B.TECH.

SIXTH SEMESTER EXAMINATION 2009-10

OPERATING SYSTEMS

Time : 3 Hours *Total Marks : 100*

Note: (i) Attempt all questions.

(ii) All questions carry equal marks.

Q.1 Attempt any four of the following:

(a) Write down the advantages of Batch Processing System.

Ans. Batch processing is execution of a series of programs ("jobs") on a computer without human interaction.

Batch jobs are set up so they can be run to completion without human interaction, so all input data is preselected through scripts or command-line parameters. This is in contrast to "online" or interactive programs which prompt the user for such input.

Advantages: Batch processing has these benefits:

- It allows sharing of computer resources among many users and programs,
- It shifts the time of job processing to when the computing resources are less busy,
- It avoids idling the computing resources with minute-by-minute human interaction and supervision,
- By keeping high overall rate of utilization, it better amortizes the cost of a computer, especially an expensive one.

Disadvantages:

- As it does not provide human interaction so user would not be able to perform on-line debugging.
- For shorter jobs it does not provide good results.

(b) What are the major functions of Operating System?

Ans. Major Functions of Operating System:

(i) **Process Management:** The operating system is responsible for the following activities in connection with process management.
(a) Process creation and deletion.
(b) process suspension and resumption.
Provision of mechanisms for:
(a) process synchronization
(b) process communication

(ii) **Main-Memory Management:** The operating system is responsible for the following activities in connections with memory management:
(a) Keep track of which parts of memory are currently being used and by whom.
(b) Decide which processes to load when memory space becomes available.
(c) Allocate and deallocate memory space as needed.

(iii) **File Management:** The operating system is responsible for the following activities in connections with file management:
(a) File creation and deletion.
(b) Directory creation and deletion.
(c) Support of primitives for manipulating files and directories.
(d) Mapping files onto secondary storage.

(iv) **Input/Output System Management :** The operating system is responsible for the following activities in connection with disk management:
(a) Free space management
(b) Storage allocation
(c) Disk scheduling

(v) **Resource allocation** – allocating resources to multiple users or multiple jobs running at the same time.

(vi) **Accounting** – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.

(vii) **Protection** – ensuring that all access to system resources is controlled.

(c) Explain the main features of an real time operating system.

Ans. Real-Time Systems: Main features of real time operating system are:

(i) Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.

(ii) Well-defined fixed-time constraints.

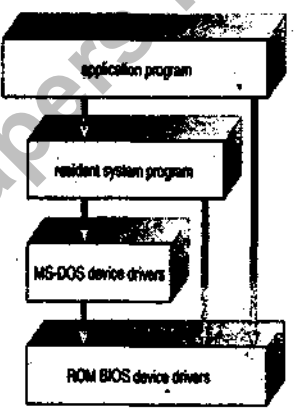
(iii) Real-Time systems may be either *hard* or *soft* real-time.

(iv) **Hard real-time:**

- Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
- Conflicts with time-sharing systems, not supported by general-purpose operating systems.

(v) **Soft real-time:**

- Limited utility in industrial control of robotics
- Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.



(d) Draw the Layered structure of an operating system.

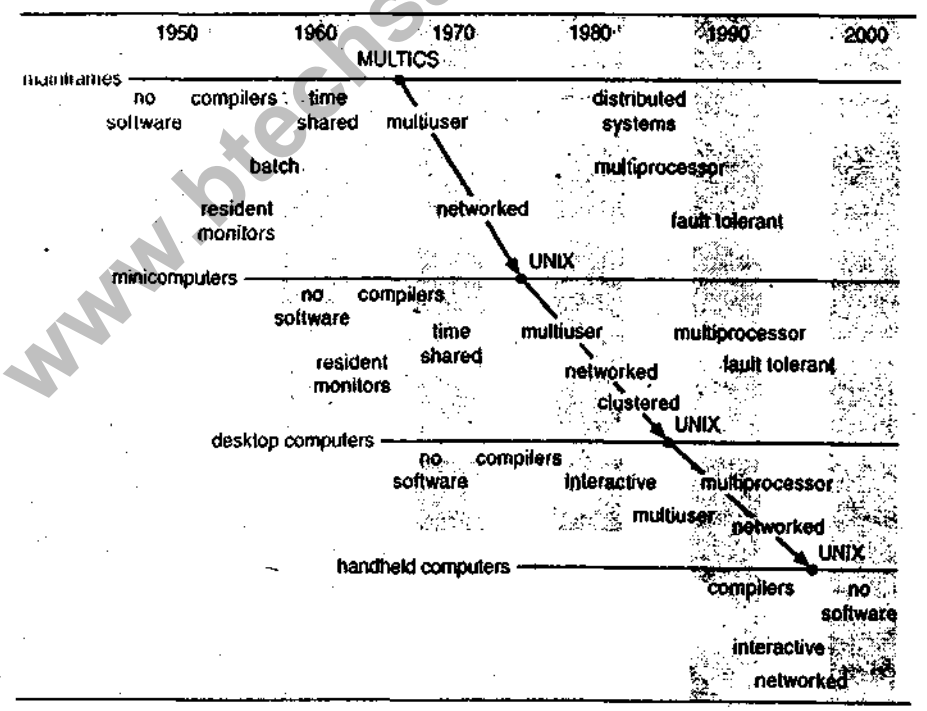
Ans. Layered Structured of Operating System:

