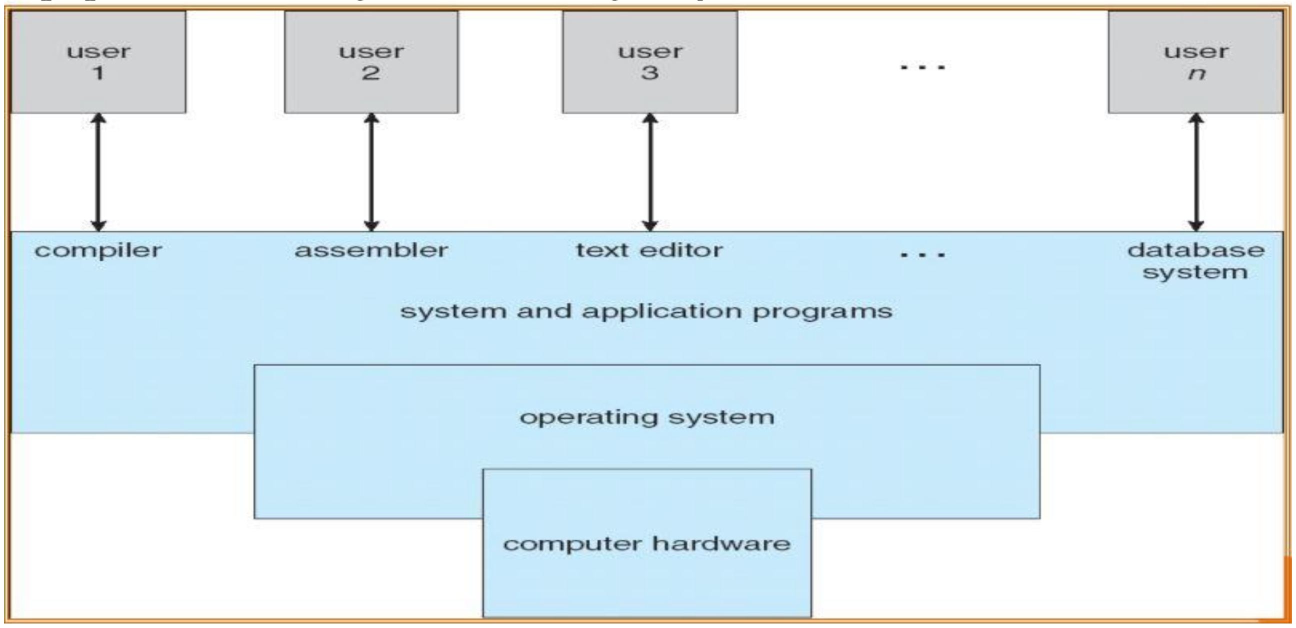
JGi JAIN DEEMED-TO-BE UNIVERSITY | FACULTY OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering

Global Campus **,Jakkasandra Post, Kanakapura Taluk, Ramanagara District, Pin Code: 562 112**

Q. No	Questions
	Module 1: Introduction to operating System
1	With a neat diagram explain components of computer system. Discuss the different views of OS.

Computer System Components
 The Hardware consists of memory, CPU, ALU, I/O devices, peripherals devices &amp; storage devices.
 The application program mainly consisted of word processors, spread sheets, compilers & web browsers defines the ways in which the resources are used to solve the problems of the users.
 The OS controls &amp; co-ordinates the use of hardware among various application program for various users.
The following figure shows the conceptual view of a computer system



1. User Views: - The user view of the computer depends on the interface used.
i. Some users may use PC's. In this the system is designed so that only one user can utilize the resources and mostly for ease of use where the attention is mainly on performances and not on the resource utilization.
ii. Some users may use a terminal connected to a mainframe or minicomputers.
iii. Other users may access the same computer through other terminals. These users may Share resources and exchange information. In this case the OS is designed to maximize Resource utilization- so that all available CPU time, memory &amp; I/O are used efficiently.
iv. Other users may sit at workstations, connected to the networks of other workstation and Servers. In this case OS is designed to compromise between individual visibility &amp; Resource utilization.

2. System Views:-
OS is a resource allocator

Manages all resources
Decides between conflicting requests for efficient and fair resource use
OS is a control program
Controls execution of programs to prevent errors and improper use of the computer

2 Discuss the various components of an Operating System in detail.

OR

Mention the activities connected with process management, memory management, and device management and file management.

1. Process Management

   The OS is responsible for the following activities in connection with process management.

 Creating and deleting both user and system processes
 Suspending and resuming processes
 Providing mechanism for process synchronization
 Providing mechanism for process communication
 Providing mechanisms for deadlock handling

2. Main Memory Management

   The OS is responsible for the following activities in connection with memory management.

    Keeping track of which part of memory are currently being used and by whom.
    Deciding which processes and data to move into and out of memory.
    Allocating and DEallocating memory space as needed.

3. File Management

   The OS is responsible for the following activities.
    Creating and deleting files
    Creating and deleting directories to organize files.
    Supporting primitives for manipulating files and directories.
    Mapping files into secondary storage
    Backing up files on stable (non-volatile) storage media.

DEVICE MANAGEMENT

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

4. I/O System Management
5. Secondary Management

6. Networking
7. Protection System
8. Command-Interpreter System

3 What are the goals of an operating system? Explain two sets of operating system services that are helpful to user as well as efficient operation of the system.

Operating System Goals
 Make the computer system convenient to use.
 Efficient operation of the computer system.
 Execute user programs and make solving user problems easier.
 Use the computer hardware in an efficient manner

Provides functions that are helpful to the user:

- User interface - Almost all operating systems have a user interface (UI), Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch

- Program execution - The system must be able to load a program into memory and to run that program, end execution

Another set of OS functions exists for ensuring the efficient operation of
the system itself via resource sharing

- Resource allocation - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them

- Accounting - To keep track of which users use how much and what kinds of computer resources.

