing c)Single Programming and Distributed processing d)None of above
19. ..... is a example of an operating system that support single user process and single thread
a)UNIX b)MS-DOS c)OS/2 d)Windows 2000
20. The operating system of a computer serves as a software interface between the user and the _____.
a)Hardware b)Peripheral c)Memory d)Screen

Part –B (3 x 2 = 6 Marks)
Answer All the Questions

21. What is Operating System?
22. What is Kernel?
23. What is Resource Abstraction?

Part –C (3 x 10 = 30 Marks)
Answer All the Questions

24. a) Explain in detail about the Types of Operating Systems
OR
b) Explain about the Basic OS Functions
25. a) Explain in detail about Process Control with suitable diagram
OR
b) Describe about the Operating Systems for Personal Computers & Workstations
26. a) Write a program to execute parent and child class using fork() System call
OR
b) Explain in detail about System Calls and System Programs

PART-A (20 X 1 = 20 Marks)

Answer Keys

1. _____is a program that manages the computer hardware and acts as an intermediary between the computer user and the computer hardware.

   a)hardware acceleration b)**Operating System** c)compiler d)logical transcation

2. _____manages the execution of user programs to prevent errors and improper use of the computer

   a) **resource allocator** b)work station c)main frame d)control program

3. ____were the first computers used to tackle many commercial & scientific application.

   a)**Mainframe computer system** b)Mainframe computer service

   c)multiframe computer system d)multiframe computer service

4. _____contains the address of an instruction to be fetched from memory

a) **Program counter (PC)**b)Instruction register (IR) c)Control registers d)Status registers

5. _____ is also known as parallel systems or tightly coupled systems)

   a)**Multiprocessor systems** b)desktop systems c)Time sharing systems d)Multiprogrammed systems

6. _____operating systems are even more complex than multi programmed operating systems.

   a)Time-sharing b)desktop systems c)Multiprogrammed systems d)**Multiprocessor systems**

7. _____ operating system keeps several jobs in memory simultaneously.

   a)**Time-sharing** b)desktop systemsc)Multiprogrammed systemsd)Multiprocessor systems

8. _____can save more money than multiple single-processor systems

   a)Multiprocessor systems b)desktop systems c)**Time sharing systems** d)Multiprogrammed systems

9. The most common multiple-processor systems now use

   a)symmetric multiprocessing b)asymmetric multiprocessing c) **multithreading** d)multiprogramming

10. Another form of a special-purpose operating system is the

    a)real-time systemb)distributed operating system c)Process states d)**multiframe computer system**

11. The assignment of the CPU to the first process on the ready list is called

    a)graceful degradation b)Time-sharing c)**dispatching** d)Multiprocessor systems

12. The manifestation of a process in an operating system is a

    a)Process state transitions b)process control block c)**child process** d)cooperating processes

13. For multiprogramming operating system

    a) **special support from processor is essential** b)special support from processor is not essential c)cache memory is essential d)cache memory is not essential

14. Which operating system reacts in the actual time

    a)Batch system b)Quick response system c)**Real time system** d)Time sharing system

15. The primary job of an OS is to _____

a)command resource b)**manage resource** c)provide utilities d)Be user friendly

16. The term " Operating System " means _____

a)**A set of programs which controls computer working** b)The way a computer operator works c)Conversion of high-level language in to machine level language d)The way a floppy disk drive operates

17. With .....more than one process can be running simultaneously each on a different processer.

a)Multiprogramming b)Uniprocessing c)**Multiprocessing** d)Uniprogramming

18. The two central themes of modern operating system are

a)**Multiprogramming and Distributed processing** b)Multiprogramming and Central Processing c)Single Programming and Distributed processing d)None of above

19. …….. is a example of an operating system that support single user process and single thread

a)UNIX b)MS-DOS c)OS/2 d)**Windows 2000**

20. The operating system of a computer serves as a software interface between the user and the _____.

A) s**Hardware** b)Peripheral c)Memory d)Screen

Part –B (3 x 2 = 6 Marks)

Answer All the Questions

21. What is Operating Syste?
22. What is Kernel?
23. What is Resource Abstraction?

Part –C (3 x 8 = 24 Marks)
Answer All the Questions

24.a) Explain in detail about the Types of Operating Systems

OR

b) Explain about the Basic OS Functions

25. a) Explain in detail about Process Control with suitable diagram

OR

b) Describe about the Operating Systems for Personal Computers & Workstations

26. a) Write a program to execute parent and child class using fork() System call.

OR

b) Explain in detail about System Calls and System Programs.

Part –B (3 x 2 = 6 Marks)

Answer Key for 2 marks

21. What is Operating System?

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.

A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and the users. The hardware provides the basic computing resources for the system. The application programs define the ways in which these resources are used to solve users' computing problems.

22. What is Kernel?

The kernel is the central module of an operating system (OS). It is the part of the operating system that loads first, and it remains in main memory. Because it stays in memory, it is important for the kernel to be as small as possible while still providing all the essential services required by other parts of the operating system and applications. The kernel code is usually loaded into a protected area of memory to prevent it from being overwritten by programs or other parts of the operating system.

23. What is Resource Abstraction?

Resource abstraction is the process of "hiding the details of how the hardware operates, Thereby making computer hardware relatively easy for an application programmer to use. One way in which the operating system might implement resource abstraction is to provide a single abstract disk interface which will be the same for both the hard disk and floppy disk. Such an abstraction saves the programmer from needing to learn the details of both hardware interfaces. Instead, the programmer only needs to learn the disk abstraction provided by the operating system.

Part –C (3 x 10 = 30 Marks)

Answer Key

24. A) Explain in detail about the Types of Operating Systems

TYPES OF OPERATING SYSTEM

- Types of operating system which are commonly used

MULTI-PROGRAMMING SYSTEM

- The work of the server is to execute the job in sequence assigned by the users at their fair intervals. This is the first time the OS are programmed (Control Program or Handler) to handle the users with the required resources. The switching between the users and the allocation of same resources to multiple processes was the difficult task.
- Multiprogramming is a rudimentary form of parallel processing in which several programs are run at the same time on a uniprocessor. Since there is only one processor, there can be no true simultaneous execution of different programs.

BATCH OPERATING SYSTEM

- The tasks are grouped as batch based on the priority specified by the user. Once the tasks are grouped they are executed as a batch by the machine. The duration of execution may be a week or even months. The tasks are grouped manually by a person and after proper execution the results are given to them by that person. Lack of interaction between the user and the job.

TIME-SHARING OPERATING SYSTEMS

- Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time..

- 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.

- Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently.

- That is, if n users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

Advantages of Timesharing operating systems are as follows −

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.
- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

REAL TIME OPERATING 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 system

- 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 likes undersea exploration and planetary rovers, etc.

DISTRIBUTED OPERATING SYSTEM

- Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.
- The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as loosely coupled systems or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.
- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing

Client-Server Computing: As PCs have become faster, more powerful, and cheaper, designers have shifted away from centralized system architecture.

Server systems can be broadly categorized as compute servers and file servers:

• The compute-server system provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client. A server running a database that responds to client requests for data is an example of such a system.

• The file-server system provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers.

Client-Server Model

Peer to peer Systems

Another structure for a distributed system is the peer-to-peer (P2P) system model. In this model, clients and servers are not distinguished from one another. Instead, all nodes within the system

are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service.

- When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service. The remainder of the communication takes place between the client and the service provider.
- An alternative scheme uses no centralized lookup service. Instead, a peer acting as a client must discover what node provides a desired service by broadcasting a request for the service to all other nodes in the network.

Network operating System

- A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.
- Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

24. (b) what is Basic OS function?

BASIC OS FUNCTION

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 user

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. 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 terminate

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

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

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.

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

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

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.

Other Important Activities

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

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

The operating system is the core software component of your computer. It performs many functions and is, in very basic terms, an interface between your computer and the outside world. In the section about hardware, a computer is described as consisting of several component parts including your monitor, keyboard, mouse, and other parts. The operating system provides an interface to these parts using what is referred to as "drivers". This is why sometimes when you install a new printer or other piece of hardware, your system will ask you to install more software called a driver.