- MS-DOS – written to provide the most functionality in the least space
  - not divided into modules

Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.

(e) Discuss the Evolution of operating system.

Ans. Evolution of Operating System:



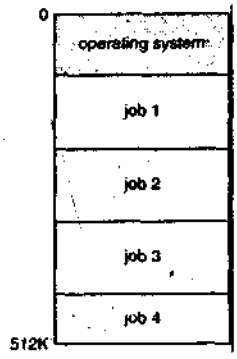
1. Mainframe Systems:

(i) Automatic job sequencing – automatically transfers control from one job to another. First rudimentary operating system.

(ii) Resident monitor
- initial control in monitor
- control transfers to job

Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them

2. Multiprogrammed Batch Systems:



Need for Multiprogramming
- I/O routine supplied by the system.
- Memory management – the system must allocate the memory to several jobs.
- CPU scheduling – the system must choose among several jobs ready to run.

Allocation of devices

3. Time-Sharing Systems–Interactive Computing:
- The CPU is multiplexed among several jobs that are kept in memory and on disk (the CPU is allocated to a job only if the job is in memory).
- A job swapped in and out of memory to the disk.
- On-line communication between the user and the system is provided; when the operating system finishes the execution of one command, it seeks the next "control statement" from the user's keyboard.

4. Desktop Systems:
- ***Personal computers*** – computer system dedicated to a single user.
- I/O devices – keyboards, mice, display screens, small printers.

User convenience and responsiveness

5. Parallel Systems:
- Multiprocessor systems with more than on CPU in close communication.
- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.
- **Advantages of parallel system:**
  - Increased *throughput*
  - Economical
- Increased reliability

6. Distributed Systems:
- Distribute the computation among several physical processors.
- *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.
- **Advantages of distributed systems.**
  - Resources Sharing
  - Computation speed up – load sharing
  - Reliability
  - Communications

7. Real-Time Systems:
- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- Well-defined fixed-time constraints.
- Real-Time systems may be either *hard* or *soft* real-time

8. Handheld Systems:
- Personal Digital Assistants (PDAs)
- Cellular telephones
- **Issues:**
  1. Limited memory
  2. Slow processors
  3. Small display screens

(f) Write down about the following in brief:

(a) System protection

(b) System Components

Ans. (a) System protection:
- Dual-Mode Operation
- I/O Protection

- Memory Protection
- CPU Protection

1. Dual-Mode Operation:

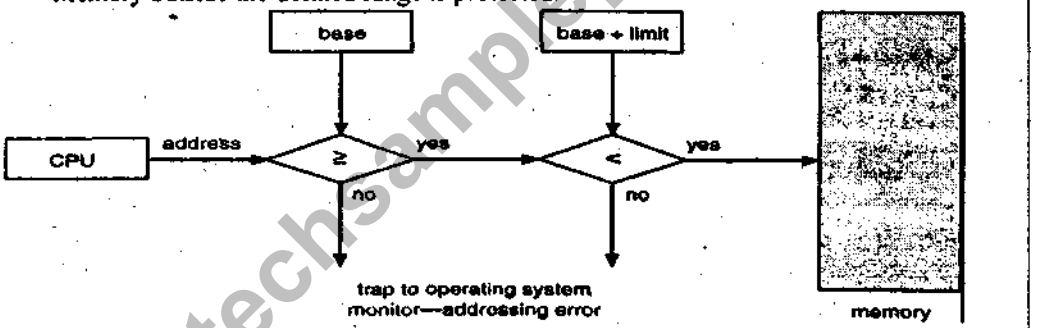
- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.
  - *User mode* – execution done on behalf of a user.
  - *Monitor mode* (also *kernel mode* or *system mode*) – execution done on behalf of operating system.