4 What are system calls? Explain the different categories of system calls.

• System provides interface between the process &amp; the OS.
• The calls are generally available as assembly language instruction & certain system allow system calls to be made directly from a high level language program.
• Several languages have been defined to replace assembly language program.
• A system call instruction generates an interrupt and allows OS to gain control of the processors.

• System calls occur in different ways depending on the computer. Sometime more information is needed to identify the desired system call. The exact type & amount of information needed may vary according to the particular OS & call.

System calls may be grouped roughly into 5 categories:-

Process Control

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

FILE MANAGEMENT

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

DEVICE MANAGEMENT

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

INFORMATION MAINTENANCE

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

COMMUNICATION:-

 create, delete communication connection
 send, receive messages
 transfer status information
 attach or detach remote devices

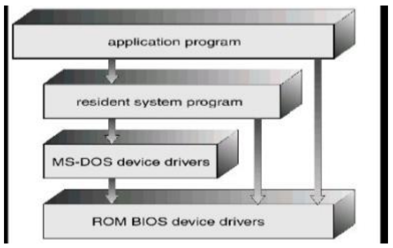
5 What is an Operating System? Discuss different Operating system structures along with their advantages and disadvantages.

- Operating system is large and complex, so a common approach is to partition the task into small components to function properly, 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 function.

MS-DOS System Structure

 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
 It was written to provide the most functionality in the least space (because of the limited hardware on which it ran), so it was not divided into modules.
 UNIX is another system that was initially limited by hardware functionality.
 It consists of two separable parts: the kernel and the system programs.
 The kernel is further separated into a series of interfaces and device drivers, which were added and expanded over the years as UNIX evolved.
 The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.



Advantages
 The main advantage is simplicity of construction and debugging.
 Modularity
Disadvantages
 The main difficulty is defining the various layers.
 The main disadvantage is that the OS tends to be less efficient than other implementations.
 Slower

Microkernel System Structure

 As the UNIX operating system expanded, the kernel became large and difficult to manage.
 In the mid-1980s, researchers developed an os called Mach that modularizes the kernel using the microkernel approach.
 Moves as much from the kernel into “user” space. This results in a small kernel.
 Communication takes place between user modules using message passing.
Benefits/Advantages:
o easier to extend a microkernel
o easier to port the operating system to new architectures
o more reliable (less code is running in kernel mode)
o more secure

Disadvantage:
 Main disadvantage is poor performance due to increased system overhead from message passing.
 Microkernels provide minimal process and memory management, in addition to a communication facility between the client program and the various services that are also running in user space.
 Ex: UNIX, Apple MacOS, QNX.
 Windows NT uses a hybrid structure.
 Part of the Windows NT architecture uses layering.
 Windows NT is designed to run various applications, including Win32, 0S/2, and POSIX.
 The kernel coordinates the message passing between client applications and application servers.

Virtual Machines

A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware. A virtual machine provides an interface identical to the underlying bare hardware. The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory. The resources of the physical computer are shared to create the virtual machines. CPU scheduling can create the appearance that users have their own processor.

Advantages:
The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.
 System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
 Multiple OS can exist on simultaneously on the same machine isolated from each other.
 Easy maintenance, application provisioning, availability and convenient recovery
 Consumes less power and physical space.
Disadvantages:
 The virtual machine concept is difficult to implement.
 Due to isolation, no direct sharing of resources is permitted.

Java Virtual Machine

Java is a very popular object-oriented language introduced by Sun Microsystems in late 1995. In addition to a language specification and a large API library, Java also provides a specification for a JVM. Java program consists of one or more classes. For each Java class, the Java compiler produces an platform- neutral bytecode output (. class) file that will run on any implementation of the JVM.

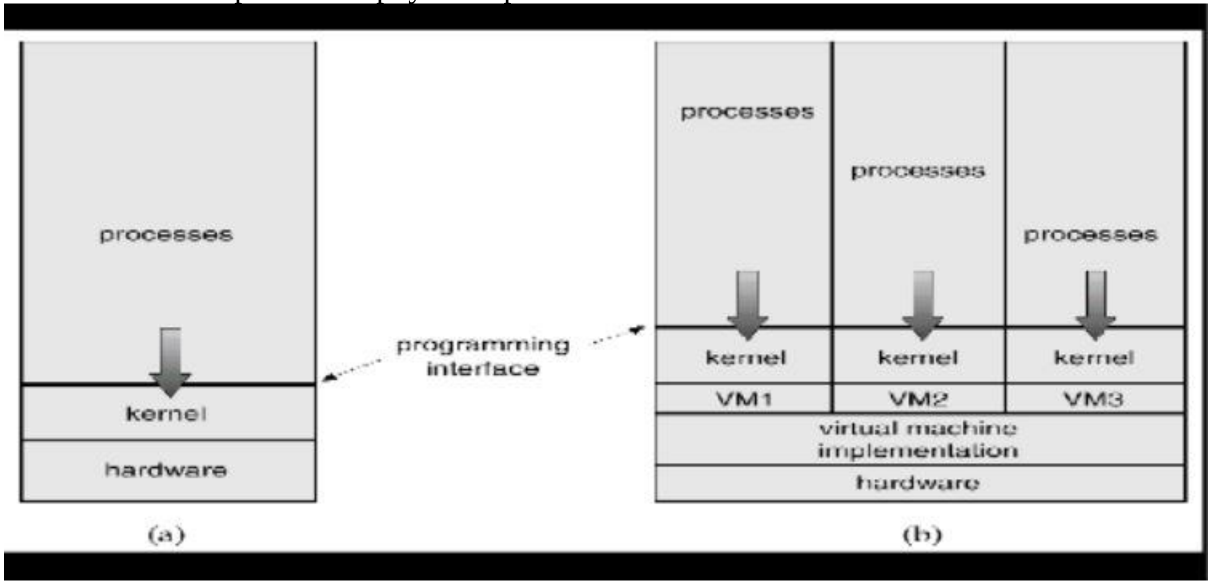
Advantages:
 Platform independence: Written once and executed on multiple platforms
 Security: JVM has built-in security features.
Disadvantage:
 Programs run on JVM takes longer time to execute.