An operating system has three main functions: (1) manage the computer's resources, such as the central processing unit, memory, disk drives, and printers, (2) establish a user interface, and (3) execute and provide services for applications software.

- System tools (programs) used to monitor computer performance, debug problems, or maintain parts of the system.

- A set of libraries or functions which programs may use to perform specific tasks especially relating to interfacing with computer system components.
- The operating system makes these interfacing functions along with its other functions operate smoothly and these functions are mostly transparent to the user.
- The operating system underpins the entire operation of the modern computer.

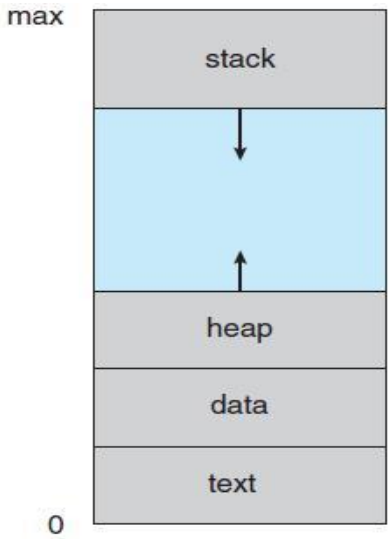
25. a) Explain in detail about Process Control with suitable diagram

Process Concept

Process is a program that is in execution. It is defined as unit of work in modern systems. A batch system executes jobs, whereas a time-shared system has user programs, or tasks. Even on a single-user system, a user may be able to run several programs at one time: a word processor, a Web browser, and an e-mail package. And even if a user can execute only one program at a time, such as on an embedded device that does not support multitasking, the operating system may need to support its own internal programmed activities, such as memory management. In many respects, all these activities are similar, so we call all of them processes.

Process in memory

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.

A program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file). In contrast, 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.

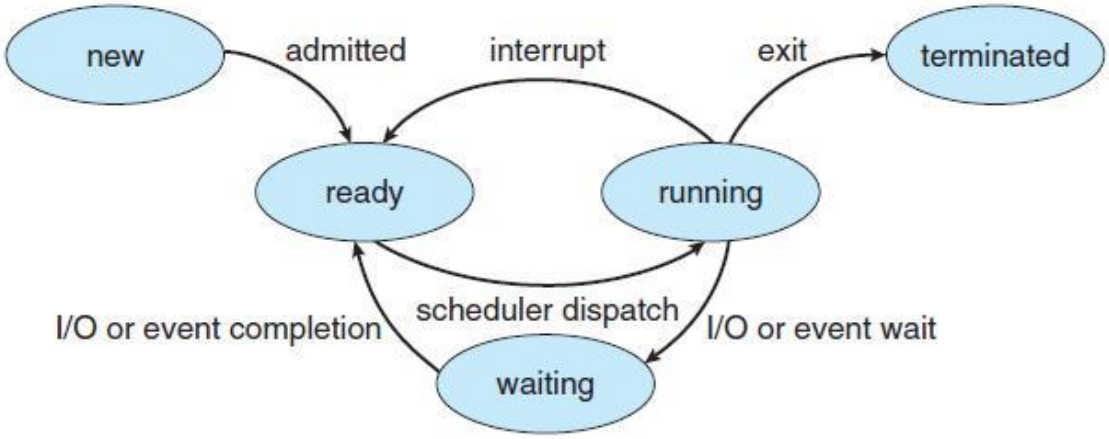
• Running. Instructions are being executed.

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

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

• 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 the following Figure.



Process State Diagram

Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

• New. The process is being created.

Process Control Block (PCB)

Each process is represented in the operating system by a process control block (PCB)—also called a task control block. It contains many pieces of information associated with a specific process, including these: Process state. The state may be new, ready, running, and waiting, halted, and so on.

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

- **CPU registers.** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-Purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

- **CPU-scheduling information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Process Control Block (PCB)

- **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system

- **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

- **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

25. b) Describe about the Operating Systems for Personal Computers & Workstations

Personal computer and workstation operating systems. The emphasis is placed on UNIX, MS DOS, MS windows and OS/2 operating systems. UNIX is cover under the U.S. Government POSIX standard, which dictates its use when practical. MS DOS is the most used operating system worldwide. OS/2 was developed to combat some of the shortcomings of MS DOS. Each operating system which is discussed has a design philosophy that fulfills specific user's needs. UNIX was designed for many users sharing a computer system. MS DOS, MS Windows and OS/2 are designed as single user computer systems. All of these operating systems are in use at the Naval Postgraduate School. All of the operating systems are discussed with regard to their: history of development, process management, file system, input and output system, user interface, network capabilities, and advantages and disadvantages. UNIX has a section devoted to the POSIX standard and MS DOS has a section devoted to Windows 3. 1. Apple Corporation's System 7 is mentioned throughout the text, but is not covered in detail. Personal Computer and Workstation Operating Systems

Computer can be broadly classified by their speed and computing power. Sr. No. Type Specifications

1. **PC (Personal Computer)** Single user computer system. Moderately powerful microprocessor.
2. **WorkStation** Single user computer system. Similar to Personal Computer but have more powerful microprocessor.
3. **Mini Computer** Multi-user computer system. Capable of supporting hundreds of users simulaneously.

4 **Main Frame** Multi-user computer system. Capable of supporting hundreds of users simulaneously.Software technology is different from minicomputer.

5. **Supercomputer** An extremely fast computer which can perform hundreds of millions of instructions per second.

PC (Personal Computer) A PC can be defined as a small, relatively inexpensive computer designed for an individual user. PCs are based on the microprocessor technology that enables manufacturers to put an entire CPU on one chip. Businesses use personal computers for word processing, accounting, desktop publishing, and for running spreadsheet and database management applications. At home, the most popular use for personal computers is for playing games and surfing the Internet. Although personal computers are designed as single-user systems, these systems are normally linked together to form a network. In terms of power, now-a-days High-end models of the Macintosh and PC offer the same computing power and graphics capability as low-end workstations by Sun Microsystems, Hewlett-Packard, and DELL.

WORKSTATION Workstation is a computer used for engineering applications (CAD/CAM), desktop publishing, software development, and other such types of applications which require a moderate amount of computing power and relatively high quality graphics capabilities

Workstations generally come with a large, high-resolution graphics screen, large amount of RAM, inbuilt network support, and a graphical user interface. Most workstations also have a mass storage device such as a disk drive, but a special type of workstation, called a diskless workstation, comes without a disk drive. Common operating systems for workstations are UNIX and Windows NT. Like PC, Workstations are also single-user computers. However, workstations are typically linked together to form a local-area network, although they can also be used as stand-alone systems.

26. a) Write a program to execute parent and child class using fork() System call.

Child class usingfork() System Call

A Process can create a new child process using fork () system call.

This new child process created through fork() call will have same memory image as of parent process i.e. it will be duplicate of calling process but will have different process ID.

For example,

Suppose there is a Process “Sample” with Process ID 1256 and parent ID 12. Now as soon as this process calls the fork() function, a new process will be created with same memory image but with different process ID.

Also, process which has called this fork() function will become the parent process of this new process i.e.

Process 1: *Sample (pid=1341 | Parent Process ID = 12)*
After calling fork() system call,

Process 1: *Sample (pid=1341 | Parent Process ID = 12)*
Process 2: *Sample (pid= 4567 | Parent Process ID = 1341)*
As memory image of new child process will be the copy of parent process’s memory image. So, all variables defined before fork() call will be available in child process with same values.

If fork() call is successful then code after this call will be executed in both the process. Therefore, fork() function’s return value will be different in both the process’s i.e.
If fork() call is successful then it will,

- Return 0 in child process.
- Return process id of new child process in parent process.