2. I/O Protection:

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (I.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

3. Memory Protection:

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
- Base register – holds the smallest legal physical memory address.
- Limit register – contains the size of the range
- Memory outside the defined range is protected.

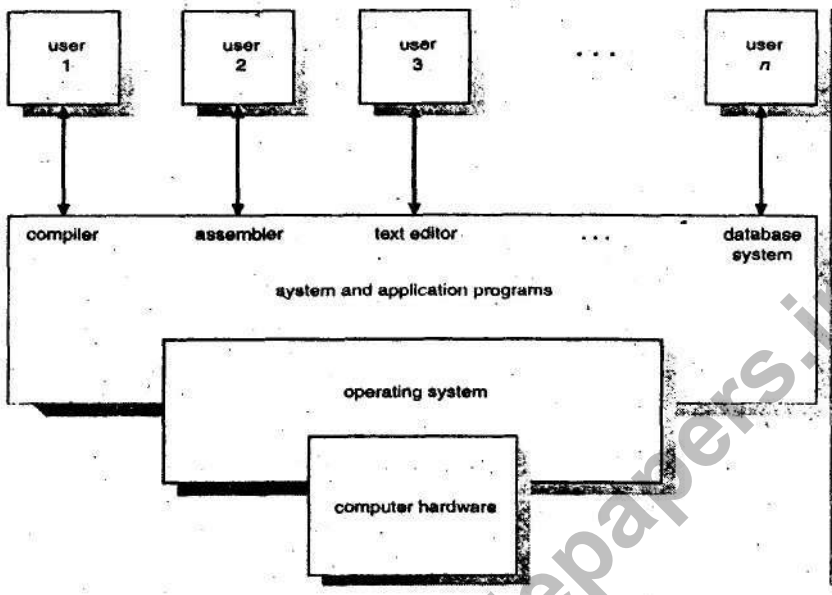


4. CPU Protection:

- *Timer* – interrupts computer after specified period to ensure operating system maintains control.
  (a) Timer is decremented every clock tick.
  (b) When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time sharing.
- Time also used to compute the current time.
- Load-timer is a privileged instruction.

Ans. (b) System Component

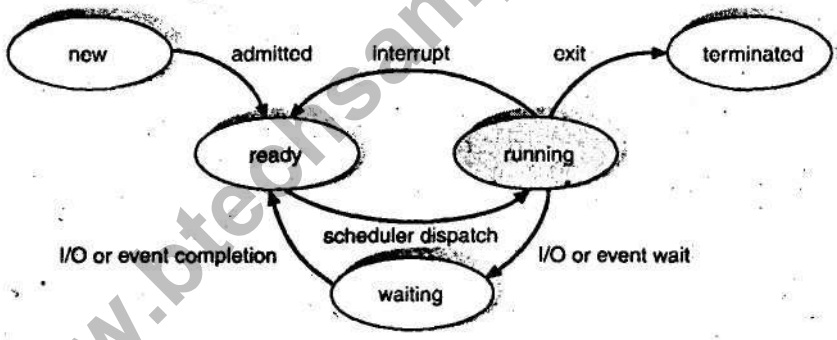
- Hardware – provides basic computing resources (CPU, memory, I/O devices).
- Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.
- Applications programs – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
- Users (people, machines, other computers).



Q.2. Attempt any four of the following:

(a) Draw the process states diagram & explain the state transaction.

Ans. Process State Diagram:



STATE TRANSACTION: As a process executes, it changes *state:*

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

(b) Provide the solution of the critical section problem.

Ans. The Critical-Section Problem:

- processes all competing to use some shared data
- Each process has a code segment, called *critical section*, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Solution to Critical-Section Problem:

- **Mutual Exclusion.** If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

- **Progress.** If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
- **Bounded Waiting.** A bound must exist 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.
  - Assume that each process executes at a nonzero speed
  - No assumption concerning relative speed of the *n* processes.

(c) Explain the binary semaphores with an example.

Ans. BINARY SEMAPHORE:

1. Integer value can range only between 0 and 1; can be simpler to implement.
2. Can implement a counting semaphore *S* as a binary semaphore

Data structures:

```
binary-semaphore S1, S2;
int C:
```

Initialization:

```
S1 = 1
S2 = 0
C = initial value of semaphore S
```

Example:

wait operation

```
wait(S1);
C—;
if(C < 0) {
        signal(S1);
        wait(S2);
}
signal(S1);
```

signal operation

```
wait(S1);
C++;
if(C <= 0)
        signal(S2);
else
        signal(S1);
```

(e) Write a brief note on Inter Process Communication.