6 What is a virtual machine? With a neat diagram discuss the implementation of virtual machine along with its benefits.

Virtual Machines

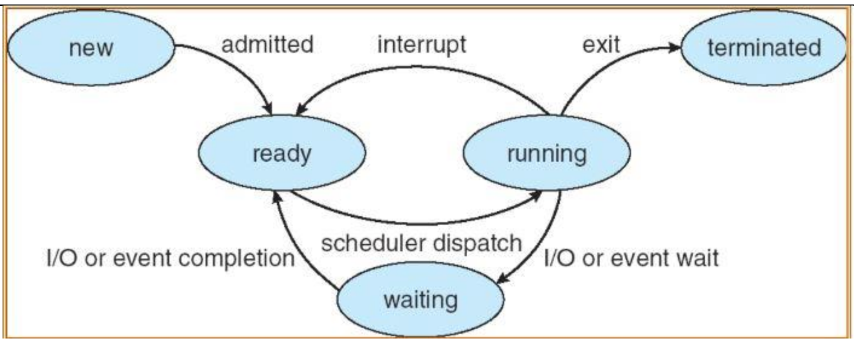
A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware. A virtual machine provides an interface identical to the underlying bare hardware. The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory. The resources of the physical computer are shared to create the virtual machines. CPU scheduling can create the appearance that users have their own processor.

Advantages: The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.
 System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
 Multiple OS can exist on simultaneously on the same machine isolated from each other.
 Easy maintenance, application provisioning, availability and convenient recovery
 Consumes less power and physical space.



7 What is a process? With the help of a state-transition diagram, explain the various states of a process.

A process is a program in execution. A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers.



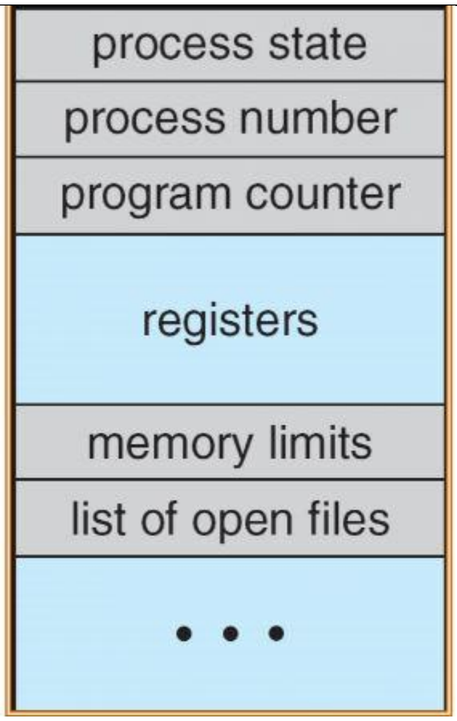
As a process executes, it changes state

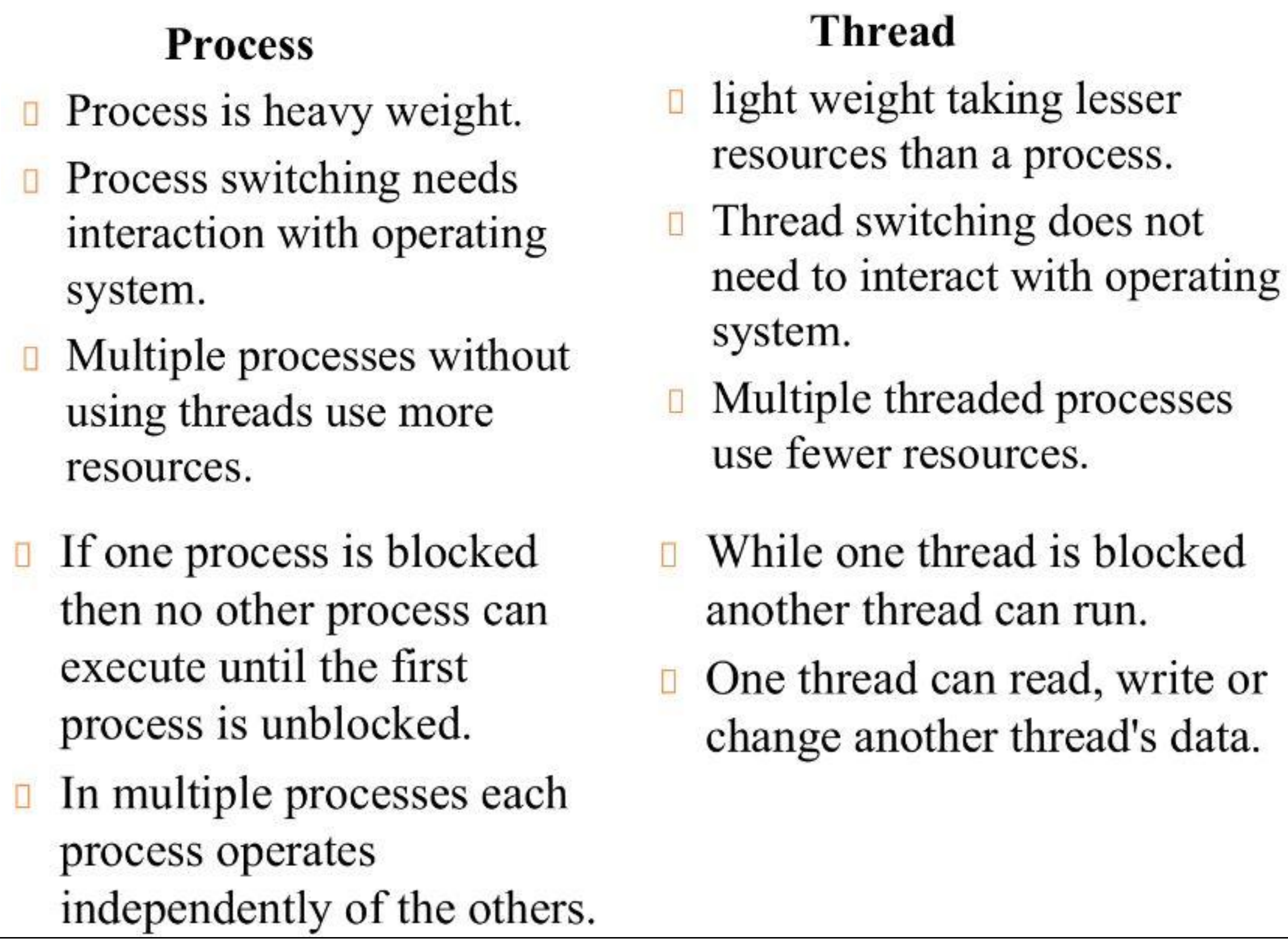
- new: The process is being created
- running: Instructions are being executed
- waiting: The process is waiting for some event to occur
- ready: The process is waiting to be assigned to a processr
- terminated: The process has finished execution
- Only one process can be running on any processor at any instant, although many processes may be ready and waiting