If fork() call is unsuccessful then it will return -1.

```
1  #include <iostream>
2  #include <unistd.h>
3
4  int main()
5  {
6          int x = 6;
7
8          pid_t childProcessId = fork();
9
10         // If fork call Code after
11         if(childProcessId < 0)
12         {
13                 std::cout<<"Failed to Create a new Process"<<std::endl;
14         }
15         else if (childProcessId == 0)
16         {
17                 // This code will be executed in Child Process Only
18                 std::cout<<"Child Process :: x = "<<x<<std::endl;
19                 x = 10;
20                 std::cout<<"Child Process :: x = "<<x<<std::endl;
```

```
21              std::cout<<"Child Process exists"<<std::endl;
22      }
23      else if (childProcessId > 0)
24      {
25              // This code will be executed in Parent Process Only
26              sleep(2);
27              std::cout<<"Parent Process :: x = "<<x<<std::endl;
28
29      }
30
31 }
```

OUTPUT

Child Process:: x = 6
Child Process:: x = 10
Child Process exists
Parent Process :: x = 6

As we can see value of x was 6 before calling fork() function. Therefore in child process value of x remain 6 but then child process modified the value of x to 10. But this change will not be reflected in parent process because parent process has seperate copy of the variable and its value remain same i.e. 6. We added sleep in parent process because to add a delay of 2 seconds and check the value of x in parent process after child process exists.

After fork() call finishes both child and parent process will run parallelly and execute the code below fork() call simultaneously.

26. (b) Explain in detail about System Calls and System Programs.

SYSTEM CALLS AND SYSTEM PROGRAMS

- 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 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.

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

SYSTEM CALL

When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a **system call**.

When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **context switch**.

Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Generally, system calls are made by the user level programs in the following situations:

- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.

- Requesting access to a hardware device, like a mouse or a printer.

SYSTEM PROGRAMS

These programs are not usually part of the OS kernel, but are part of the overall operating system.

File Management

These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.

Status Information

Some programs simply request the date and time, and other simple requests. Others provide detailed performance, logging, and debugging information. The output of these files is often sent to a terminal window or GUI window

File modification

Programs such as text editors are used to create, and modify files.

Communications

These programs provide the mechanism for creating a virtual connect among processes, users, and other computers. Email and web browsers are a couple examples.

II BCA
28-8-2019 AN
Extra

Reg.No_____
[18CAU302]

KARPAGAM ACADEMY OF HIGHER EDUCATION
KARPAGAM UNIVERSITY
(Established Under section 3 of UGC Act 1956)
Coimbatore – 641 021
BCA DEGREE EXAMINATION
(For the candidates admitted from 2018 onwards)
Third Semester
Second Internal Exam August 2019
OPERATING SYSTEMS

Time: 2 Hours **Maximum: 50 marks**
Date&Session: 28-8-2019 & AN **CLASS: II BCA(A&B)**

PART A-[20 X 1 = 20 Marks]
Answer all the questions

1. When a process is to be run, its code and data are brought to the?
   a) Main storage b) Auxiliary storage
   c) Secondary storage d) Backing storage
2. A scheduling discipline is said to be _____, if the CPU can be taken away in between.
   a) Non preemptive b) Preemptive c) Dedicated d) None
3. Which scheduling is effective in time sharing environments
   a) FIFO scheduling b) RR scheduling c) SJF scheduling d) SRT scheduling
4. Block state transition is
   a) Running □ Ready b) Ready □ Running
   c) Running □ Blocked d) Blocked □ Ready
5. Process is said to be blocked if
   a) It is the first process in ready list b) It is waiting for some event to happen
   c) It currently has CPU d) It could use a CPU if available
6. Two basic types of operating systems are _____
   a) Sequential and direct b) batch and time share
   c) Sequential and real time d) batch and interactive
7. A Semaphore is a _____ variable whose value can be accessed and altered only by the operations P & V
   a) Predefined b) Protected c)User defined d) Built-in
8. Process states are _____
   a) Submit, Ready, Block b)Submit, Run, Ready
   c) Ready, Run, Block d) Ready,Submit,Block
9. Which access a process should have to modify any information in a segment?
   a) Write b) Read c)Execute d) Append
10. Critical section can also be called
    a) Critical region b) Mutual exclusion c) Parallelism d) Critical part
11. The processors communicate with one another through various communication lines, such as High-speed buses or telephone lines
    a) Distributed b) Time sharing c) distributed systems d) Virtual storage
12. Variable size blocks are called _____
    a) Pages b) Segments c) Tables d) Region

22/07/19
28/08/19.
14/10/19

13. Using Priority Scheduling algorithm, find the average waiting time for the following set of processes given with their priorities in the order: Process : Burst Time : Priority respectively P1 : 10 : 3 , P2 : 1 : 1 , P3 : 2 : 4 ,P4 : 1 : 5 , P5 : 5 : 2.
    a) 8 milliseconds b) 8.2 milliseconds c) 7.75 milliseconds d) 3 milliseconds
14. The _____ algorithm selects the request with the minimum seek time from the current head position.
    a) FCFS b) SSTF c) SCAN d) C-SCAN
15. An optimal scheduling algorithm in terms of minimizing the average waiting time of a given set of processes is _____
    a) FCFS scheduling algorithm b) Round robin scheduling algorithm
    c) Shortest job - first scheduling algorithm d) LRU
16. The Hardware mechanism that enables a device to notify the CPU is called _____.
    a) Polling b) Interrupt c) System Call d) Invoke
17. The SJF algorithm is a special case of the general _____
    a) FIFO scheduling b) priority scheduling algorithm c) RR scheduling
    d) SRT scheduling
18. A scheduling discipline is said to be _____, if the CPU can be taken away in between.
    a) Non preemptive b) Preemptive c) Dedicated d) None
19. If a time quantum used in round-robin preemptive CPU scheduling is allowed to grow too large, the algorithm degenerates to:
    a) Shortest- job-First(SJF) b)Shortest-Time-Remaining(STR)
    c) Job Exclusion (JBE). d) First-Come-First-Served (FIFS).
20. The collection of processes on the disk that are waiting to be brought into memory for execution, forms
    a) Queue b) List c) Input queue d) Output queue

8

PART – B [3x 2 = 6 Marks]
Answer ALL the questions

21. What is System Call?
22. What is Kernel?
23. Define Semaphore.

PART – C [3x 8= 24 Marks]
Answer ALL the questions

24. a. Discuss Process states with neat diagram
    (or)
    b. Explain about process scheduling in detail

25. a. Explain about Virtual address space.
    (or)
    b. Explain about Deadlock and its process

26. a. Explain in detail about Preemptive and Non preemptive Scheduling Algorithms with suitable examples (or)
    b. Explain the methods for inter-process communication in detail

PART-A-[20X1=20 MARKS

ANSWER KEY

1. When a process to be run, its code and data are brought to the?

Answer**: Main storage**

2. A scheduling discipline is said to be_____if the CPU can be taken away between

Answer:**Preemptive**

3. Which scheduling is effective in time sharing environments?

Answer: **RR Scheduling**

4. Block state transition is

Answer: **Running Blocked**

5. Process is sait to be blocked if

Answer: **Its waiting for some event to be happen**

6. Two basic types of operating systems are_____

Answer: **Batch and Interactive**

7. A Semaphore is a_____variable whose value can be accessed and altered only by operations P & V

Answer: **Protected**

8. Process states are_____

Answer: **Ready,Run,Block**

9. Which access a process should have to modify any information in a segment?

Answer: **Read**

10. Critical section can also be called____

Answer: **Mutual Exclusion**

11. The processors communicate with one another through various communication lines,such as High-speed buses or telephone lines

Answer:**Distributed System**

12.Variable size blocks are called_____

Answer: **Segments**

13.Using priority scheduling algorithm,find the average waiting time for the following set of processes given with their priorities in the order :Process:Burst time:Priority respectively p1 : 10 :3,p2 : 1 : 1,p3 : 2 : 4.p4 : 1 : 5 ,p5 : 5 : 2.

Answer: **8.2 milliseconds**

14. The_____algorithm selects the request with the minimum seek time for the current head position.

Answer: **SSTF**

15. An optimal scheduling algorithm in terms of minimizing the average waiting time of the given set of processes is_____.