Ans. Concurrent Processes: *Cooperating* process can affect or be affected by the execution of another processes are known as concurrent processes.

Advantages of process cooperation

- Information sharing
- Computation speed-up
- Modularity
- Convenience

Producer-Consumer Problem: Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.

unbounded-buffer places no practical limit on the size of the buffer.

bounded-buffer assumes that there is a fixed buffer size

Example: Bounded-Buffer – Shared-Memory Solution

Shared data

```
#define BUFFER_SIZE 10
Typedef struct {
    ...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Solution is correct, but can only use BUFFER_SIZE-1 elements

Bounded-Buffer – Producer Process

```
item nextProduced;
    while (1) {
        while (((in + 1) %
BUFFER_SIZE) == out)
            ; /* do nothing */
        buffer[in] = nextProduced;
        in = (in + 1) %
BUFFER_SIZE;
    }
```

Bounded-Buffer – Consumer Process

```
item nextConsumed;
    while (1) {
        while (in == out)
            ; /* do nothing */
        nextConsumed = buffer[out];
        out = (out + 1) %
BUFFER_SIZE;
    }
```

(e) Write a brief note on Inter Process Communication.

Ans. Interprocess Communication (IPC):

- Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations:
  (a) send(*message*) – message size fixed or variable
  (b) receive(*message*)
- If *P* and *Q* wish to communicate, they need to:
  (a) establish a *communication link* between them
  (b) exchange messages via send/receive
- Implementation of communication link
  (a) physical (e.g., shared memory, hardware bus)
  (b) logical (e.g., logical properties)

Implementation Questions:

1. How are links established?
2. Can a link be associated with more than two processes?
3. How many links can there be between every pair of communicating processes?
4. What is the capacity of a link?
5. Is the size of a message that the link can accommodate fixed or variable?
6. Is a link unidirectional or bi-directional?

Direct Communication:

1. Processes must name each other explicitly:
   (a) **send** (*P, message*) – send a message to process P
   (b) **receive**(*Q, message*) – receive a message from process Q
2. Properties of communication link
   (a) Links are established automatically.
   (b) A link is associated with exactly one pair of communicating processes.
   (c) Between each pair there exists exactly one link.
   (d) The link may be unidirectional, but is usually bi-directional

Indirect Communication:

1. Messages are directed and received from mailboxes (also referred to as ports).
   (a) Each mailbox has a unique id.
   (b) Processes can communicate only if they share a mailbox.
2. Properties of communication link
   (a) Link established only if processes share a common mailbox
   (b) A link may be associated with many processes.
   (c) Each pair of processes may share several communication links.
3. Link may be unidirectional or bi-directional.

Synchronization:

(a) Message passing may be either blocking or non-blocking.
(b) Blocking is considered synchronous
(c) Non-blocking is considered asynchronous
(d) send and receive primitives may be either blocking or non-blocking.

Buffering: Queue of messages attached to the link; implemented in one of three ways.

1. ZeroCapacity
2. BoundedCapacity
3. Unbounded capacity

(f) How concurrency problems are solved with producer & consumer problem?

Ans. Producer-Consumer Problem: Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.

unbounded-buffer places no practical limit on the size of the buffer.

bounded-buffer assumes that there is a fixed buffer size.

Example:

Bounded-Buffer – Shared-Memory Solution

Shared data

```
#define BUFFER_SIZE 10
Typedef struct {
        ...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Solution is correct, but can only use BUFFER_SIZE-1 elements

Bounded-Buffer – Producer Process

```
item nextProduced;
```

```
while (1) {
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```

Bounded-Buffer – Consumer Process

