Differentiate between Semaphore and Monitor.

Features	Semaphore	Monitor
Definition	A semaphore is an integer variable that allows many processes in a parallel system to manage access to a common resource like a multitasking OS.	It is a synchronization process that enables threads to have mutual exclusion and the wait() for a given condition to become true.

Syntax	// Wait Operation
	wait(Semaphore S) {
	while (S<=0);
	S--;
	}
	// Signal Operation
	signal(Semaphore S) {
	S++;
	}
		monitor {
		//shared variable declarations
		data variables;
		Procedure P1() { ... }
		Procedure P2() { ... }
		.
		.
		.
		Procedure Pn() { ... }
		}

Basic	Integer variable	Abstract data type
Access	When a process uses shared resources, it calls the wait() method on S, and when it releases them, it uses the signal() method on S.	When a process uses shared resources in the monitor, it has to access them via procedures.
Action	The semaphore's value shows the number of shared resources available in the system.	The Monitor type includes shared variables as well as a set of procedures that operate on them.
Condition Variable	No condition variables.	It has condition variables.

Differentiate between Safe state and Unsafe state with respect to deadlock.

Safe State	Unsafe State
A safe state refers to a system state where the allocation of resources to each process ensures the avoidance of deadlock.	An unsafe state implies a system state where a deadlock may occur.
The successful execution of all processes is assured	The successful completion of all processes is not assured
Risk of a deadlock is low	Risk of deadlock is high.
The system attains a safe state when a suitable sequence of resource allocation enables the successful completion of all processes.	The system attains unsafe state when no sequence of resource allocation ensures the successful execution of all processes.

Define Race-Condition. How can Race-Condition be eliminated/resolved? Discuss the various technique to resolve the race-condition.

A race condition is a problem where two or more processes or threads are executing concurrently. The outcome of their execution depends on the order in which they are executed. In a race condition, the exact timing of events is unpredictable, and the outcome of the execution may vary based on the timing. This can result in unexpected or incorrect behaviour of the system.

For example:

If two threads are simultaneously accessing and changing the same shared resource, such as a variable or a file, the final state of that resource depends on the order in which the threads execute. If the threads are not correctly synchronized, they can overwrite each other's changes, causing incorrect results or even system crashes.

Techniques to resolve race condition:

Use locks or semaphores: Use synchronization mechanisms such as locks or semaphores to ensure that only one process or thread can access a shared resource at any given time.

Atomic operations: Use atomic operations, which are operations that cannot be interrupted, to manipulate shared data.

Avoid global variables: Avoid using global variables because they can be accessed and modified by multiple processes or threads simultaneously.

Use message passing: Use message passing instead of shared memory to communicate between processes or threads. This ensures that only one process or thread can access a particular message at a time.

Use thread-safe libraries: Use thread-safe libraries that are designed to prevent race conditions.

Design for concurrency: When designing your application, consider concurrency issues from the start. Make sure that your design can handle multiple processes or threads accessing shared resources.

Discuss the various data-structures used in Process Control Block (PCB).

- **Process state.** The state may be new, ready running, waiting, halted, and so on.
- **Program counter.** The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers**. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information
- **CPU-scheduling information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information.** This information may include such information as the value of the base and limit registers, 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.

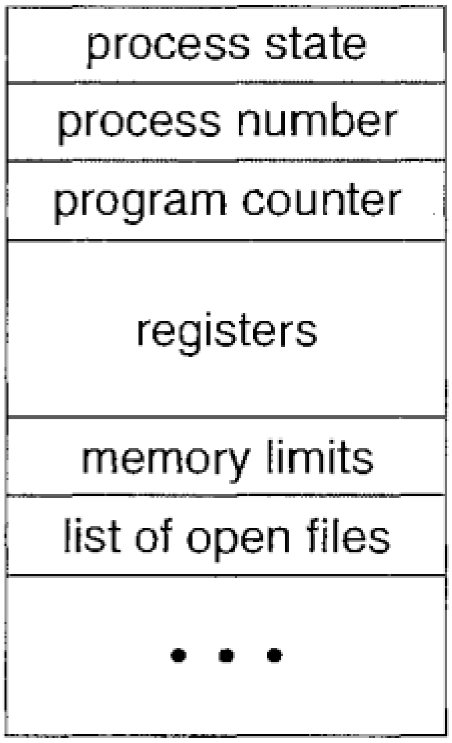


Figure 3.3 Process control block (PCB).

