14CS IT304

Hall Ticket Number:

II/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

November, 2016 **Common for CSE & IT**

Third Semester **Operating Systems**

Time: Three Hours **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1. Answer all questions (1X12=12 Marks)

a) Describe Various Process states.
b) Write the differences between Process & Program.
c) What is "Inter Process Communication (IPC)"?
d) Define Response time.
e) What is a "Monitor"?
f) What is the purpose of Process Synchronization?
g) What is need for Swapping?
h) What is "Thrashing"?
i) What is a "Page Fault"?
j) List operations of a File.
k) Describe various File types.
l) List operations on Directory.

UNIT I

2. a) Define Operating System. List and Explain Operating System services. 6M
b) Explain operating system structure 6M

(OR)

3. a) What is "Multi- Threading"? Explain various Multi -Threading models. 6M
b) Explain process state transition diagram with neat sketch 6M

UNIT II

4. a) What is "Dinning-Philosophers" Problem? Give its solution. 6M
b) What is critical section? Explain Peterson's solution 6M

(OR)

5. a) Describe in detail about types of Schedulers and Scheduling Criteria. 4M
b) Consider the following set of process with the length of the CPU burst time given in milliseconds.
The process are assumed to have arrived in the order 1,2,3 all at the time zero.

Process	CPU Burst Time
P1	24
P2	3
P3	3

i) Draw Gantt chart that indicates the execution of these processes using the Scheduling
Algorithms FCFS & SJF. 8M
ii) What is waiting time of each process for each of these scheduling algorithms?
iii) What is turnaround time of each process for each of these scheduling algorithms?

UNIT III

6. Discuss about Deadlock Detection and its Recovery methods 12M

(OR)

7.a. What is need for "Demand Paging"? 4M
b. What are the steps in handling a "Page Fault"? 8M

UNIT IV

8.	a) Explain in detail implementation of access matrix.	8M
	b) Explain different types of I/O devices and its characteristics.	4M

(OR)

9.a	Explain different File access methods.	6M
9.b	Explain different Directory structures.	6M

14CS IT304

II/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

November, 2016 **Common for CSE & IT**

Third Semester **Operating Systems**

Time: Three Hours **Maximum:** 60 Marks

Scheme of Evaluation & Answers

1. ***Answer all questions*** (1 x 12 = 12 Marks)

a) Describe various process states.

Ans: 1) New 2) Ready 3) Running 4) Waiting 5) Terminate

b) Write the difference between process and program.

Ans: a process is a program in execution. A process is the unit of work in a modern time-sharing system. a process is an *active* entity, with a program counter specifying the next instruction to execute and a set of associated resources.

A program is a *passive* entity, such as a file containing a list of instructions stored on disk.

c) What is inter process communication (IPC)?

Ans: Inter-process communication (IPC) is a mechanism that allows the exchange of data between processes and provides a mechanism to allow the co-operating process to communicate with each other and synchronies their actions without sharing the same address space. It is provided a message passing system.

d) Define Response time.

Ans: the time from the submission of a request until the first response is produced. This is called *response time. or*

Response time, is the time it takes to start responding not the time it takes to output the response.

e) What is a monitor?

Ans: A *abstract data type-* or ADT- encapsulates private data with public methods to operate on that data. A *monitor type* is an ADT which presents a set of programmer-defined operations that are provided mutual exclusion within the monitor.

f) What is the purpose of process synchronization?

Ans: To ensure the orderly execution of cooperating processes that share a logical address space, so that data consistency is maintained.

g) What is need for swapping?

Ans: A process needs to be in memory to be executed. However a process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.

h) What is thrashing?

Ans: High paging activity is called thrashing. Thrashing results in severe performance problems.

CPU utilization is plotted against the degree of multiprogramming. As the degree of multiprogramming increases, CPU utilization also increases. If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply.

i) What is a page fault?

Ans: The required page is not in main memory is called page fault.

j) List operations of a file.

Ans: 1) Creating a file 2) Wring a file 3) Reading a file 4) Repositioning within a file 5) Deleting a file 6) Truncating a file

k) Describe various file types.
Ans:

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Any two file types - 1 mark will be awarded

l) List operations on directory.
Ans: 1) Search for a file 2) Create a file 3) Delete a file 4) List a directory
5) Rename a file 6) Traverse the file system

UNIT-I

2 a) Define operating system. List and explain operating system services. 6M

OS definition- 1 Mark;	**List of OS services- 1 Mark;**	**Explanation- 4 Marks;**

Ans: An operating system is a program that manages the computer hardware. It acts as an intermediate between users of a computer and the computer hardware. It controls and coordinates the use of the hardware among the various application programs for the various users.
An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. These operating-system services are provided for the convenience of the programmer, to n1.ake the programming task easier.
One set of operating-system services provides functions that are helpful to the user.
User interface: Almost all operating systems have an interface can take several forms. One is a CLI, which uses text commands and a method for entering them. Another is a batch in which commands and directives to control those commands are entered into files, and those files are executed. Most commonly, a GUI is used. Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text. Some systems provide two or all three of these variations.