```
item nextConsumed;
    while (1) {
        while (in == out)
            ; /* do nothing */
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
    }
```

Q.3. Attempt any four of the following:

(a) What do you understand by CPU Scheduling? Which one is best & Why?

Ans. CPU SCHEDULING:

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.
- CPU burst distribution.

CPU Scheduler:

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  - Switches from running to waiting state.
  - Switches from running to ready state.
  - Switches from waiting to ready.
  - Terminates.
  - Scheduling under 1 and 4 is *nonpreemptive*.
  - All other scheduling is *preemptive*.

Optimization Criteria:

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1, P_2, P_3

The Gantt Chart for the schedule is:

Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$

Average waiting time: $(0 + 24 + 27)/3 = 17$

Shortest-Job-First (SJF) Scheduling:

1. Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
2. Two schemes:
   (a) nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
   (b) preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
3. SJF is optimal – gives minimum average waiting time for a given set of processes.

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

SJF (non-preemptive)

Average waiting time $= (0 + 6 + 3 + 7)/4 = 4_{m.s}$

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

SJF (preemptive)

Average waiting time $= (9 + 1 + 0 + 2)/4 = 3_{m.s}$

Priority Scheduling:

1. A priority number (integer) is associated with each process

2. The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
   (a) Preemptive
   (b) nonpreemptive
3. SJF is a priority scheduling where priority is the predicted next CPU burst time.
4. Problem ≡ Starvation – low priority processes may never execute.
5. Solution ≡ Aging – as time progresses increase the priority of the process.

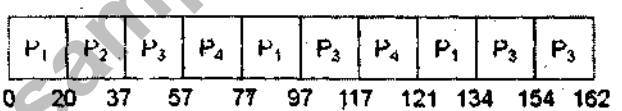
Round Robin (RR):

1. Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
2. If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
3. Performance
   a. q large ⇒ FIFO
   b. q small ⇒ q must be large with respect to context switch, otherwise overhead is too high.

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

The Gantt chart is:



Typically, higher average turnaround than SJF, but better *response*.

From the above calculation it is clear that SJF is having the smallest Turnaround time & good throughput. But it is tough to implement it practically. Larger job will suffer in this approach. Sometimes starvation problem also occur for the larger job, because they will never get CPU.

(b) Calculate turn around time & waiting time for the folling processes are using:

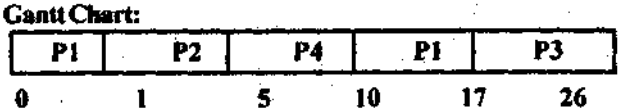
(i) SJF

(ii) FCFS

Process	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Ans. (i) Using Non-preemptive SJF:

Gantt Chart:



- Waiting time for P1=10-0=10

- Waiting time for P2=1-1=0
- Waiting time for P3=17-2=15
- Waiting time for P4=5-3=2

Average waiting time=10+0+15+2/4=6.75 m.s

- Turnaround time for P1=10+8=18
- Turnaround time for P2=0+4=4
- Turnaround time for P3=15+9=24
- Turnaround time for P4=2+5=7

Average Turnaround Time=18+4+24+7/4=13.25 m.s

(ii) Using FCFS: Gantt Chart:

P1	P2	P3	P4

0 8 12 21 26

- Waiting time for P1=0
- Waiting time for P2=8
- Waiting time for P3=12
- Waiting time for P4=21

Average waiting time=0+8+12+21/4=41/4=10.25 m.s

- Turnaround time for P1=8
- Turnaround time for P2=12
- Turnaround time for P3=21
- Turnaround time for P4=26

Average Turnaround Time=12+8+21+26/4=67/4=16.75 m.s

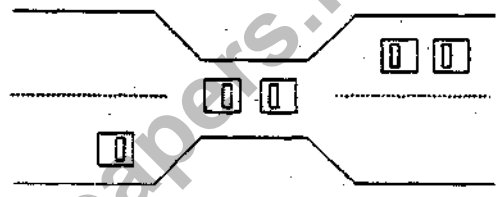
(c) Explain the different conditions of deadlock.

Ans. Different condition of dead lock are:

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
  - System has 2 tape drives.
  - P_1 and P_2 each hold one tape drive and each needs another one.
- Example
  - semaphores *A* and *B*, initialized to 1
  - P_0 P_1
  - *wait (A);* *wait(B)*
  - *wait (B);* *wait(A)*

Bridge Crossing Example:

- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

Deadlock Characterization

- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by
- $P_2 \ldots P_{n-1}$ is waiting for a resource that is held by P_n, and P_0 is waiting for a resource that is held by P_0.

(d) Write down the methods of deadlock prevention.

Ans. Deadlock Prevention:

1. Preventing deadlocks by constraining how requests for resources can be made in the system and how they are handled (system design).
2. The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.

Eliminate one of the four conditions:

- **Mutual Exclusion:**
  - Well, if we need it then we need it.
- **Hold and Wait:**
  - Require a process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
  - May lead to low resource utilization.