Explain the Process State Model and discuss the waiting state of a process.

- **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.

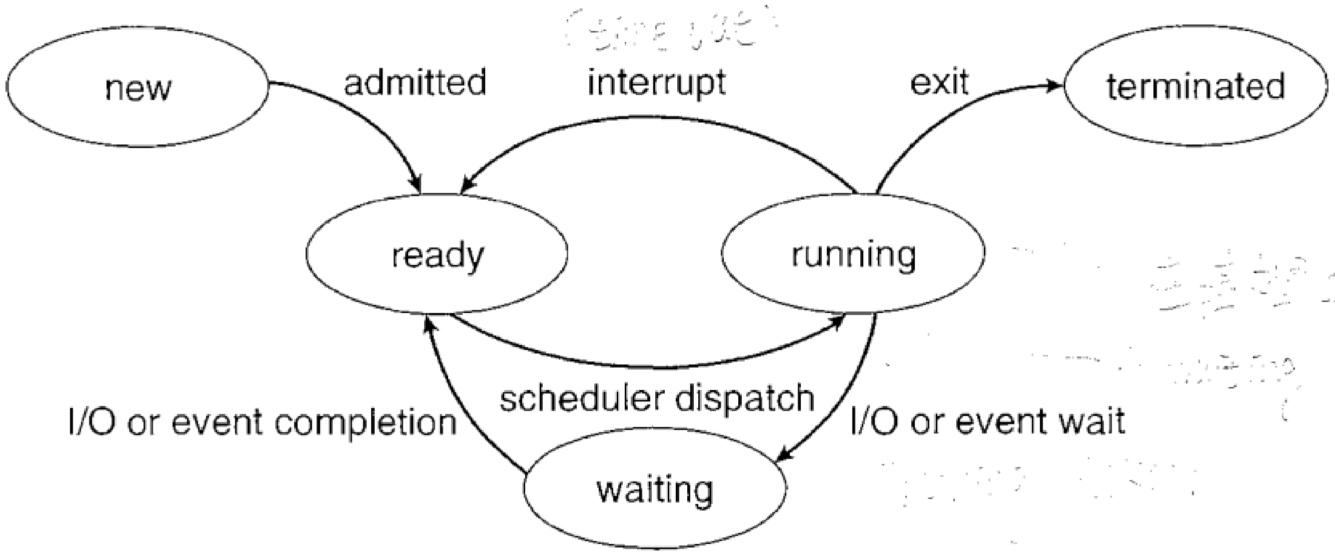


Figure 3.2 Diagram of process state.

Waiting State: The waiting state is a state in which the process is waiting for an event to occur. This event could be the completion of an I/O operation, the availability of a resource, or the arrival of a message. While the process is in the waiting state, it is not using any CPU time and is not occupying any memory.

The waiting state is typically implemented using a queue. The queue contains all of the processes that are currently waiting for an event to occur. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. The list of processes waiting for a particular I/0 device is called a device queue.

Discuss and explain Banker's algorithm to avoid deadlock. Provide an example to understand the algorithm.

Banker's algorithm consists of a Safety algorithm and a Resource request algorithm.

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

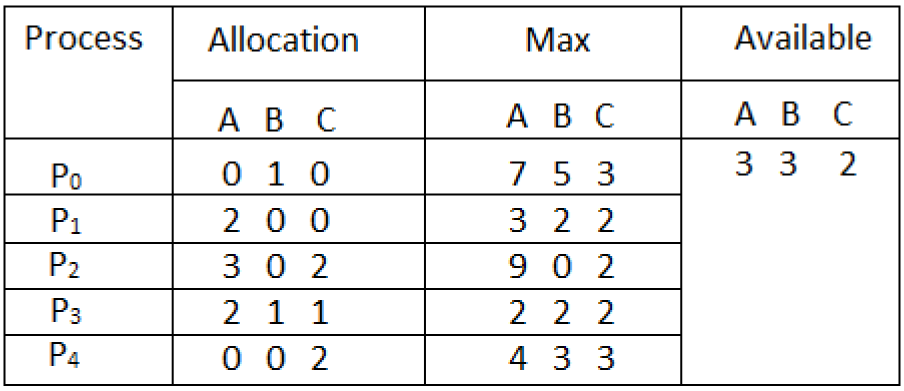
```
1) Let Work and Finish be vectors of length 'm' and 'n' respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4….n
2) Find an i such that both
a) Finish[i] = false
b) $Need_i$ <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)
4) if Finish [i] = true for all i
then the system is in a safe state
```