Answer: **Shortest job-first scheduling algorithm**

16. The hardware mechanism that enables a device to notify the CPU is called_____.

Answer: **Interrupt**

17. The SJF algorithm is a special case of the general _____

Answer: **Priority Scheduling Algorithm**

18. A Scheduling discipline is said to be_____if the CPU can be taken away in between.

Answer: **Preemptive**

19. If a time quantum used in round –robin preemptive CPU scheduling is allowed to grow too large, the algorithm degenerates to:

Answer: **First come First Serve**

20. The collection of process on the disk that are waiting to be brought into memory for execution, forms

Answer: **Input Queue**

Part-B[3x2=6 Marks]

21.What is 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.

22. WHAT IS KERNAL?

The kernel is the central module of an operating system (OS). It is the part of the operating system that loads first, and it remains in main memory. Because it stays in memory, it is important for the kernel to be as small as possible while still providing all the essential services required by other parts of the operating system and applications. The kernel code is usually loaded into a protected area of memory to prevent it from being overwritten by programs or other parts of the operating system.

23. DEFINE SEMAPHORE.

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait() and signal(). The wait() operation was originally termed P (from the Dutch proberen, ―to test‖); signal() was originally called V (from verhogen, ―to increment‖).

24.a.Discuss process stated with neat Diagram

Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

• New. The process is being created.

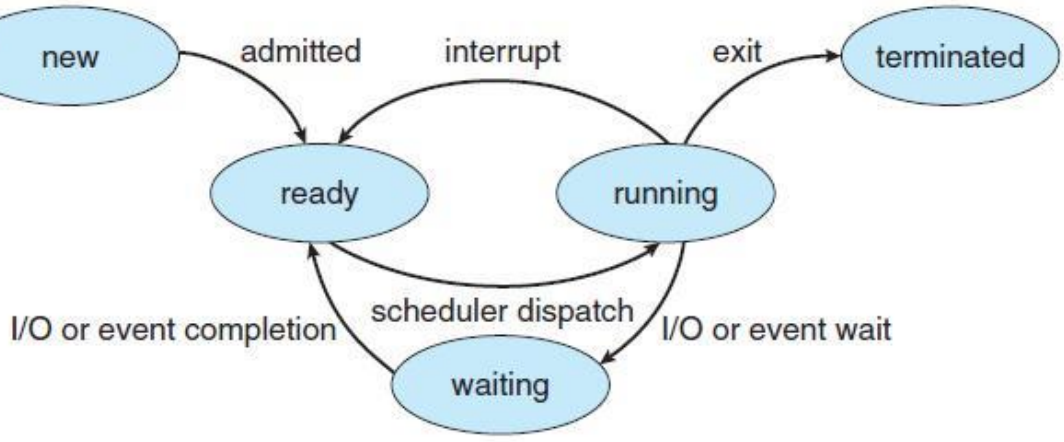
• Running. Instructions are being executed.

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

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

• 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 the following Figure.



Process State Diagram

24. B.Explain about process scheduling in detail

Definition

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

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

Scheduling Queues

- Scheduling queues refers to queues of processes or devices. When the process enters into the system, then this process is put into a job queue. This queue consists of all processes in the system. The operating system also maintains other queues such as device queue. Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has its own device queue.

This figure shows the queuing diagram of process scheduling.

- Queue is represented by rectangular box.
- The circles represent the resources that serve the queues.
- The arrows indicate the process flow in the system.

Queues are of two types

- Ready queue

- Device queue

A newly arrived process is put in the ready queue. Processes waits in ready queue for allocating the CPU. Once the CPU is assigned to a process, then that process will execute. While executing the process, any one of the following events can occur.

- The process could issue an I/O request and then it would be placed in an I/O queue.
- The process could create new sub process and will wait for its termination.

 The process could be removed forcibly from the CPU, as a result of interrupt and put back in the ready queue.

Two State Process Mode

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

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

Schedulers

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

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

Long Term Scheduler

It is also called job scheduler. Long term scheduler determines which programs are admitted to the system for processing. Job scheduler selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling. The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

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

Short Term Scheduler

- It is also called CPU scheduler. Main objective is increasing system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects process among the processes that are ready to execute and allocates CPU to one of them.

- Short term scheduler also known as dispatcher, execute most frequently and makes the fine grained decision of which process to execute next. Short term scheduler is faster than long term scheduler.

Medium Term Scheduler

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

SCHEDULING (PREEMPTIVE AND NONPREEMPTIVE)

Non preemptive Scheduling

- A scheduling discipline is non preemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process.

Following are some characteristics of non preemptive scheduling

1. In non preemptive system, short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
2. In non preemptive system, response times are more predictable because incoming high priority jobs cannot displace waiting jobs.
3. In non preemptive scheduling, a scheduler executes jobs in the following two situations.
    a. When a process switches from running state to the waiting state.
    b. When a process terminates.

Preemptive Scheduling

- A scheduling discipline is preemptive if, once a process has been given the CPU can taken away.

- The strategy of allowing processes that are logically runable to be temporarily suspended is called Preemptive Scheduling and it is contrast to the "run to completion" method.

  Schedule:

  A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter

  - First-Come, First-Served (FCFS) Scheduling
  - Shortest-Job-Next (SJN) Scheduling
  - Priority Scheduling
  - Shortest Remaining Time
  - Round Robin(RR) Scheduling
  - Multiple-Level Queues Scheduling

- These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

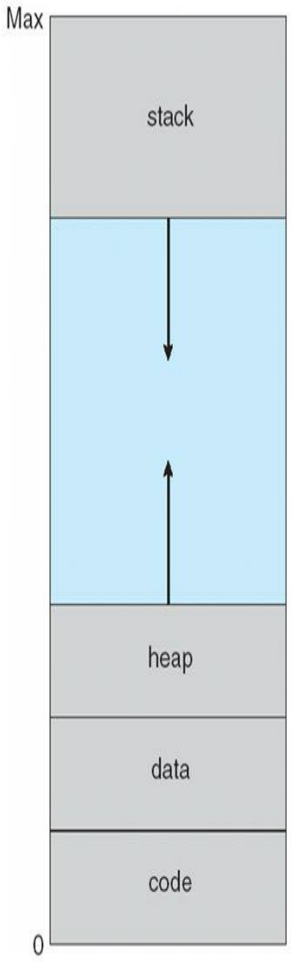
25. A.Explain about Virtual address space.

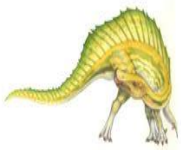
We have already discussed how the logical address are referenced by the program as it runs, but the Memory Management Unit calculates the physical address before loading the address on to bus. With a virtual memory management system, the logical address space becomes a virtual address space that uses the full range of memory addresses for each process (0 to 4 GB for a 32 bit system).

The virtual memory address space for each process is quite **sparse**. Recall what the contiguous address space for a process looks like. If address space is sparse, then the heap is quite large, which mean that the stack is unlikely to grow into other memory. A large heap leaves plenty of room for dynamically allocated memory and also for the memory addresses of shared libraries (called DLLs in Windows).

Virtual-address Space

- Usually design logical address space for stack to start at Max logical address and grow "down" while heap grows "up"
  - Maximizes address space use
  - Unused address space between the two is hole
    - No physical memory needed until heap or stack grows to a given new page
- Enables **sparse** address spaces with holes left for growth, dynamically linked libraries, etc
- System libraries shared via mapping into virtual address space
- Shared memory by mapping pages read-write into virtual address space
- Pages can be shared during `fork()`, speeding process creation





25. b .Explain about Deadlock and its process

DEADLOCK

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. Request. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.

2. Use. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).

3. Release. The process releases the resource.

- For each use of a kernel-managed resource by a process or thread, the operating system checks to make sure that the process has requested and has been allocated the resource. A system table records whether each resource is free or allocated. For each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.

- A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set. The events with which we are mainly concerned here are resource acquisition and release. The resources may be either physical resources (for example, printers, tape drives, memory space, and CPU cycles) or logical resources (for example, semaphores, mutex locks, and files).