  - Starvation is a problem - a process may be held for a long time waiting for all its required resources.
  - If it needs additional resources, it releases all of the currently held resources and then requests all of those it needs (the application needs to be aware of all of the required resources).
- **No Preemption:**
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
  - The state of preempted resources has to be saved and later restored. Not practical for many types of resources (e.g. printer).
- **Circular Wait:**
  - Impose a total ordering on all resource types.
  - Require each process to request resources only in a strict increasing order.
  - Resources from the same resource type have to be requested together.

(e) Discuss about multiprocessor scheduling in brief.

Ans. In multiprocessor scheduling a system is having a no of processors .They compute a computation simultaneously by using different scheduling policy. Some time 1 processor out of all may serve as a master to control others to assign and co-ordinate works.

(f) How the recovery from deadlock is done using combined approach?

Ans. Deadlock Detection:

- *The system may enter a deadlock state.*
- *The system needs:*
  - An algorithm that periodically determines whether a deadlock has occurred in the system.
  - A procedure to recover from a deadlock.
- *Two algorithms:*
  - One instance per resource type.
  - Multiple inst

Single Instance for Each Resource Type

- ***Maintain a wait-for graph:***
  - Nodes are processes.
  - If process i is waiting for a resource held by process j then there is an edge from i to j.
  - Exactly the same as a resource allocation graph but optimize it for the search by collapsing edges.
  - Similar optimization could be done also for the resource allocation graph algorithm.
- ***Periodically invoke an algorithm that searches for cycles in the graph***

Several Instances of a Resource Type:

- ***Similar to the Banker?s algorithm safety procedure with the following semantics differences:***
  - Replacing Need by Waiting where Waitingi is the actual vector of resources process i is currently waiting to acquire.
  - May be slightly optimized by initializing Finish[i]to true for every process i where Allocationi = 0.
  - This is valid because we only care whether there is a deadlock right now. We are optimistic. If processes will need more resources in the future, this might lead to deadlock and we will discover it in the future.
  - Processes with a false entry in the end are the ones actually involved in a deadlock at this time.

Detection Algorithm Usage:

When or how often to invoke ?

- In the extreme case - every time a request for resource cannot be granted immediately.
  - This comes with enormous cost (think about complexity).
- Reasonable alternative - periodically. But what period to use?
  - How long are we willing to wait after it happens to be detected.
  - How much system resources are we willing to commit to the detection.

Recovery from Deadlock

- ***Process Termination***
  - Abort all deadlocked processes:
    - Fast
    - A lot of process work is lost.
  - Abort one deadlocked process at a time and check for deadlocks again:
    - More work to resolve a deadlock.
    - Better in terms of process work.
    - What is a good order to abort processes?
- ***Resource Preemption***
  - what is a good way to select a victim
  - How can we rollback and then recover from preemption?
  - How can we protect from starvation

  What can the system do if it discovers a deadlock?

An Integrated Deadlock Approach:

- *Group Resources into a number of different classes and order them.*
- *Use prevention of circular wait to prevent deadlock between resource classes.*
- *Use the best approach for each class to handle deadlocks within each class.*

4. (a) Discuss paging system for memory management in details. Discuss its advantages and disadvantages.

Ans. External fragmentation: Permit the logical address space of the process to be non contiguous, allowing a process to be allocated physical memory whenever the later is available.

Paging is a solution.

Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.

Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes).

Divide logical memory into blocks of same size called pages.

Keep track of all free frames.

To run a program of size n pages, need to find n free frames and load program.

Set up a page table to translate logical to physical addresses.

Internal fragmentation.

Address generated by CPU is divided into:

Page number (p) – used as an index into a page table which contains base address of each page in physical memory.

Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit.

Page number is an index to the page table.