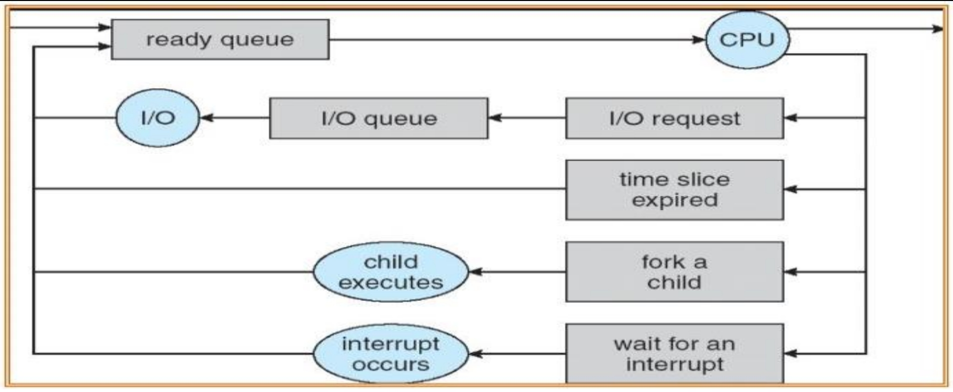
8 What is PCB? List out the contents of PCB with a neat diagram.

Each process is represented in OS by PCB, also known as TCB Task control blocks

- Process state
- Program counter
- CPU registers (accumulators, index registers, stack pointers, and general-purpose registers)
- CPU scheduling information (includes a process priority, pointers to scheduling queues)
- Memory-management information (value of the base and limit registers, the page tables, or the segment tables)
- Accounting information (amount of CPU and real time used, time limits, account numbers, job or process numbers)
- I/O status information (list of I/O devices allocated to this process, a list of open files)

	 process state process number program counter registers memory limits list of open files • • •
9	List out the differences between process and thread. OR Define thread. Mention the benefits of Threads over Processes? Thread: Thread is the segment of a process which means a process can have multiple threads and these multiple threads are contained within a process. Differences Unlike processes, threads are not independent of one another. Unlike processes, all threads can access every address in the task. Unlike processes, thread are design to assist one other. Note that processes might or might not assist one another because processes may originate from different users.

	 **Process** • Process is heavy weight. • Process switching needs interaction with operating system. • Multiple processes without using threads use more resources. • If one process is blocked then no other process can execute until the first process is unblocked. • In multiple processes each process operates independently of the others. **Thread** • light weight taking lesser resources than a process. • Thread switching does not need to interact with operating system. • Multiple threaded processes use fewer resources. • While one thread is blocked another thread can run. • One thread can read, write or change another thread's data.
10	What is a scheduler? With a neat diagram discuss how process scheduling is done by operating system. 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.



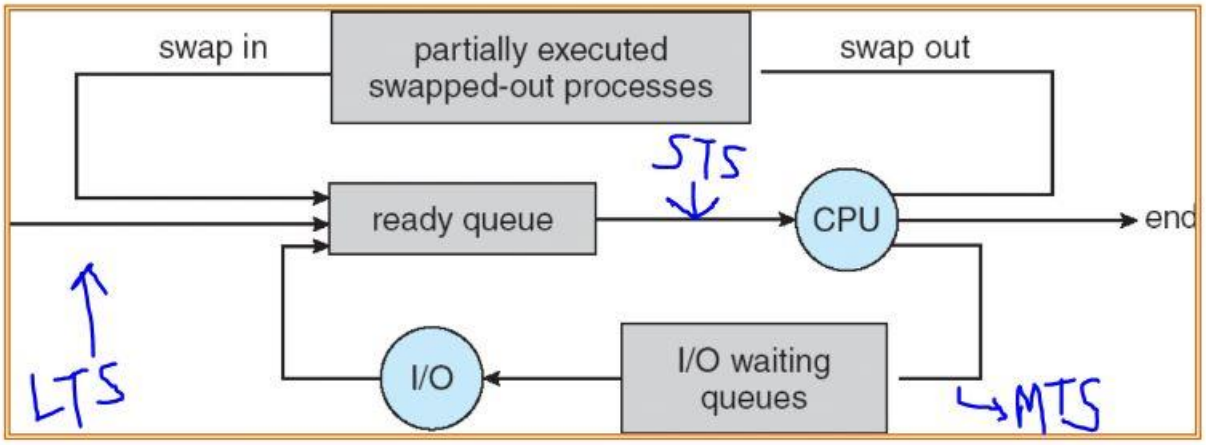
The following are the different types of process scheduling queues.
1. Job queue – set of all processes in the system
2. Ready queue – set of all processes residing in main memory, ready and waiting to execute
3. Device queues – set of processes waiting for an I/O device
4. Processes migrate among the various queues