Resource-Request Algorithm

Let $Request_i$ be the request array for process P_i. $Request_i$ [j] = k means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

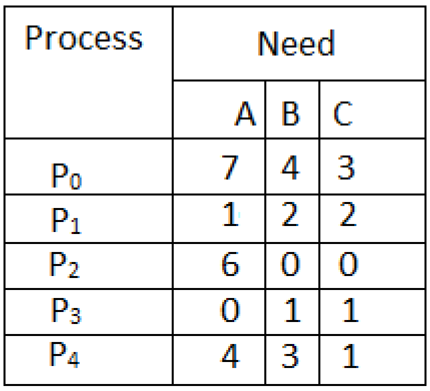
```
1) If $Request_i$ <= $Need_i$
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2) If $Request_i$ <= Available
Goto step (3); otherwise, $P_i$ must wait, since the resources are not available.
3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as
follows:
Available = Available – Requesti
$Allocation_i$ = $Allocation_i$ + $Request_i$
$Need_i$ = $Need_i$– $Request_i$
```

Example:
Considering a system with five processes P_0 through P_4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:



Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

What will be the content of the Need matrix?
Need [i, j] = Max [i, j] – Allocation [i, j]
So, the content of Need Matrix is:



Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Discuss the following terminologies: (a) Segmentation (b) Thrashing (c) Buffering (d) Demand Paging

Segmentation:

Segmentation divides processes into smaller subparts known as **modules**. The divided segments need not be placed in contiguous memory. Since there is no contiguous memory allocation, internal fragmentation does not take place. The length of the segments of the program and memory is decided by the purpose of the segment in the user program.

Segmentation in OS can be divided into two types:

1. **Virtual Memory Segmentation**: Virtual Memory Segmentation divides the processes into **n** number of segments. All the segments are not divided at a time. Virtual Memory Segmentation may or may not take place at the run time of a program.
2. **Simple Segmentation**: Simple Segmentation also divides the processes into **n** number of segments but the segmentation is done all together at once. Simple segmentation takes place at the run time of a program. Simple segmentation may scatter the segments into the memory such that one segment of the process can be at a different location than the other(in a noncontinuous manner).

Thrashing:

A process is said to be thrashing if the CPU spends more time serving page faults than executing the pages. This leads to low CPU utilization and the Operating System in return tries to increase the degree of multiprogramming.

Three main causes of thrashing are:

1. High degree of Multiprogramming.
2. Less number of frames compared to the processes required.
3. The process scheduling scheme which swaps in more processes when CPU utilization is low.

Thrashing can be eliminated by:

- Adjusting the swap file size
- Increasing the amount of RAM
- Decreasing the number of applications running on the computer.
- Replace programs with a more efficient alternative

Buffering:

Buffering is a method of storing data in a buffer or cache temporarily, this buffered data then can be accessed more quickly as compared to the original source of the data.

The three reasons for buffering are:

- Buffering creates a synchronization between two devices having different processing speed.
- Buffering is also required in cases where two devices have different data block sizes.
- Buffering is also required to support copy semantics for application I/O operations.

Types of buffering:

- **Single Buffering:** When a process issues an I/O request, the operating system assigns a buffer (or cache) in the system potion of the main memory to the operation. Then, the input transfers are made to the buffer and are moved to the user space when needed.
- **Double Buffering:** A process can transfer data to or from one buffer while the operating system removes or fills the other.
- **Circular Buffering:** When more than two buffers are used, then it is called circular buffering.