Program execution: The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally.

I/O operations: A running program may require I/O, which may involve a file or an I/O device. For specific devices, special functions may be desired. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.

File-system manipulation: The file system is of particular interest. Obviously, programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Finally, some programs include permissions management to allow or deny access to files or directories based on file ownership. Many operating systems provide a variety of file systems, sometimes to allow personal choice, and sometimes to provide specific features or performance characteristics.

Communications: There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network. Communications may be implemented via *shared memory* or through *message passing,* in which packets of information are moved between processes by the operating system.

Error detection: The operating system needs to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware, in I/0 devices, and in the user program. For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. Of course, there is variation in how operating systems react to and correct errors. Debugging facilities can greatly enhance the user's and programmer's abilities to use the system efficiently.

Resource allocation: When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different -types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code.

Protection and security: The owners of information stored in a multiuser or networked computer system may want to control use of that information. When. Several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself. Protection involves ensuring that all access to system-resources is controlled.

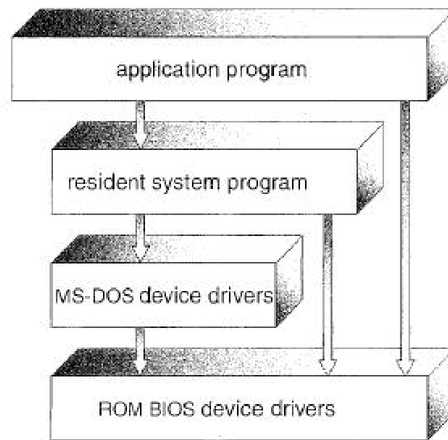
2 b) Explain operating system structure? 6M

Simple structure -	**3 Marks**
Layered Approach -	**3 Marks**

Ans: A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily. A common approach is to partition the task into small components rather than have one monolithic system. Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.

Simple Structure: Many commercial operating systen1.s do not have well-defined structures. Such systems started as small, simple, and limited systems and then grew beyond their original scope. MS-DOS is an example of such a system. It was originally designed and implemented by a few people who had no idea that it would become so popular. It was written to provide the most functionality in the least space, so it was not divided into modules carefully.

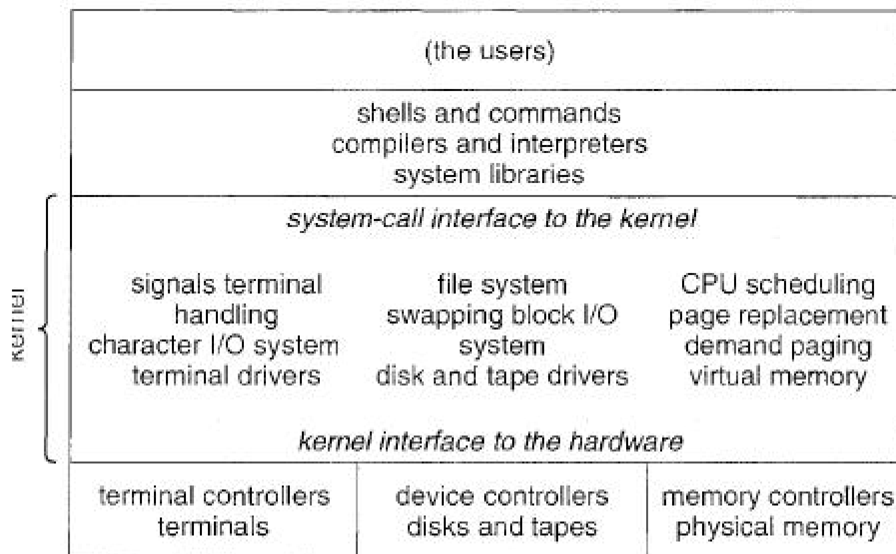
In MS-DOS, the interfaces and levels of functionality are not well separated. Another example of limited structuring is the original UNIX operating system. Like MS-DOS, UNIX initially was limited by hardware functionality. It consists of two separable parts: the kernel and the system programs.



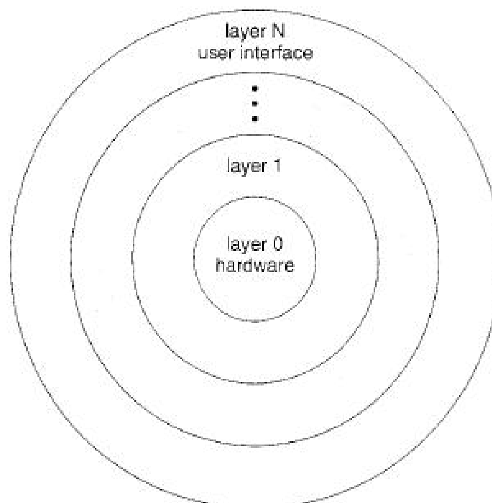
is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved. The traditional UNIX operating system as being layered.