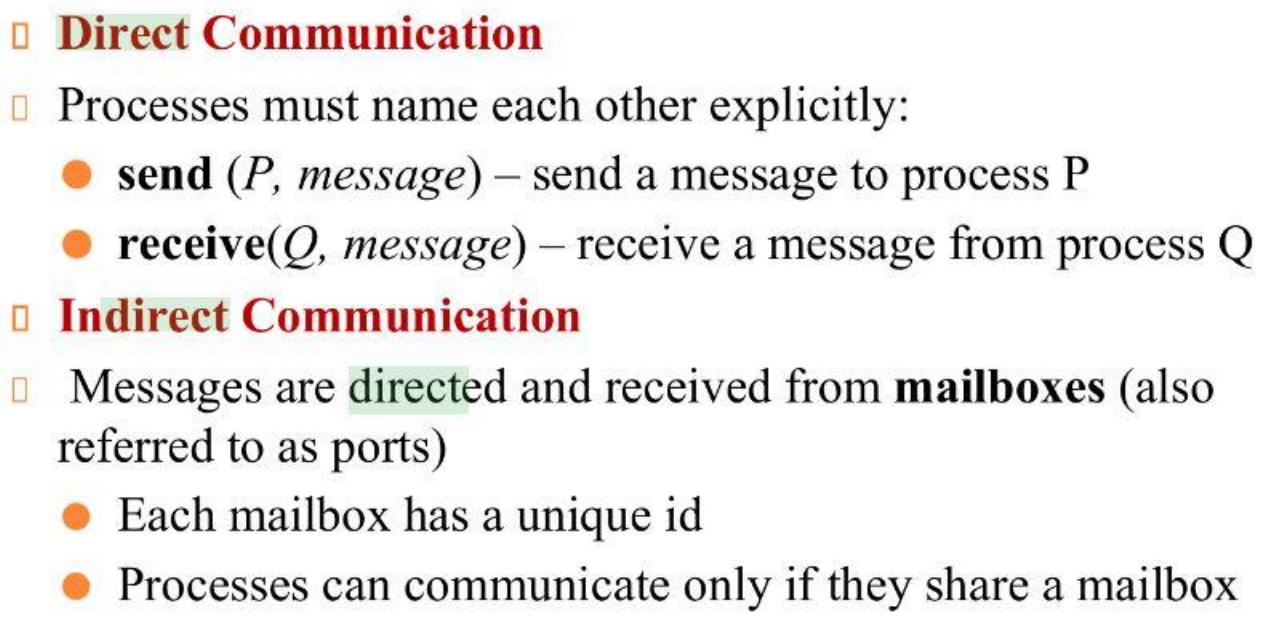
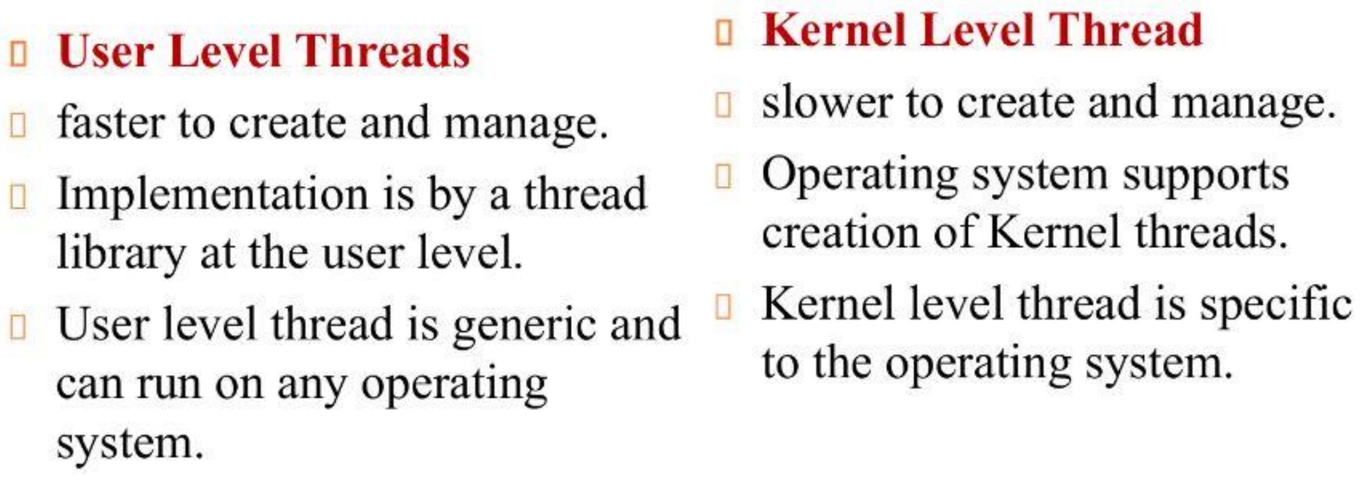
The process scheduling is represented using a queuing diagram. Queues are represented by the rectangular box & resources they need are represented by circles. It contains two queues ready queue & device queues.
Once the process is assigned to CPU and is executing the following events can occur,
a. It can execute an I/O request and is placed in I/O queue.
b. The process can create a sub process &amp; wait for its termination.
c. The process may be removed from the CPU as a result of interrupt and can be put back into ready queue.

A new process is initially put in the ready queue. It waits there until it is selected for execution, or is dispatched. Once the process is allocated the CPU and is executing, one of several events could occur:
• The process could issue an I/O request and then be placed in an I/O queue.
• The process could create a new sub-process and wait for the sub-process&#39;s termination.
• The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.
 A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources de-allocated.