Deadlock Prevention

Mutual Exclusion

- The mutual exclusion condition must hold. That is, at least one resource must be non-sharable. Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock. Read-only files are a good example of

a sharable resource. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file.

- A process never needs to wait for a sharable resource. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable. For example, a mutex lock cannot be simultaneously shared by several processes.

Hold and Wait

- To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.

- One protocol that we can use requires each process to request and be allocated all its resources before it begins execution. We can implement this provision by requiring that system calls requesting resources for a process precede all other system calls. An alternative protocol allows a process to request resources only when it has none.

- A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.

- Both these protocols have two main disadvantages. First, resource utilization may be low, since resources may be allocated but unused for a long period. In the example given, for instance, we can release the DVD drive and disk file, and then request the disk file and printer, only if we can be sure that our data will remain on the disk file. Otherwise, we must request all resources at the beginning for both protocols.

- Second, starvation is possible. A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

No Preemption

- The third necessary condition for deadlocks is that there be no preemption of resources that have already been allocated. To ensure that this condition does not hold, we can use the following protocol.
- If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources the process is currently holding are preempted. In other words, these resources are implicitly

released. The preempted resources are added to the list of resources for which the process is waiting.

- The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting. Alternatively, if a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process and allocate them to the requesting process. If the resources are neither available nor held by a waiting process, the requesting process must wait.

- While it is waiting, some of its resources may be preempted, but only if another process requests them. A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were preempted while it was waiting.

Circular Wait

- The fourth and final condition for deadlocks is the circular-wait condition. One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance:

- For avoiding deadlocks, it is to require additional information about how resources are to be requested. For example, in a system with one tape drive and one printer, the system might need to know that process P will request first the tape drive and then the printer before releasing both resources, whereas process Q will request first the printer and then the tape drive.

- With this knowledge of the complete sequence of requests and releases for each process, the system can decide for each request whether or not the process should wait in order to avoid a possible future deadlock. Each request requires that in making this decision the system consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

Safe State:

- A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence.

26.a.Explain in Detail About Preemptive and Non Preemptive Scheduling Algorithms with suitable examples

There are six popular process scheduling algorithms

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

- These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0= 0
P1	5 - 1= 4
P2	8 - 2= 6
P3	16 - 3= 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.

- The processer should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8

Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

Average Wait Time: (3+0+14+5) / 4 = 5.50

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.

- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0

Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

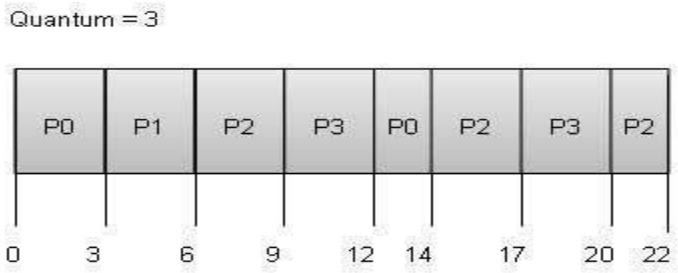
Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.

- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12

P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

26. a .Explain the methods for interprocess communication

METHODS OF INTER-PROCESS COMMUNICATION (IPC)

Inter-process communication using shared memory requires communicating processes to establish a region of shared memory. Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by these processes and are not under the operating system's control.

The processes are also responsible for ensuring that they are not writing to the same location simultaneously. Two types of buffers can be used. The unbounded buffer places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the

producer can always produce new items. The bounded buffer assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Message-Passing Systems

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. It is particularly useful in a

distributed environment, where the communicating processes may reside on different computers connected by a network. For example, an Internet chat program could be designed so that chat participants communicate with one another by exchanging messages. A message-passing facility provides at least two operations:

send(message) receive(message)

Messages sent by a process can be either fixed or variable in size. If only fixed-sized messages can be sent, the system-level implementation is straightforward. This restriction, however, makes the task of programming more difficult. Conversely, variable-sized messages require a more complex system common kind of tradeoff seen throughout operating-system design. If processes P and Q want to communicate, they must send messages to and receive messages from each other: a communication link must exist between them. This link can be implemented in a variety of ways. We are concerned here not with the link's physical implementation (such as shared memory, hardware bus, or network, but rather with its logical implementation. Here are several methods for logically implementing a link and the send()/receive() operations:

- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

Naming

Processes that want to communicate must have a way to refer to each other. They can use either direct or indirect communication. Under direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send() and receive() primitives are defined as:

• send(P, message)—Send a message to process P.

• receive(Q, message)—Receive a message from process Q.

A communication link in this scheme has the following properties:

• A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
• A link is associated with exactly two processes.

• Between each pair of processes, there exists exactly one link.

This scheme exhibits symmetry in addressing; that is, both the sender process and the receiver process must name the other to communicate. A variant of this scheme employs asymmetry in addressing. Here, only the sender names the recipient; the recipient is not required to name the sender. In this scheme, the send () and receive () primitives are defined as follows:

• send (P, message)—Send a message to process P.

• receive (id, message)—Receive a message from any process. The variable id is set to the name of the process with which communication has taken place.

The disadvantage in both of these schemes (symmetric and asymmetric) is the limited modularity of the resulting process definitions. Changing the identifier of a process may necessitate examining all other process definitions. All references to the old identifier must be found, so that they can be modified to the new identifier. In general, any such hard-coding techniques, where identifiers must be explicitly stated, are less desirable than techniques involving indirection.

With indirect communication, the messages are sent to and received from mailboxes, or ports. A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed. Each mailbox has a unique identification. For example, POSIX message queues use an integer value to identify a mailbox. A process can communicate with another process via a number of different mailboxes, but two processes can communicate only if they have a shared mailbox. The send () and receive () primitives are defined as follows:

• send (A, message)—Send a message to mailbox A.

• receive (A, message)—Receive a message from mailbox A.

In this scheme, a communication link has the following properties:

• A link is established between a pair of processes only if both members of the pair have a shared mailbox.

Synchronization

Communication between processes takes place through calls to send() and receive() primitives. There are different design options for implementing each primitive. Message passing may be either blocking or non blocking— also known as synchronous and asynchronous. (Throughout this text, you will encounter the concepts of synchronous and asynchronous behavior in relation to various operating-system algorithms.)

• Blocking send. The sending process is blocked until the message is received by the receiving process or by the mailbox.

• Non-blocking send. The sending process sends the message and resumes operation.

• Blocking receive. The receiver blocks until a message is available.

• Non-blocking receive. The receiver retrieves either a valid message or a null.

Different combinations of send () and receive () are possible. When both send () and receive () are blocking, we have a rendezvous between the sender and the receiver. The solution to the producer–consumer problem becomes trivial when we use blocking send () and receive () statements. The producer merely invokes the blocking send () call and waits until the message is delivered to either the receiver or the mailbox. Likewise, when the consumer invokes receive (), it blocks until a message is available.

Buffering

Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue. Basically, such queues can be implemented in three ways:

• Zero capacity. The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.

• Bounded capacity. The queue has finite length n; thus, at most n messages can reside in it. If

the queue is not full when a new message is sent, the message is placed in the queue (either the message is copied or a pointer to the message is kept), and the sender can continue execution without waiting. The link's capacity is finite, however. If the link is full, the sender must block until space is available in the queue.