Layered Approach:

With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS and UNIX systems.



A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest layer (layer N) is the user interface.



(OR)

3 a) What is multi-Threading? Explain various multi-Threading models? 6M

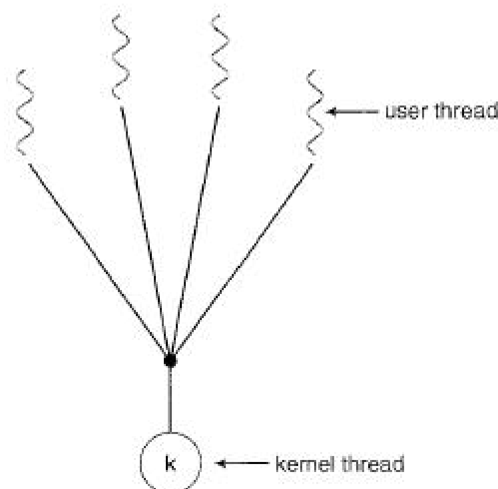
Multi-Threading -	**2 Marks**
Multi-Threading models -	**4 Marks**

Ans: A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
A traditional (or heavy weight) process has a single thread of control.
If a process has multiple threads of control, it can perform more than one task at a time.
Many software packages that run on modern desktop PCs are multithreaded. An application typically is implemented as a separate process with several threads of control. A Web browser might have one thread display images or text while another thread retrieves data from the network.
The benefits of multithreaded programming can be broken down into four major categories:

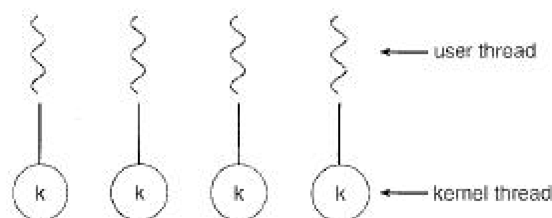
- Responsiveness
- Resource sharing
- Economy
- Utilization of multiprocessor architectures

Multi-Threading models: Three common ways of establishing a relationship between user threads and kernel threads.

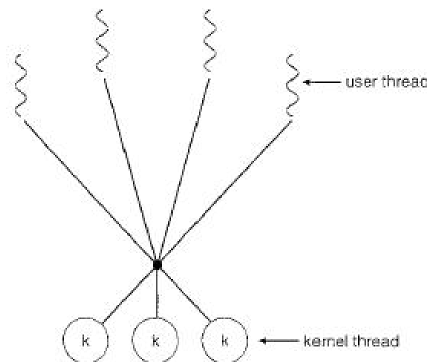
Many-to-One Model: The many-to-one model maps many user-level threads to one kernel thread. Thread management is done by the thread library in user space, so it is efficient; but the entire process will block if a thread makes a blocking system call, because only one thread can access the kernel at a Time, multiple threads are unable to nm in parallel on multiprocessors.



One-to-One Model: The one-to-one model (Figure 4.6) maps each user thread to a kernel thread. It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call; it also allows multiple threads to run in parallel on multiprocessors. The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.



Many-to-Many Model: The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads. The number of kernel threads may be specific to either a particular application or a particular machine. Whereas the many-to-one model allows the developer to create as many user threads as she wishes, true concurrency is not gained because the kernel can schedule only one thread at a time.

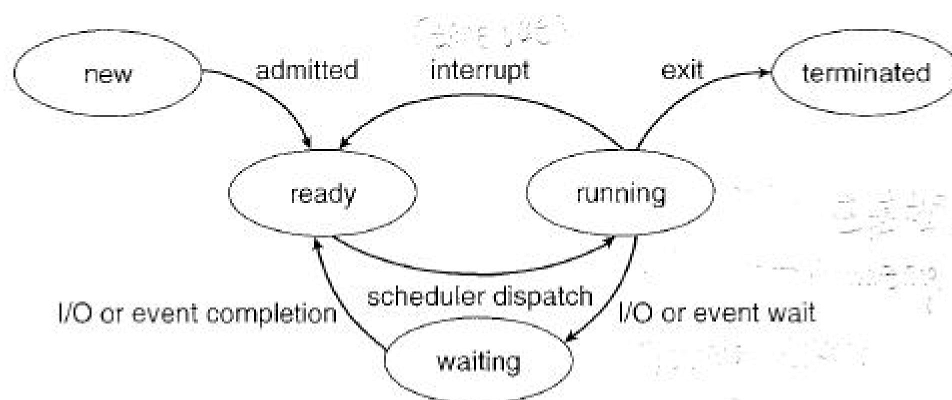


3 b) Explain process state transition diagram with neat sketch? 6M

Process states & explanation – 3 Marks;	**Process state diagram & explanation - 3 Marks**

Ans: A process is a program in execution. It is an active entity and it includes the process stack, containing temporary data and the data section contains global variables.
As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