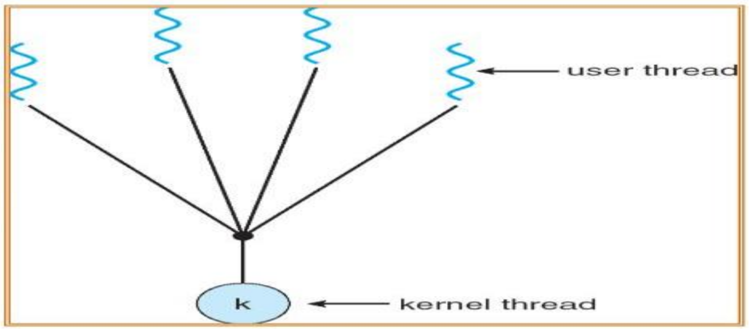
11 Describe the functions of long term, medium term and short term schedulers with a block diagram.

1. Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
2. Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.
3. Medium-term schedulers
   - Short-term scheduler is invoked very frequently (milliseconds) Þ (must be fast)
   - Long-term scheduler is invoked very infrequently (seconds, minutes) (may be slow)
   - The long-term scheduler controls the degree of multiprogramming
   - Processes can be described as either:
    I/O-bound process – spends more time doing I/O than computations, many short CPU bursts

	CPU-bound process – spends more time doing computations; few very long CPU bursts  swap in partially executed swapped-out processes swap out STS ready queue CPU end LTS I/O I/O waiting queues MTS
12	What are the benefits offered by cooperating processes? Explain direct and indirect communications with respect to message passing system. OR What are the benefits offered by cooperating processes? Discuss two fundamental methods of inter process communication. Cooperating process can affect or be affected by the execution of another process (any process that shares data with other processes) **Advantages of process cooperation-** • Information sharing (concurrent access to same resources) • Computation speed-up (If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel) • Modularity (dividing the system functions into separate processes or threads) • Convenience (user may have many tasks on which to work at one time)

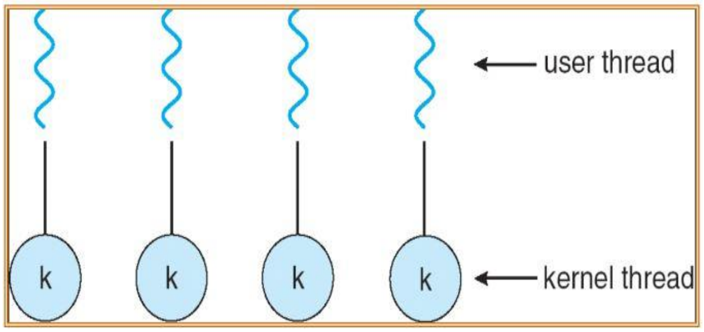
	 **Direct Communication** Processes must name each other explicitly: ● **send** (*P*, *message*) – send a message to process P ● **receive**(*Q*, *message*) – receive a message from process Q **Indirect Communication** Messages are directed and received from **mailboxes** (also referred to as ports) ● Each mailbox has a unique id ● Processes can communicate only if they share a mailbox
13	Discuss the design characteristics of message passing systems. OR Explain direct and indirect communications with respect to message passing system.
14	List out the differences between user level threads and kernel level threads.  **User Level Threads** faster to create and manage. Implementation is by a thread library at the user level. User level thread is generic and can run on any operating system. **Kernel Level Thread** slower to create and manage. Operating system supports creation of Kernel threads. Kernel level thread is specific to the operating system.
15	With a neat diagram discuss various multithreading models along with their advantages and disadvantages. MANY-TO-MANY MODEL

Allows many user level threads to be mapped to a smaller or equal number of kernel threads
Allows the operating system to create a sufficient number of kernel threads corresponding kernel threads can run in parallel on a multiprocessor
Solaris prior to version 9
Windows NT/2000 with the ThreadFiber package



ONE-TO-ONE

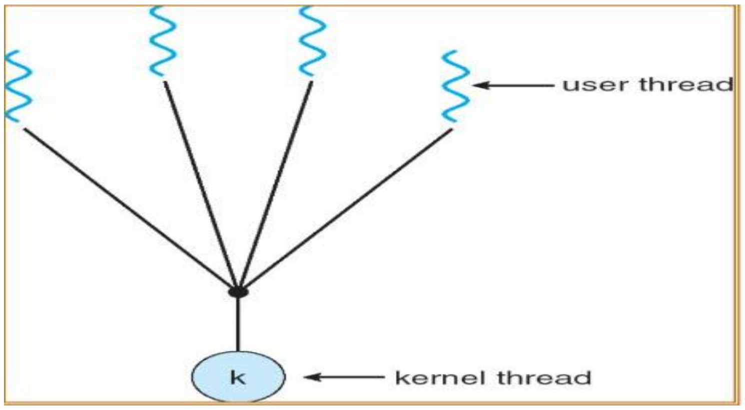
Each user-level thread maps to kernel thread
Provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call
Allows multiple threads to run in parallel on multiprocessors
creating a user thread requires creating the corresponding kernel thread
Examples
Windows NT/XP/2000
Linux
Solaris 9 and later