The page table contains base address of each page in physical memory.

The base address is combined with the page offset to define the physical address that is sent to the memory unit.

The size of a page is typically a power of 2.

512 –8192 bytes per page.

The size of logical address space is 2m and page size is 2n address units.

Higher m-n bits designate the page number; n lower order bits indicate the page offset.

Paging separates user's view of memory and the actual physical memory.

User program views memory as single contiguous space.

In fact, user program is scattered throughout physical memory.

OS maintains copy of the page table for each process. This copy is used to translate logical addresses to physical addresses.

With paging we have no external fragmentation.

We may have some internal fragmentation.

In general, page sizes of 2K or 4K bytes.

Page table is kept in main memory.

Ex: address space = 232 to 264

Page size= 212

Page table= 232 / 212=220

If each entry consists of 4 bytes, the page table size = 4MB.

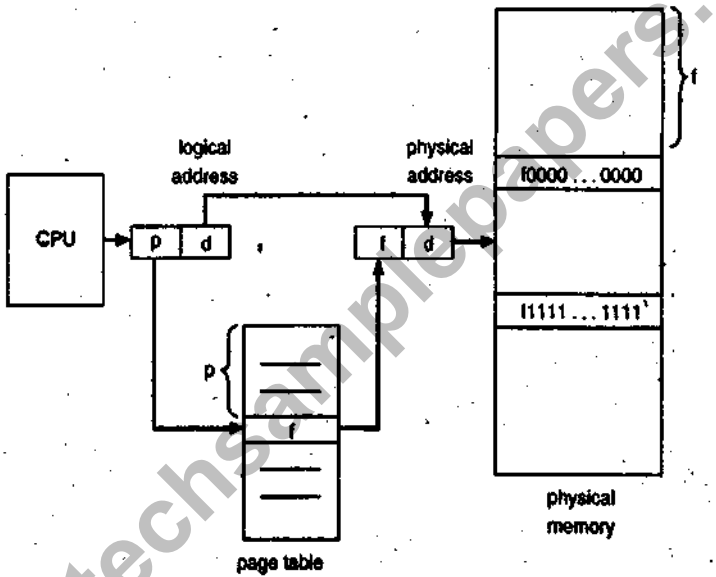
Page-table base register (PTBR) points to the page table.

Page-table length register (PRLR) indicates size of the page table.

In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

Memory access is slowed by a factor of 2.

The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs).



Q.4. (b) Discuss about following in details :

(i) Demand Paging

(ii) Thrashing

Ans. (i) Demand paging: It provides an illusion to the user that user has a very large amount of main memory for execution of his programs. It allows the execution of programs while they are there partially in memory.

Paging system with swapping.

When we execute a process we swap into memory.

For demand paging, we use lazy swapper.

Never swaps a page into memory unless required.

Bring a page into memory only when it is needed.

Less I/O needed

Less memory needed

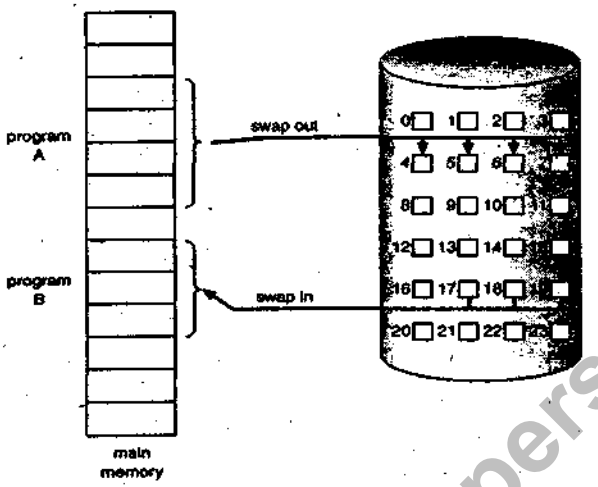
Faster response

More users

Page is needed ⇒ reference to it

invalid reference ⇒ abort

not-in-memory bring to memory



If a process does not have "enough" pages, the page-fault rate is very high. This leads to:

low CPU utilization.

operating system thinks that it needs to increase the degree of multiprogramming.

another process is added to the system.

Thrashing is High paging activity.

(ii) Thrashing: A process is spending more time in swapping pages in and out.

If the process does not have # of frames equivalent to # of active pages, it will very quickly page fault.

Since all the pages are in active use it will page fault again.

Causes of thrashing: OS monitors CPU utilizationIf it is low, increases the degree of MPL

Consider that a process enters new execution phase and starts faulting.

It takes pages from other processesSince other processes need those pages, they also fault, taking pages from other processes.

The queue increases for paging device and ready queue empties CPU utilization decreases.

Solution: provide process as many frames as it needs.

But how we know how many frames it needs ?

Locality model provides hope.Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Enormous virtual memory can be provided on a smaller physical memory.

Two major problems are solved to implement demand paging

Frame allocation algorithm: If multiple processes exist in memory we have to decide the number frames for each process

Q.4.(c) (i) What do you understand by page replacement?Name the algorithm available for page replacement.

Ans. Page replacement algorithm: We have to select a frame that is to be replaced. Want lowest page-fault rate.

Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.

In all our examples, the reference string is

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Address reference divided by page size.

If the address reference is 0432 and page size is 100 then the reference number is 0432/100=4

Some page replacement algorithms are FIFO, Optimal, LRU etc.

(ii) How many page faults occur for optimal page replacement algorithm with following reference string for 4 page frames: 1,2,3,4,5,3,4, 1,6,7,8,7,8,9,7,8,9,5,4,5,4,2

Ans. The no of page faults are 10 for optimal page replacement policy.

Q.5. Attempt any four of the following:

(a) Explain the file system & its function & how system calls are related with file systems.

Ans. File System:

Function: main secondary data storage; also permanent

Extreme speed bottleneck!

Capacity not a problem nowadays: 40 GB disks even for PC.

But backup becoming a problem.

Logical view (view of programmer):

Have a tree structure of _les together with read/write operation and creation of directories

Physical view:

Just a sequence of blocks, which can be read and written

OS has to map logical view to physical view must impose tree structure and assign blocks for each _le