New: The process is being created.
Running: Instructions are being executed.
Waiting: The process is waiting for some event to occur (such as an I/0 completion or reception of a signal).
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.*

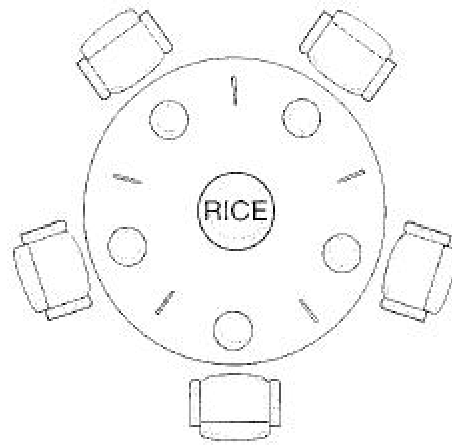
UNIT-II

4 a) What is dining philosopher's problem? Give its solution? 6M

Dinning philosophers problem – 2 Marks;	**Solution- 4 Marks**

Ans: Five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks. When a philosopher thinks, philosopher does not interact with colleagues. From time to time, a philosopher gets hungry and tries to pick up the two

chopsticks that are closest to philosopher. *A* philosopher may pick up only one chopstick at a time. Obviously, philosopher cannot pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, philosopher eats without releasing chopsticks. When philosopher is finished eating, philosopher puts down both of chopsticks and starts thinking again.




```
do {
        wait(chopstick[i]);
        wait(chopstick[(i+l) % 5]);
                // eat
        signal(chopstick[i]);
        signal(chopstick[(i+l) % 5]);
                // think
} while (TRUE);
```

Solution to the dining philosopher's problem:

```
monitor dp
{
        enum {THINKING, HUNGRY, EATING} state[5];
        condition self[5];
        void pickup(int i)
         {
                state[i] =HUNGRY;
                test(i);
                if (state [i] ! = EATING)
                self [i] . wait() ;
        }
        void putdown(int i)
         {
                state[i] =THINKING;
                test((i + 4) % 5);
                test((i + 1) % 5);
        }
        void test(int i)
         {
                if ((state[(i + 4) % 5] !=EATING) &&(state[i] ==HUNGRY) &&(state[(i + 1) % 5]
                        !=EATING))
                 {
                        state[i] =EATING;
                        self[i] .signal();
                }
        }
```

```
    Initialization_code()
    {
        for (int i = 0; i < 5; i++)
        state[i] =THINKING;
    }
}
```

4 b) What is critical section? Explain Peterson's solution? 6M

Critical Section Problem – 2 Marks;	**Peterson's solution- 4 Marks**

Ans: Consider a system consisting of *n* processes {Po, P1 , ... , P_{n-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 to be 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*

The general structure of a typical process *Pi* is

```
do {
        entry section
        exit section        on
            Remainder section
} while (TRUE);
```

A classic software-based solution to the critical-section problem known as Peterson's solution.
Peterson's solution requires the two processes to share two data items:

```
int turn;
10oolean flag[2];
```

The variable turn indicates whose turn it is to enter its critical section. If turn == I, then process *Pi* is allowed to execute in its critical section. The flag array is used to indicate if a process *is ready* to enter its critical section.

To enter the critical section, process *Pi* first sets flag [i] to be true and then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section, it can do so. If both processes try to enter at the same time, turn will be set to both I and j at roughly the same time. Only one of these assignments will last; the other will occur but will be overwritten immediately.

```
do {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);

        critical section

    flag[i] = FALSE;

        remainder section

} while (TRUE);
```

(OR)

5 a) Describe in detail about types of schedulers and scheduling criteria? 4M

Types of schedulers- 2 Marks;	Scheduling criteria- 2Marks

Ans: A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.

The long-term scheduler, or job scheduler, selects processes from this pool and loads them into memory for execution.

The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them.

The key idea behind a **medium-term scheduler** is that sometimes it can be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of multi programrning.

CPU utilization: We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

Throughput: the number of processes that are completed per time unit, called *throughput.*

Turnaround time: The interval from the time of submission of a process to the time of completion is the *turnaround time.* Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

Waiting time: *Waiting time* is the sum of the periods spent waiting in the ready queue.

Response time: the time from the submission of a request until the first response is produced. This measure, called *response time,* is the time it takes to start responding, not the time it takes to output the response.

5 b) Consider the following set of process with the length of the CPU burst time given in milliseconds. The processes are assumed to have arrived in the order 1, 2, 3 all at the time zero.