Demand Paging

Demand paging is identical to the paging system with swapping. In demand paging, a page is delivered into the memory on demand i.e., only when a reference is made to a location on that page. Demand paging combines the feature of simple paging and implement virtual memory as it has a large virtual memory. Lazy swapper concept is implemented in demand paging in which a page is not swapped into the memory unless it is required.

Demand Paging	Segmentation
In demand paging, the pages are of equal size.	While in segmentation, segments can be of different size.
Page size is fixed in the demand paging.	Segment size may vary in segmentation as it grants dynamic increase of segments.
It does not allows sharing of the pages.	While segments can be shared in segmentation.
In demand paging, on demand pages are loaded in the memory.	In segmentation, during compilation segments are allocated to the program.
Page map table in demand paging manages record of pages in memory.	Segment map table in segmentation demonstrates every segment address in the memory.
It provides large virtual memory and have more efficient use of memory.	It provides virtual memory and maximum size of segment is defined by the size of memory.
Virtual memory is divided into pages.	Virtual memory is divided into segments.
Pages can be of equal or variable size.	Segments are variable in size.
Pages are allocated dynamically.	Segments are allocated at the process start.
Memory access is Page-level protection.	Memory access is segment-level protection.
Memory wastage is Internal fragmentation.	Memory wastage is external fragmentation.

Explain the various methods of Disk allocation strategy for allocating files in secondary memory.

Same as File Allocation methods

Write short notes on any two of the following: (a) Differences between Linux and Windows. (b) Recovery from deadlock. (c) System calls.

Parameters	Linux	Windows
Case Sensitivity	The file system in Linux is very case-sensitive.	The file system in Windows is not case-sensitive.
Cost Incurred	Linux is free to use for everyone.	Windows do not come free for any user.
Open Source	It is open source.	It is not open source.
Type of Kernel Used	Linux utilizes the monolithic kernel.	Windows uses the micro-kernel.
Path Separator	The path separator that Linux uses is Slash.	The path separator that Windows uses is backward slash.
Efficiency	In the case of operations, Linux is way more efficient than Windows.	For operations, Windows are comparatively way less efficient than Linux.
Security	It is more secure than Windows OS.	It provides much less security to its users than Linux.
Uses in Hacking	People generally use Linux for the systems that are hacking-based.	Windows is not a very efficient OS for hacking purposes as compared to Linux.

Deadlock Recovery

Process termination

1. **Kill a Process**
   - In this process, the process responsible for the Deadlock is terminated. However, selecting which process to kill can be difficult. As a result, the operating system primarily kills the process that no longer works.

2. **Kill all Processes**
   - It is not appropriate to kill all the processes. However, when a critical situation arises, this approach may be suitable.
   - Putting an end to all processes reduces the system's efficiency, so we must start all the processes again.

Resource Preemption

1. **Preemption**
    - Here, we transfer resources from one process to the process that needs them to finish its execution, and once that process is done, the resource is released.
    - The selection of resources is complex, and grabbing the resources is also challenging. Several states are necessary for the system to reach the Deadlock.
2. **RollBack**
    - A rollback of the system to the earlier safe state is possible using the operating system. This requires the implementation of checkpoints in every state.
    - When we identify Deadlock, we must roll back all allocations and return to the previous safe state.

System Calls

A system call is a mechanism used by programs to request services from the operating system (OS). A system call is initiated by the program executing a specific instruction, which triggers a switch to kernel mode, allowing the program to request a service from the OS. The OS then handles the request, performs the necessary operations, and returns the result back to the program.

Features of System Calls

1. **Interface:** System calls provide a well-defined interface between user programs and the operating system.
2. **Protection:** System calls are used to access privileged operations that are not available to normal user programs
3. **Kernel Mode:** When a system call is made, the program is temporarily switched from user mode to kernel mode.
4. **Context Switching:** A system call requires a context switch, which involves saving the state of the current process and switching to the kernel mode to execute the requested service.
5. **Error Handling:** System calls can return error codes to indicate problems with the requested service.
6. **Synchronization:** System calls can be used to synchronize access to shared resources, such as files or network connections.

There are mainly 5 types of system calls available:

- Process Control
- File Management
- Device Management
- Information Maintenance
- Communication