Two possibilities:

_ Linked list: Each block contains pointer to next Problem: random access costly: have to go through whole _le.

_ Indexed allocation: Store pointers in one location: so-called index block. (cf. page table).

To cope with vastly di_ering _le sizes, may introduce indirect index blocks.

File structure

Logical storage unit

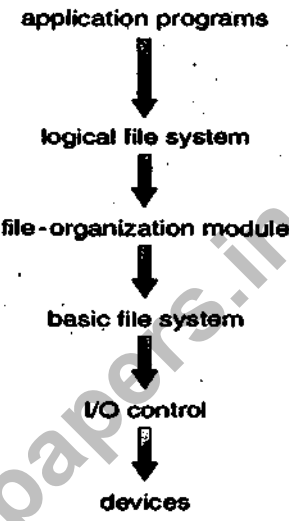
Collection of related information

File system resides on secondary storage (disks).

Explain the file system and its function .

File system organized into layers.

File control block – storage structure consisting of information about a file.



Q.5. (b) Write short notes on following.

(i) Memory mapped I/O

(ii) Direct access method for files

Ans. (i) Memory-mapped I/O (MMIO) and **port I/O** (also called **port-mapped I/O (PMIO)** or **isolated I/O**) are two complementary methods of performing input/output between the CPU and peripheral devices in a computer. Another method, not discussed in this article, is using dedicated I/O processors — commonly known as channels on mainframe computers — that execute their own instructions.

Memory-mapped I/O (not to be confused with memory-mapped file I/O) uses the same address bus to address both memory and I/O devices, and the CPU instructions used to access the memory are also used for accessing devices. In order to accommodate the I/O devices, areas of the CPU's addressable space must be reserved for I/O. The reservation might be temporary — the Commodore 64 could bank switch between its I/O devices and regular memory — or permanent. Each I/O device monitors the CPU's address bus and responds to any of the CPU's access of device-assigned address space, connecting the data bus to a desirable device's hardware register.

Port-mapped I/O uses a special class of CPU instructions specifically for performing I/O. This is generally found on Intel microprocessors, specifically the IN and OUT instructions which can read and write one to four bytes (outb, outw, outl) to an I/O device. I/O devices have a separate address space from general memory, either accomplished by an extra "I/O" pin on the CPU's physical interface, or an entire bus dedicated to I/O. Because the address space for I/O is isolated from that for main memory, this is sometimes referred to as isolated I/O.

A device's direct memory access (DMA) is not affected by those CPU-to-device communication methods, especially it is not affected by memory mapping. This is because by definition, DMA is a memory-to-device communication method that bypasses the CPU.

Hardware interrupt is yet another communication method between CPU and peripheral devices. However, it is always treated separately for a number of reasons. It is device-initiated, as opposed to the methods mentioned above, which are CPU-initiated. It is also unidirectional, as information flows only from device to CPU. Lastly, each interrupt line carries only one bit of information with a fixed meaning, namely "an event that requires attention has occurred in a device on this interrupt line".

(ii) Direct access method: Direct access is based on a disk model of a file. For direct access, the file is viewed as a numbered sequence of block or records. A direct-access file permits random blocks to be read or written. After block 18 has been read, block 57 could be next and then block 3. There are no restrictions on the order of reading and writing for a direct access file. Direct access files are of great use for intermediate access to large amounts of information. The file operations must be modified to include the block number as a parameter. It works like "read n", where n is the block number rather than "read next". Similarly, it writes with "write, n" rather that "write next". An alternative approach retains "read next" and "write next". It adds an operation "position file to n" where n is the block number. Then we would issue the command "position to n" and then "read next". All operating systems maintain sequential as well as direct access for files. Some systems allow only sequential file access. Others allow only direct access. Some systems require that a file should be defined as sequential or direct when it is created. Such a file can be accessed only in a manner defined at the time of its declaration.

Direct Access

read n
write n
position to n
read next
write next
rewrite n

n = relative block number

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

Q.5. (c) Write down the criteria for selection of disk scheduling algorithm.

Ans. Criteria for selection of disk scheduling algorithm.

Total no of head movements. We want minimum movements.

Ease of implementation of the algorithm.

Fairness should be there.

No starvation problem should be there.