Process	Burst time
P1	24
P2	3
P3	3

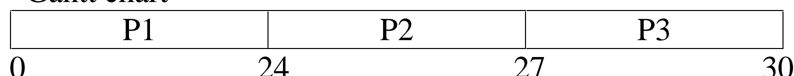
i) Draw Gantt chart that indicates the execution of these processes using the scheduling algorithms FCFS & SJF.

ii) What is waiting time of each process for each of these scheduling algorithms?

iii) What is turnaround time of each process for each of these scheduling algorithms? 8M

Ans:

FCFS Algorithm: Gantt chart



Waiting time of process P1= 0 ms

Waiting time of process P2= 24 ms

Waiting time of process P3= 27 ms

Average Waiting time of processes= (0 + 24 + 27) / 3 = 17 ms

Gantt Charts – 2 Marks
Waiting time- 3 Marks
Turnaround time- 3 Marks

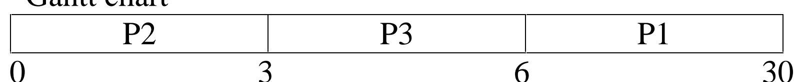
Turnaround time of process P1= 24 ms

Turnaround time of process P2= 27 ms

Turnaround time of process P3= 30 ms

Average turnaround time of processes= (24 + 27+30) / 3 = 27 ms

SJF Algorithm: Gantt chart



Waiting time of process P1= 6 ms
Waiting time of process P2= 0 ms
Waiting time of process P3= 3 ms
Average Waiting time of processes= (6 + 0 + 3) / 3 = 3 ms

Turnaround time of process P1= 30 ms
Turnaround time of process P2= 3 ms
Turnaround time of process P3= 6 ms

Average turnaround time of processes= (30 + 3+6) / 3 = 13 ms

UNIT-III

6. Discuss about deadlock detection and its recovery methods. 12M

Deadlock detection -	**Single Instance of Each Resource Type- 2M**
	Several Instances of a Resource Type- 6 M
Deadlock Recovery-	**Process termination- 2M**
	Resource preemption- 2M

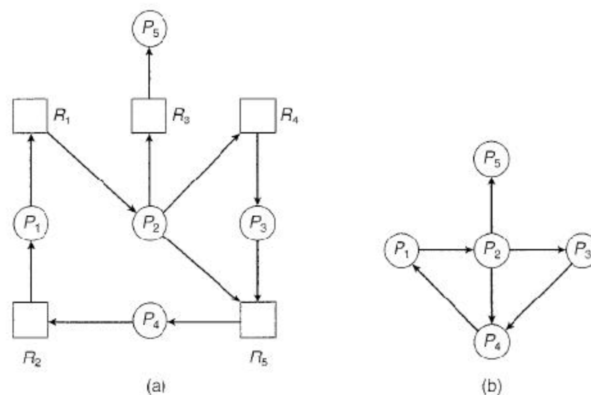
Ans: If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system may provide:

- 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.

An edge from *Pi* to *Pi* 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*.



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.

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.

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

Allocation: An nxm matrix defines the number of resources of each type currently allocated to each process.

Request: An *n* x *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.*

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. Fori=* 0, 1, ... , *n-1,* if *Allocation;* # 0, then *Finish[i] =false;* otherwise, *Finish[i] = tme.*

2. Find an index *i* such that both

a. *Finish[i] ==false*

b. *Request;* :::: *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] ==false* for some *i,* 0 :::: *i* < *n,* then the system is in a deadlocked state. Moreover, if *Finish[i] ==false,* then process *P;* is deadlocked.

This algorithm requires an order o£ *m* x *n2* operations to detect whether the system is in a deadlocked state.

Example:

we consider a system with five processes *Po* 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*
	ABC	*ABC*	*ABC*
P0	0 1 0	000	000
P1	200	202	
P2	303	000	
P3	2 11	100	
P4	002	002	

We claim that the system is not in a deadlocked state. Indeed, if we execute our algorithm, we will find that the sequence <*Po, 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
	ABC
Po	000
P1	202
P2	001
P3	100
P4	002

We claim that the system is now deadlocked. Although we can reclaim the resources held by process *Po,* 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:

There are two options for breaking a deadlock:

- One is simply to abort one or more processes to break the circular wait.
- The other is to preempt some resources from one or more of the deadlocked processes.

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;

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.

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:

Selecting a victim- Which resources and which processes are to be preempted

Roll back-If we preempt a resource from a process, what should be done with that process

Starvation-how can we guarantee that resources will not always be preempted from the same Process

(OR)

7 a) What is need for demand paging? 4M

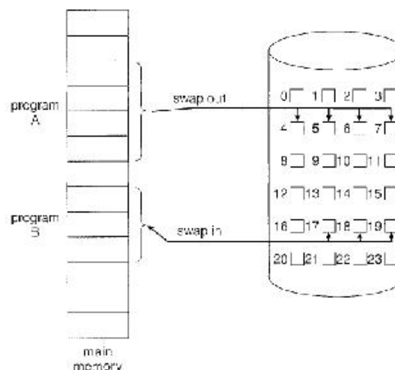
Explanation of demand paging- 4 Marks

Ans: Load the entire program in physical memory at program execution time. However, a problem with this approach is that we may not initially *need* the entire program in memory. Suppose a program starts with a list of available options from which the user is to select. Loading the entire program into memory results in loading the executable code for *all* options, regardless of whether an option is ultimately selected by the user or not. An alternative strategy is to load pages only as they are needed. This technique is known as demand paging and is commonly used in virtual memory systems.