Reg.No_____

[18CAU302]

KARPAGAM ACADEMY OF HIGHER EDUCATION

KARPAGAM UNIVERSITY

(Established Under section 3 of UGC Act 1956)

Coimbatore – 641 021

BCA DEGREE EXAMINATION

(For the candidates admitted from 2018 onwards)

Third Semester

Third Internal Exam October 2019

OPERATING SYSTEMS

Time: 2 Hours **Maximum: 50 marks**

Date & Session: 14-10-2019 & AN **CLASS: II BCA (A&B)**

PART A-[20 X 1 = 20 Marks]

Answer all the questions

1. Expand UFD
   a) user file directory b) uniform file directory
   c) user filter directory d) uniform filter directory
2. MS-DOS environment provided by a _____ application is called a VDM.
   a) win45 b) win60 c) win32 d) win16
3. Linux is a _____ like system that has gained popularity in recent years.
   a) Windows 2000 b) windows xp
   c) unix d) windows server
4. The _____ contains all the internal worker threads and never executes in user mode
   a) boot device b) system restore c) system process d) system hive
5. _____ is used for both anonymous and authenticated access
   a) http b) ftp c) smtp d) fdp
6. The _____ program job is to bring in a full bootstrap program from disk.
   a) bootstrap loader b) boot loader
   c) boot block d) system block
7. Which of the following are forms of malicious attack?
   a) Theft of information b) Non intrusion c) Non denial of attack d) Virus
8. _____ access control is an approach to restricting system access to authorized users.
   a) Role-based b) Process-based c) Job-based d) Task-based
9. _____ is the ability to move from one hardware architecture to another with relatively few changes.
   a) Portability b) Reliability c) Extensibility d) compatibility
10. The code segment that misuses its environment is called a
    a) Internal thief b) Trojan horse c) Code stacker d) Thief
11. The pattern that can be used to identify a virus is known as
    a) Stealth b) Virus signature c) Armoured d) Multipartite
12. The authentication method that measures the physical characteristics of the user that are hard to forge is called as _____
    a) Biometrics b) Password c) Stegnography d) Access control
13. Any malware hidden in software or a Web page that people voluntarily download is called _____
    a) Worm b) Trojan Horse c) Virus d) Backdoor
14. What are common security threats?
    a) File Shredding b) File sharing and permission c) File corrupting
    d) File integrity
15. A _____ is a group of related records
    a) Field b) record c) file d) character
16. The operation of building a new file is called _____
    a) Open b) close c) create d) destroy
17. Which malware is hidden in software / Web page that user voluntarily to download is called _____
    a) Worm b) Virus c) Trojan Horse d) Backdoor
18. The most widely used form of authentication is to require the user to type a _____
    a) PIN number b) Login name. c) PIN number d) Username
19. What are characteristics of Authorization?
    a) RADIUS and RSA b) 3 way handshaking with syn and fin.
    c) Multilayered protection for securing resources
    d) Deals with privileges and rights
20. Which of the following is least secure method of authentication?
    a) Key card b) Fingerprint c) Retina pattern d) Password
    a) Worm b) Trojan c) Threat d) Virus

PART – B [3x 2 = 6 Marks]

Answer ALL the questions

21. What are the file allocation methods?
22. What is Device management?
23. What is internal access authorization?

PART – C [3x 8= 24 Marks]

Answer ALL the questions

24. a. Explain in detail about directory structure

    (or)

    b. Explain in detail about contiguous file allocation method.

25. a. Explain in detail about Linked file allocation method.

    (or)

    b. Explain in detail about Authentication.

26. a. Explain in detail about Policy mechanism..

    (or)

    b. Discuss briefly about Internal access authorization.

1. Expand UDF

Answer: **User file directory**

2. MS-DOS environment provided by a_____application is called by a VDM.

Answer: **win32**

3. Linux is a_____like system that has gained popularity in recent years.

Answer: **UNIX**

4. The _____contains all the internal worker threads and never executes in user mode.

Answer: **System Process**

5._____is used for both anonymous and authenticated access

Answer: **ftp**

6. The _____program is to bring in a full bootstrap program from disk

Answer: **Bootstrap loader**

7. Which of the following are forms of malicious attack

Answer: **Theft of information**

8. _____access control is an approach to restricting system access to authorized user

Answer: **Role-based**

9. _____is the ability to move from one hardware architecture to another with relatively few changes

Answer: **Portability**

10. The code segment that misuses its environment is called

Answer: **Trojan horse**

11. The pattern that can be used to identify a virus is known as

Answer: **Virus Signature**

12. The authentication method that measures the physical characteristics of the user that are to forge is called as_____

Answer: **Biometrics**

13. Any maleware hidden in software or a webpage that people voluntarily download is Called
    Answer: **Trojan Horse**
14. what are the common security threats

    Answer: **File sharing and permission**

15. A_____ is a group of related records

    ANSWER: **File**

16. The operation of building a new file is called _____

    Answer: **Create**

17. Which maleware is hidden in software/Web page that user voluntarily to download is called _____
    Answer: **Trojan Horse**
18. The most widely used form of authenticationbis to required the user to type a _____

    Answer: **Login name**

19. What are Characteristics of Authorization?

    Answer: **Deals with Privilages and rights**

20. Which of the following is least secure method of authentication?

    Answer: **Password**

21. What are the file allocation methods?

FILE ALLOCATION METHODS

There are three major methods of storing files on disks:

1. Contiguous.
2. Linked.
3. Indexed.

22. What is device management?

DISK MANAGEMENT

Disk Formatting

Before a disk can be used, it has to be low-level formatted, which means laying down all of the headers and trailers marking the beginning and ends of each sector. Included in the header and trailer are the linear sector numbers, and error-correcting codes, ECC, which allow damaged sectors to not only be detected, but in many cases for the damaged data to be recovered (depending on the extent of the damage.) Sector sizes are traditionally 512 bytes, but may be larger, particularly in larger drives

Boot Block

Computer ROM contains a bootstrap program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it. (The ROM bootstrap program may look in floppy and/or CD drives before accessing the hard drive, and is smart enough to recognize whether it has found valid boot code or not.).

23. What is internal access authorization?

INTERNAL ACCESS AUTHORIZATION

Authorization:

- Goal: determine which principals can perform which operations on which objects.
- Logically, authorization information represented as an access matrix:
- One row per principal.
- One column per object.
- Each entry indicates what that principle can do to that object.

- In practice a full access matrix would be too bulky, so it gets stored in one of two.
- Access Control Lists (ACLs): organize by columns.
- With each object, store information about which users are allowed to perform which operations.
- Most general form: list of <user, privilege> pairs.
- For simplicity, users can be organized into groups, with a single ACL for an entire group.
- ACLs can be very general (Windows) or simplified (Unix).
- UNIX: 9 bits per file:
- owner, group, anyone
- read, write, execute permissions for each of the above
- In addition, user "root" has all permissions for everything
- ACLs are simple and are used in almost all file systems.
- Capabilities: organize by rows.
- Example of a simple capability-based protection scheme: page tables.

24. a.Explain in detail about directory structure

DIRECTORY STRUCTURE

- 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. 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.

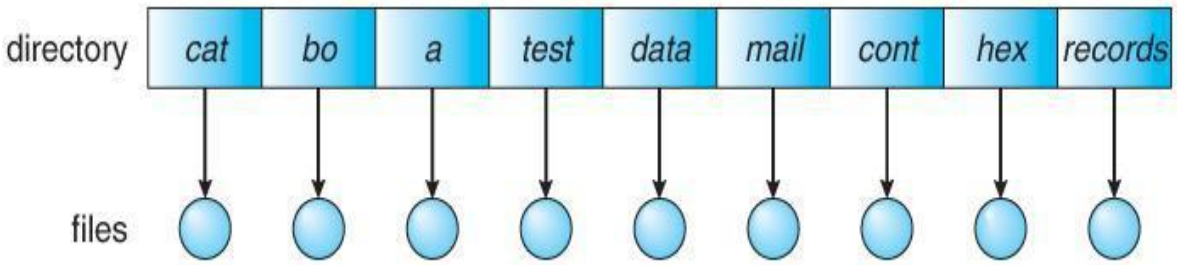
• **Create a file**. New files need to be created and added to the directory. • Delete a file. When a file is no longer needed, we want to be able to remove it from the directory.