With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.

A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk). 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.

A lazy swapper never swaps a page into memory unless that page will be needed.



7 b) What are the steps in handling a page fault? 8M

Steps in handling a page fault- 5 Marks
Explanation & diagram- 3 Marks

Ans: The procedure for handling this page fault is straightforward.

1. We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

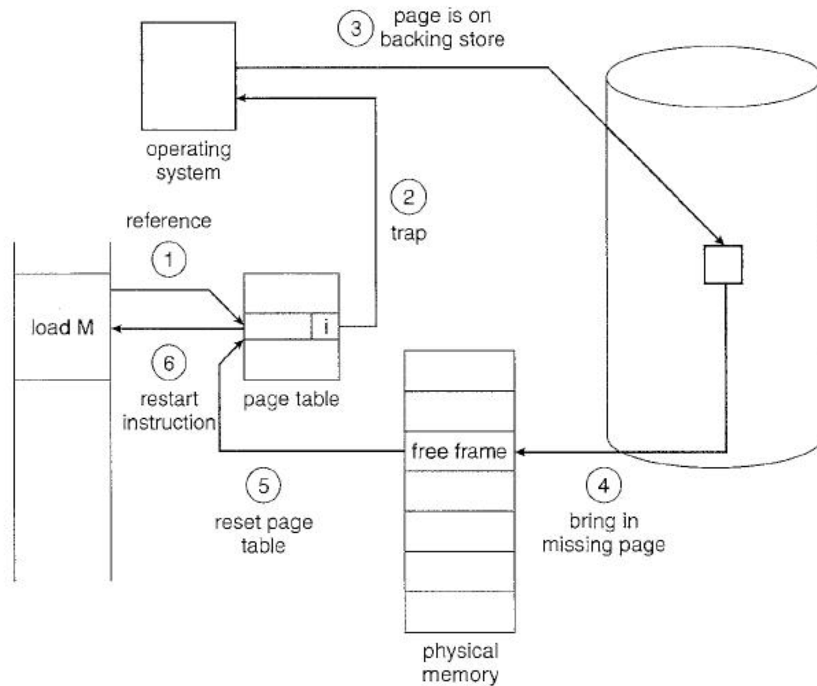
2. If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.

3. We find a free frame (by taking one from the free-frame list, for example).

4. We schedule a disk operation to read the desired page into the newly allocated frame.

5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.



UNIT-IV

8 a) Explain in detail implementation of access matrix. 8M

Access matrix- 2 Marks
Implementation of access matrix- 6 Marks

Ans: Protection can be viewed abstractly as a matrix, called an access matrix. The rows of the access matrix represent domains, and the columns represent objects. Each entry in the matrix consists of a set of access rights. Because the column defines objects explicitly, we can omit the object name from the access right. The entry access(i,j) defines the set of operations that a process executing in domain D*i* can invoke on object *Oj.*
To illustrate these concepts, we consider the access matrix. There are four domains and four objects- three files (F1, F2, F3) and one laser printer. A process executing in domain D1 can read files F1 and *F3* . A process executing in domain D4 has the same privileges as one executing in domain D1; but in addition, it can also write onto files F1 and *F3* . Note that the laser printer can be accessed only by a process executing in domain D2.

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

The access-matrix scheme provides us with the mechanism for specifying a variety of policies. The mechanism consists of implementing the access matrix and ensuring that the semantic properties we have outlined hold. More specifically, we must ensure that a process executing in domain Di can access only those objects specified in row i and then only as allowed by the access-matrix entries.
The access matrix can implement policy decisions concerning protection. The policy decisions involve which rights should be included in the (i, j)th entry. We must also decide the domain in which each process executes. This last policy is usually decided by the operating system.
The access matrix provides an appropriate mechanism for defining and implementing strict control for both the static and dynamic association between processes and domains.
When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain).

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Processes should be able to switch from one domain to another. Switching from domain Di to domain *Dj* is allowed if and only if the access right switch € access(i,j). Thus, in Figure 14.4, a process executing in domain D2 can switch to domain D3 or to domain D4. *A* process in domain D4 can switch to D1, and one in domain D1 can switch to D2.
The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an asterisk (*) appended to the access right. The *copy* right allows the access right to be copied only within the column. (that is, for the object) for which the right is defined.
This scheme has two variants:

- A right is copied from access(i, *j)* to access(k, *j);* it is then removed from access(i, *j).* This action is a *transfer* of a right, rather than a copy.
- Propagation of the *copy* right may be limited. That is, when the right R* is copied from access(i,j) to access(k,j), only the right *R* (not R*) is created. A process executing in domain Dk cannot further copy the right *R.*

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

We also need a mechanism to allow addition of new rights and removal of some rights. The *owner* right controls these operations. If access(i, *j)* includes the *owner* right, then a process executing in domain *Di* can add and remove any right in any entry in column *j.*

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

The *copy* and *owner* rights allow a process to change the entries in a column. A mechanism is also needed to change the entries in a row. The *control* right is applicable only to domain objects. If access(i, *j)* includes the *control* right, then a process executing in domain Di can remove any access right from row *j.*

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

the matrix will be sparse; that is, most of the entries will be empty. Although data structure techniques are available for representing sparse matrices, they are not particularly useful for this application, because of the way in which the protection facility is used.

- The simplest implementation of the access matrix is a global table consisting of a set of ordered triples <*domain, object, rights-set*>. Whenever an operation *M* is executed on an object *Oj* within domain D;, the global table is searched for a triple <Di, *Oj,* Rk>, with M €Rk. If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.
- Each column in the access matrix can be implemented as an access list for one object. Obviously, the empty entries can be discarded. The resulting list for each object consists of ordered pairs <*domain, rights-set*>, which define all domains with a nonempty set of access rights for that object.
- Other methods related to access matrix

8 b) Explain different types of I/O devices and its characteristics? 4M

I/O devices and its characteristics- 4 Marks

Ans: The two main jobs of a computer are I/O and processing. The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices

The control of devices connected to the computer is a major concern of operating-system designers. Because I/O devices vary so widely in their function and speed (consider a mouse, a hard disk, and a CD-ROM jukebox), varied methods are needed to control them. These methods form the *I/O subsystem* of the kernel which separates the rest of the kernel from the complexities of managing I/O devices.

Computers operate a great many kinds of devices. Most fit into the general categories of storage devices (disks, tapes), transmission devices (network cards, modems), and human-interface devices (screen, keyboard, mouse).

A device communicates with a computer system by sending signals over a cable or even through the air. The device communicates with the machine via a connection point, or port. If devices use a common set of wires, the connection is called a *bus. A is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.*

(OR)

9 a) Explain different file access methods. 6M

Sequential Access - 3 Marks
Direct Access – 3 Marks

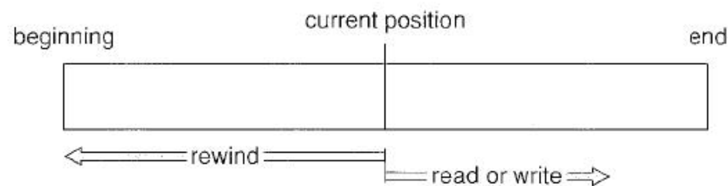
Ans: Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways. Some systems provide only one access method for files. File access methods are two types

- Sequential Access
- Direct Access

Sequential Access

The simplest access method is sequential access. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

Reads and writes make up the bulk of the operations on a file. A read operation-read *next-reads* the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, the write *operation-write* next-appends to the end of the file and advances to the end of the newly written material. Such a file can be reset to the beginning; and on some systems, a program may be able to skip forward or backward *n* records for some integer n-perhaps only for *n* = 1. Sequential access, which is based on a tape model of a file and works as well on sequential-access devices as it does on random-access ones.



Direct Access

A file is made up of fixed length that allows programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.

Direct-access files are of great use for immediate access to large amounts of information. Databases are often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.

For the direct-access method, the file operations must be modified to include the block number as a parameter. Thus, we have *read n,* where *n* is the block number, rather than *read next,* and *write n* rather than *write next.* An alternative approach is to retain *read next* and *write next,* as with sequential access, and to add an operation *position file to n,* where *n* is the block number. Then, to effect a *read n,* we would *position to n* and then *read next.*

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

The block number provided by the user to the operating system is normally a relative block number is an index relative to the begirm.ing of the file. Thus, the first relative block of the file is 0, the next is 1, and so on, even though the absolute disk address may be 14703 for the first block and 3192 for the second. The use of relative block numbers allows the operating system to decide where the file should be placed (called the *allocation problem*) and helps to prevent the user from accessing portions of the file system that may not be part of her file. Some systems start their relative block numbers at 0; others start at 1.

9 b) Explain different directory structures. 6M

One directory structure- 2 Marks will be awarded

Any three directory structures- 3x2 Marks

Ans: The most common schemes for defining the logical structure of a directory are

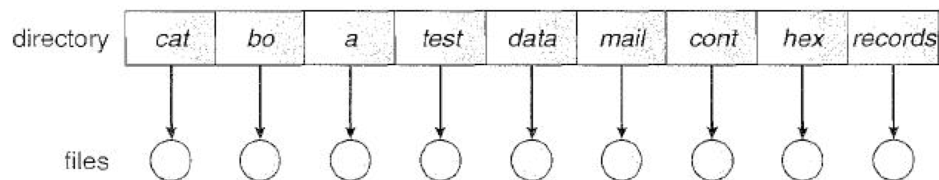
1. Single level directory
2. Two-level directory
3. Tree structured directories
4. Acyclic-Graph Directories

Single –level directory

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

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

Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases.