• **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

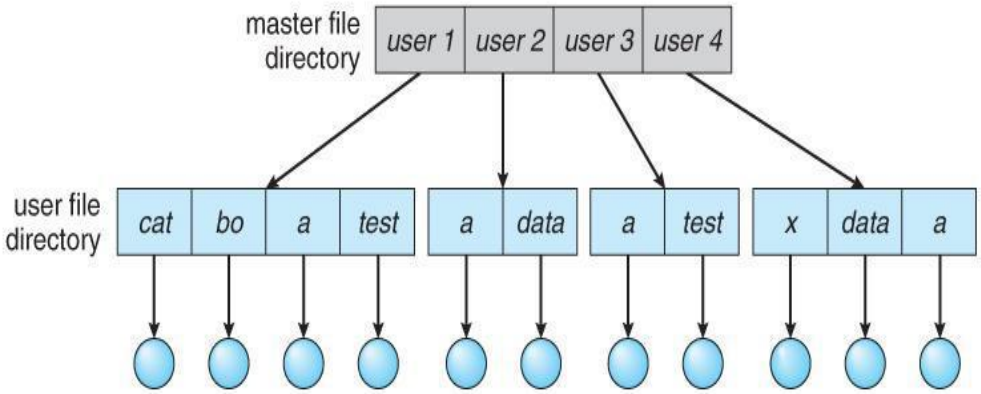
- longer in use, the file can be copied to tape and the disk space of that file released for reuse by another file.
-
- **Single-Level Directory**
-
- It is Simple to implement, but each file must have a unique name.



Two-Level Directory

- In this structure each user gets their own directory space. File names only need to be unique within a given user's directory. A master file directory is used to keep track of each user's directory, and must be maintained when users are added to or removed from the system.

- A separate directory is generally needed for system (executable) files. Systems may or may not allow users to access other directories besides their own If access to other directories is allowed, then provision must be made to specify the directory being accessed. If access is denied, then special consideration must be made for users to run programs located in system directories. A **search path** is the list of directories in which to search for executable programs, and can be set uniquely for each user.



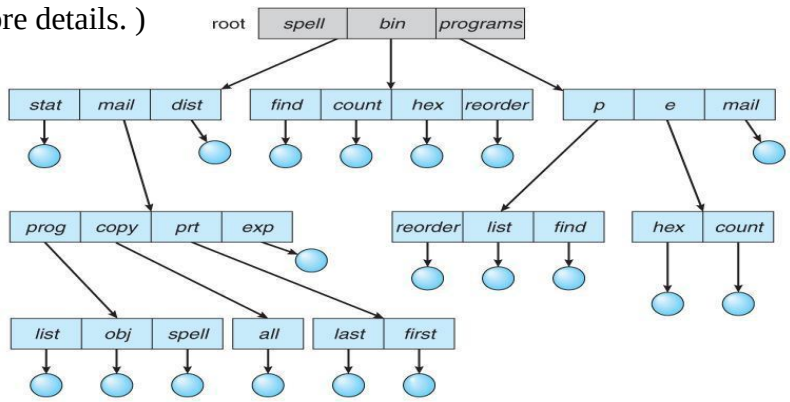
Tree-Structured Directories

- An obvious extension to the two-tiered directory structure, and the one with which we are all most familiar. Each user / process has the concept of a **current directory** from which all (relative) searches take place. Files may be accessed using either absolute pathnames (relative to the root of the tree) or relative pathnames (relative to the current directory.) Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands. One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.

Acyclic-Graph Directories

- When the same files need to be accessed in more than one place in the directory structure (e.g. because they are being shared by more than one user / process), it can be useful to provide an acyclic-graph structure. (Note the **directed** arcs from parent to child.)

  UNIX provides two types of **links** for implementing the acyclic-graph structure. (See "man ln" for more details.)

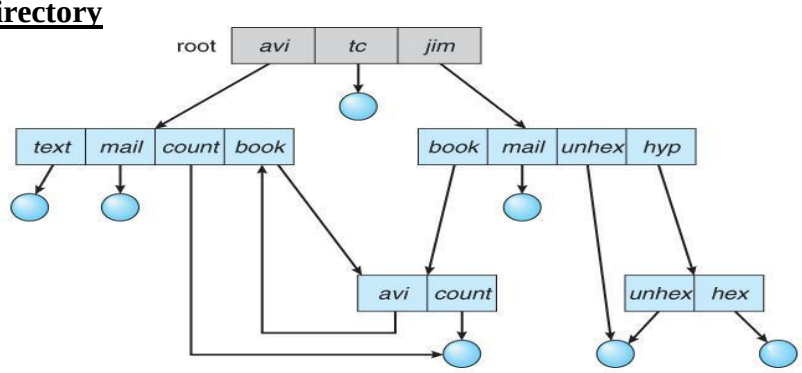


- A **hard link** (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same filesystem.

- A **symbolic link**, that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other file systems, as well as ordinary files in the current file system.

Windows only supports symbolic links, termed **shortcuts.** Hard links require a **reference count**, or **link count** for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.

General Graph Directory



- If cycles are allowed in the graphs, then several problems can arise: Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms. (Or not to follow symbolic links, and to only allow symbolic links to refer to directories.) Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero.

24.b.Explain about the Contiguous file allocation method.

Contiguous Allocation

- **Contiguous Allocation** requires that all blocks of a file be kept together contiguously. The performance is very fast, because reading successive blocks of the same file generally requires no movement of the disk heads, or at most one small step to the next adjacent cylinder.

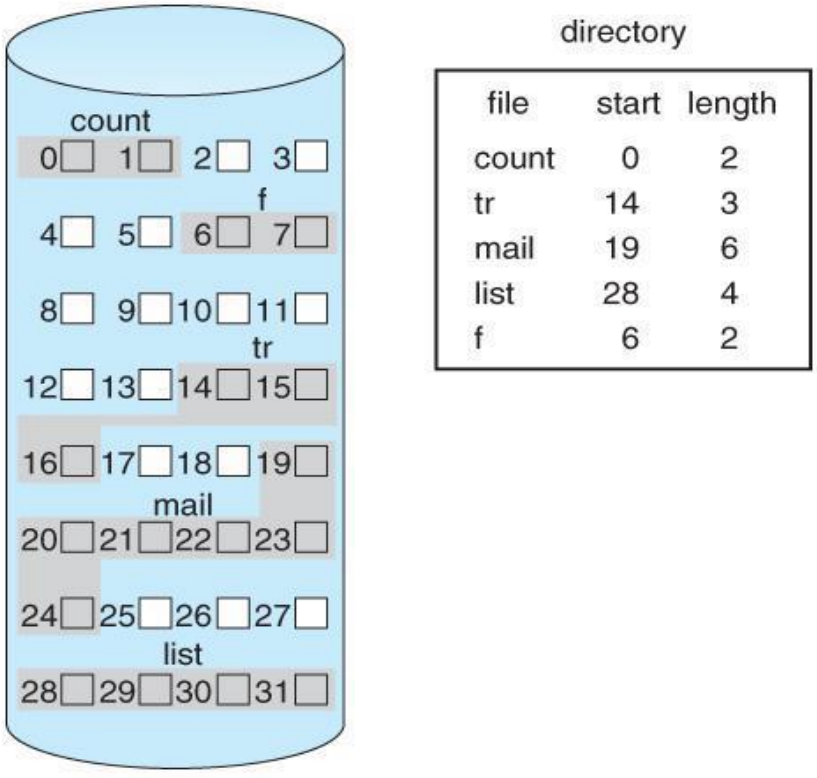
- Storage allocation involves the same issues discussed earlier for the allocation of contiguous blocks of memory (first fit, best fit, fragmentation problems, etc.) The

distinction is that the high time penalty required for moving the disk heads from spot to spot may now justify the benefits of keeping files contiguously when possible.

Problems can arise when files grow, or if the exact size of a file is unknown at creation time:

- Over-estimation of the file's final size increases external fragmentation and wastes disk space.
- Under-estimation may require that a file be moved or a process aborted if the file grows beyond its originally allocated space.
- If a file grows slowly over a long time period and the total final space must be allocated initially, then a lot of space becomes unusable before the file fills the space.

A variation is to allocate file space in large contiguous chunks, called **extents.** When a file outgrows its original extent, then an additional one is allocated. (For example an extent may be the size of a complete track or even cylinder, aligned on an appropriate track or cylinder boundary.) The high-performance files system VERITAS uses extents to optimize performance.

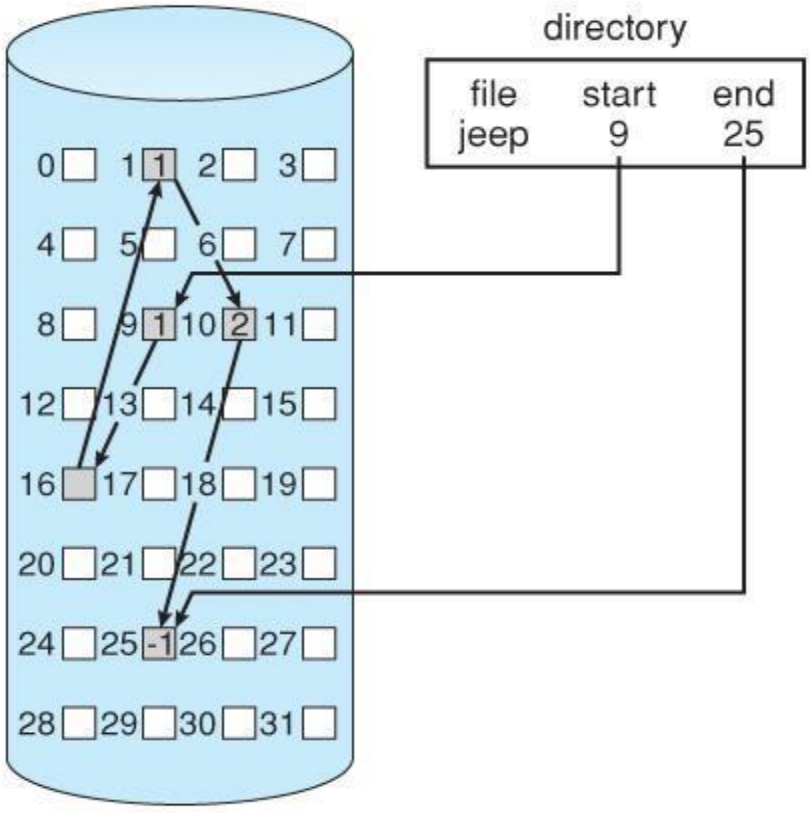


Contiguous allocation of disk space

25.a.Explain in detail about Linked file allocation method

Linked Allocation

Disk files can be stored as linked lists, with the expense of the storage space consumed by each link. (E.g. a block may be 508 bytes instead of 512.) Linked allocation involves no external fragmentation, does not require pre-known file sizes, and allows files to grow dynamically at any time. Unfortunately linked allocation is only efficient for sequential access files, as random access requires starting at the beginning of the list for each new location access. Allocating **clusters** of blocks reduces the space wasted by pointers, at the cost of internal fragmentation. Another big problem with linked allocation is reliability if a pointer is lost or damaged. Doubly linked lists provide some protection, at the cost of additional overhead and wasted space.



Linked allocation of disk space.

25.b.Explain in Detail About Authentication

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 can not 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.

- Typically done with *passwords*:

  - A secret piece of information used to establish identity of a user.

- Must not be stored in a directly-readable form: use one-way transformations.
- Passwords should be relatively long and obscure.

 Alternate form of authentication: badge or key.

- Does not have to be kept secret.
- Should not be forgable or copyable.
- Can be stolen, but owner should know if it is.

 Paradox: key must be cheap to make, hard to duplicate.

 Once authentication is complete, the identity of the principal must be protected from tampering, since other parts of the system will rely on it.

 Once you log in, your user id is associated with every process executed under that login: each process inherits the user id from its parent.

26.a.Explain in detail about Policy mechanism

Protection: mechanisms that prevent accidental or intentional misuse of a system.

- Accidents: generally easier to solve (make them unlikely)
- Malicious abuse: much more difficult to eliminate (can't leave any loopholes, can't use probabilities).

Three aspects to a protection mechanism:

- Authentication: identify a responsible party (*principal*) behind each action.
- Authorization: determine which principals are allowed to perform which actions.
- Access enforcement: combine authentication and authorization to control access.

A tiny flaw in any of these areas can compromise the entire protection mechanism.

AUTHENTICATION

- Typically done with *passwords*:
  - A secret piece of information used to establish identity of a user.
  - Must not be stored in a directly-readable form: use one-way transformations.
  - Passwords should be relatively long and obscure.
- Alternate form of authentication: badge or key.
  - Does not have to be kept secret.
  - Should not be forgable or copyable.
  - Can be stolen, but owner should know if it is.
- Paradox: key must be cheap to make, hard to duplicate.
- Once authentication is complete, the identity of the principal must be protected from tampering, since other parts of the system will rely on it.
- Once you log in, your user id is associated with every process executed under that login: each process inherits the user id from its parent.

INTERNAL ACCESS AUTHORIZATION

Authorization:

- Goal: determine which principals can perform which operations on which objects.
- Logically, authorization information represented as an access matrix:
- One row per principal.
- One column per object.

- Each entry indicates what that principle can do to that object.
- In practice a full access matrix would be too bulky, so it gets stored in one of two compressed ways: access control lists or capabilities.
- Access Control Lists (ACLs): organize by columns.
- With each object, store information about which users are allowed to perform which operations.
- Most general form: list of <user, privilege> pairs.
- For simplicity, users can be organized into groups, with a single ACL for an entire group.
- ACLs can be very general (Windows) or simplified (Unix).
- UNIX: 9 bits per file:
- owner, group, anyone
- read, write, execute permissions for each of the above
- In addition, user "root" has all permissions for everything
- ACLs are simple and are used in almost all file systems.
- Capabilities: organize by rows.
- With each user, indicate which objects may be accessed, and in what ways.
- Store a list of <object, privilege> pairs with each user. This is called a capability list.
- Typically, capabilities also act as names for objects: can't even name objects not referred to in your capability list.
- Almost as if there were no root directory in Unix and no "..".
- Systems based on ACLs encourage visibility of objects: shared public namespace.

- Capability systems discourage visibility; namespaces are private by default.
- Capabilities have been used in experimental systems attempting to be very secure. However, they have proven to be clumsy to use (painful to share things), so they have mostly fallen out of favor for managing objects such as files.
- Example of a simple capability-based protection scheme: page tables.

Access Enforcement

- Some part of the system must be responsible for enforcing access controls and protecting authentication and authorization info.
- This portion of the system has total power, so it should be as small and simple as possible. Example: the portion of the system that sets up page tables.
- Security kernel: an inner layer of the operating system that enforces security; only this layer has total power.
- Most operating systems have no security kernel: the entire OS has unlimited power.
- Miscellaneous Issues
- There are many other things that need to be protected besides just file access
- In Unix, root access is used to control most of these things.

26. b.Discuss briefly about Internal access authorization

INTERNAL ACCESS AUTHORIZATION

Authorization:

- Goal: determine which principals can perform which operations on which objects.
- Logically, authorization information represented as an access matrix:

- One row per principal.
- One column per object.
- Each entry indicates what that principle can do to that object.

- In practice a full access matrix would be too bulky, so it gets stored in one of two compressed ways: access control lists or capabilities.
- Access Control Lists (ACLs): organize by columns.
- With each object, store information about which users are allowed to perform which operations.
- Most general form: list of <user, privilege> pairs.
- For simplicity, users can be organized into groups, with a single ACL for an entire group.
- ACLs can be very general (Windows) or simplified (Unix).
- UNIX: 9 bits per file:
- owner, group, anyone
- read, write, execute permissions for each of the above
- In addition, user "root" has all permissions for everything
- ACLs are simple and are used in almost all file systems.
- Capabilities: organize by rows.
- With each user, indicate which objects may be accessed, and in what ways.
- Store a list of <object, privilege> pairs with each user. This is called a capability list.
- Typically, capabilities also act as names for objects: can't even name objects not referred to in your capability list.

- Almost as if there were no root directory in Unix and no "..".

- Systems based on ACLs encourage visibility of objects: shared public namespace.

- Capability systems discourage visibility; namespaces are private by default.

- Capabilities have been used in experimental systems attempting to be very secure. However, they have proven to be clumsy to use (painful to share things), so they have mostly fallen out of favor for managing objects such as files.

- Example of a simple capability-based protection scheme: page tables.