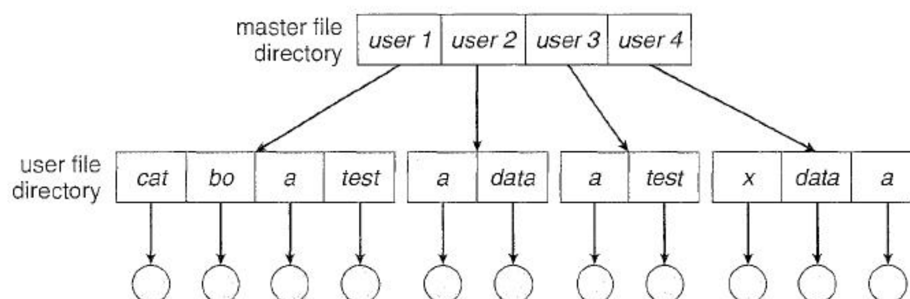
Two-Level Directory

A single-level directory often leads to confusion of file names among different users. The standard solution is to create a *separate* directory for each user.

In the two-level directory structure, each user has his own user level file directory (UFD). The UFDs have similar structures, but each lists only the files of a single user. W11en a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user. When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.

The user directories themselves must be created and deleted as necessary.

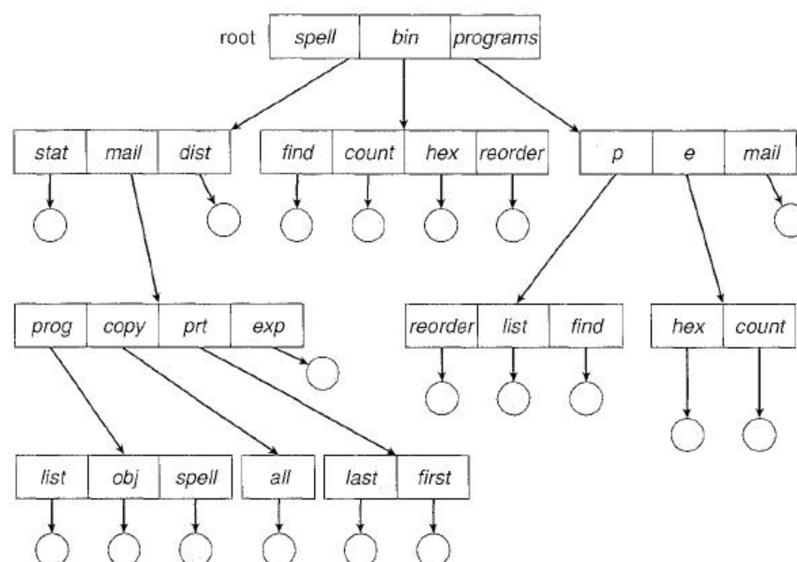
The two-level directory structure solves the name-collision problem.



Tree-Structured Directories

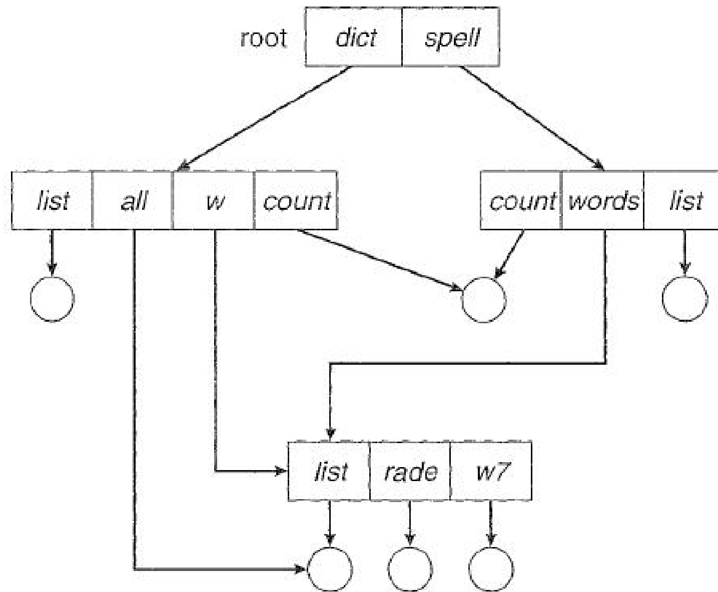
the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name.

A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way. All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1). Special system calls are used to create and delete directories.



Acyclic-Graph Directories

A tree structure prohibits the sharing of files or directories. An acyclic graph -that is, a graph with no cycles-allows directories to share subdirectories and files. The *same* file or subdirectory may be in two different directories. The acyclic graph is a natural generalization of the tree-structured directory scheme.



****

Scheme prepared by **Signature of the HOD, IT DEPT.**

Paper Evaluators:

S.No	Name of the College	Name of the Examiner	Signature