MANY-TO-ONE

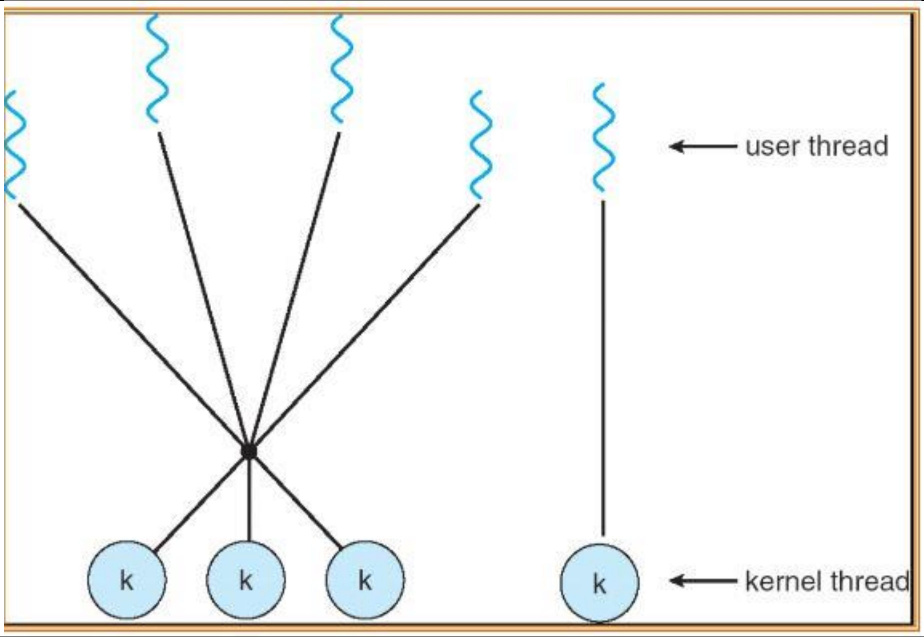
Many user-level threads mapped to single kernel thread
Thread management is done in user space
entire process will block if a thread makes a blocking system

call
only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors
Examples:
Solaris Green Threads
GNU Portable Threads



TWO-LEVEL MODEL

Similar to M:M, except that it allows a user thread to be
bound to kernel thread
Examples
IRIX
HP-UX
Tru64 UNIX
Solaris 8 and earlier



Module 2: CPU Scheduling

1 What is scheduling? Differentiate between pre-emptive and non-pre-emptive scheduling.

Scheduling in operating system is the process of selecting a process from a ready queue. And allotting CPU to this process for execution. The operating system schedules the processes in such a way that the CPU doesn't sit idle. And keeps processing some or the other process.

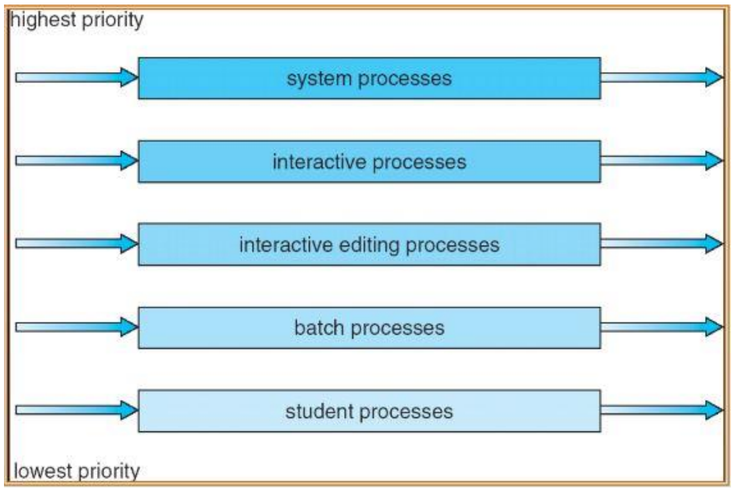
Nonpreemptive Scheduling
A scheduling discipline is nonpreemptive if, once a process has been given the CPU; the CPU cannot be taken away from that process.

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

2 Define a dispatcher. What are the functions of a dispatcher?

Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:

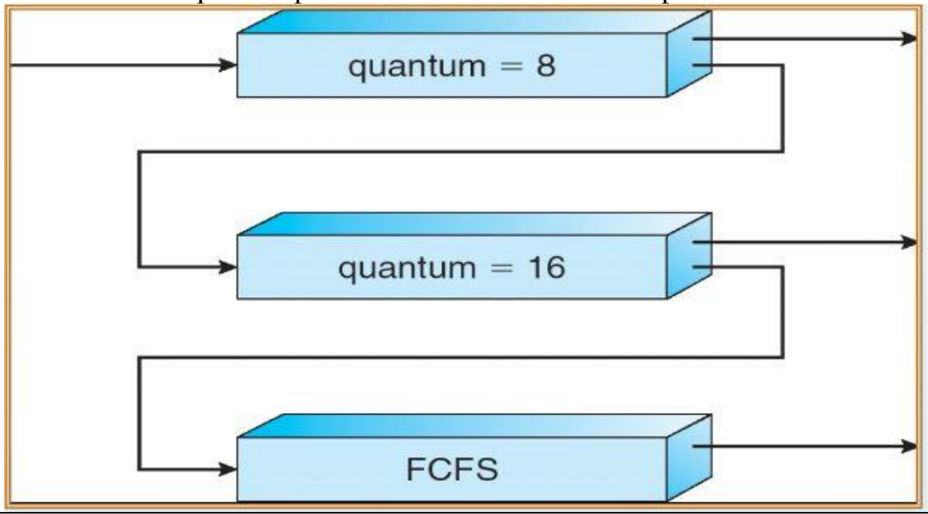
- switching context
- switching to user mode
- jumping to the proper location in the user program to restart that program

	Dispatch latency – time it takes for the dispatcher to stop one process and start another running
3	Discuss the issues in multiple-processor scheduling.
4	Define all scheduling criteria's. Scheduling Criteria 1. CPU utilization – keep the CPU as busy as possible 2. Throughput – # of processes that complete their execution per time unit 3. Turnaround time – amount of time to execute a particular process 4. Waiting time – amount of time a process has been waiting in the ready queue 5. Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
5	With a neat diagram explain multilevel queue scheduling and multilevel feedback queue scheduling.  highest priority system processes interactive processes interactive editing processes batch processes student processes lowest priority Multilevel queue scheduling algorithm partitions the ready queue in several separate queues, for instance In a multilevel queue scheduling processes are permanently assigned to one queues. The processes are permanently assigned to one another, based on some property of the process, such as Memory size Process priority Process type Algorithm choose the process from the occupied queue that has the highest priority, and run that process either Pre-emptive or Non-pre-emptively Each queue has its own scheduling algorithm or policy. Scheduling must be done between the queues • Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.

- Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
- 20% to background in FCFS

Multilevel Feedback Queue Scheduling
Multilevel feedback queue-scheduling algorithm allows a process to move between queues. It uses many ready queues and associates a different priority with each queue.
Multilevel-feedback-queue scheduler defined by the following parameters:

- number of queues
- scheduling algorithms for each queue
- method used to determine when to upgrade a process
- method used to determine when to demote a process
- method used to determine which queue a process will enter when that process needs service



6 For the following set of the process find the average waiting time and average turn around turn using FCFS, SJF, SRTF, pre-emptive priority and Round - Robin (TQ =2ms) scheduling algorithms.

Process	**Arrival Time**	**Burst time**	**Priority**
P1	0	7	3
P2	2	4	1

P3	4	1	4
P4	5	4	2

NOTE: Data like time quantum (TQ), arrival time, burst time, priority and number of process may vary. All or few scheduling algorithms may be asked according to marks.

Process Synchronization

7 Define Race Condition With an example discuss how Race Condition can be avoided?

OR

What is race condition? Discuss Race Condition with an example.

In operating systems, processes that are working together share some common storage (main memory, file etc.) that each process can read and write. When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race conditions.

Concurrently executing threads that share data need to synchronize their operations and processing in order to avoid race condition on shared data. Only one 'customer' thread at a time should be allowed to examine and update the shared variable.
Race conditions are also possible in Operating Systems. If the ready queue is implemented as a linked list and if the ready queue is being manipulated during the handling of an interrupt, then interrupts must be disabled to prevent another interrupt before the first one completes. If interrupts are not disabled than the linked list could become corrupt.

8 Describe critical section problem. List out the three requirements to be met by a solution to the critical section problem.

Consider a system consisting of n processes {P0, P1... Pn}. 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.

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

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

1. Mutual Exclusion - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2.Progress - If no process is executing in its critical section and 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

3.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.

9 Discuss Peterson’s solution for critical section problem.

a classic software-based solution to the critical-section problem known as Peterson’s solution. Because of the way modern computer architectures perform basic machine-language instructions, such as load and store, there are no guarantees that Peterson’s solution will work correctly on such architectures.
Peterson’s solution is restricted to two processes that alternate execution between their critical sections and remainder sections. The processes are numbered Po and Pi. For convenience, when presenting P,-, we use Pj to denote the other process; that is, j equals 1 — i. Peterson&#39;s solution requires two data items to be shared between the two processes:

int turn;
boolean f l a g [2] ;

The variable turn indicates whose turn it is to enter its critical section. That is, if turn == i, then process P; 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. For example, if f lag[i] is true, this value indicates that P; is ready to enter its critical section.

10 Define the hardware instructions TestAndSet () and Swap (). Discuss solution for critical section problem using these instructions.

The definition of the TestAndSet () instruction

```
boolean TestAndSet(boolean *target) {
boolean rv = *target;
*target = TRUE;
return rv;
}
```

Solution using TestAndSet

Shared boolean variable lock., initialized to
false.

Solution;

```
do {
while ( TestAndSet (&lock ))
/* do nothing
// critical section
lock = FALSE;
// remainder section
} while ( TRUE);
```

Swap instruction Definition:

```
void Swap (boolean *a, boolean *b)
{
boolean temp = *a;
*a = *b;
*b = temp:
}
```

Solution using Swap

Shared Boolean variable lock initialized to FALSE; Each process has a local Boolean variable key.

Solution:

```
do {
key = TRUE;
while ( key == TRUE)
Swap (&lock, &key );
// critical section
lock = FALSE;
// remainder section
} while ( TRUE);
```

11 What are semaphores? Illustrate the semaphore implementation with no busy waiting.

The various hardware-based solutions to the critical-section problem using the TestAndSetC) and SwapO instructions are complicated for application programmers to use. To overcome this difficulty, we can use a synchronization tool called a semaphore.

With each semaphore there is an associated waiting queue. Each entry in a waiting queue has two data items:

- value (of type integer)
- pointer to next record in the list

Two operations:

- Block – place the process invoking the operation on the appropriate waiting queue.
- Wakeup – remove one of processes in thewaiting queue and place it in the ready queue.

Implementation of wait:

```
wait (S){
value--;
if (value < 0) {
add this process to waiting queue
block(); }
}
```

Implementation of signal:

```
Signal (S){
value++;
if (value <= 0) {
remove a process P from the waiting queue
wakeup(P); }
}
```

12 What are semaphores? Show its usage & implementation in detail.

The various hardware-based solutions to the critical-section problem using the TestAndSetC) and SwapO instructions are complicated for application programmers to use. To overcome this difficulty, we can use a synchronization tool called a semaphore.

Usage
Operating systems often distinguish between counting and binary semaphores. The value of a counting semaphore can range over an unrestricted domain. The value of a binary semaphore can range only between 0 and 1. On some systems, binary semaphores are known as mutex locks, as they are locks that provide mutual exclusion. We can use binary semaphores to deal with the critical-section problem for multiple processes. The n processes share a semaphore, mutex, initialized to 1.
Counting semaphores can be used to control access to a given resource consisting of a finite number of instances. The semaphore is initialized to the number of resources available. Each process that wishes to use a resource performs a waitQ operation on the semaphore (thereby decrementing the count). When a process releases a resource, it performs a signal () operation (incrementing the count). When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.
Semaphore Implementation
- Must guarantee that no two processes can execute wait () and signal () on the same semaphore at the same time
- Thus, implementation becomes the critical section problem where the wait and signal code are placed in the critical section.
- Note that applications may spend lots of time in critical sections and therefore this is not a good solution.

13 What are monitors? Illustrate solution to the Dining Philosopher’s problem using monitors.

The monitor is one of the ways to achieve Process synchronization. The monitor is supported by programming languages to achieve mutual exclusion between processes.

1. high-level abstraction that provides a convenient and effective mechanism for process synchronization
2. Only one process may be active within the monitor at a time

Solution to Dining Philosophers

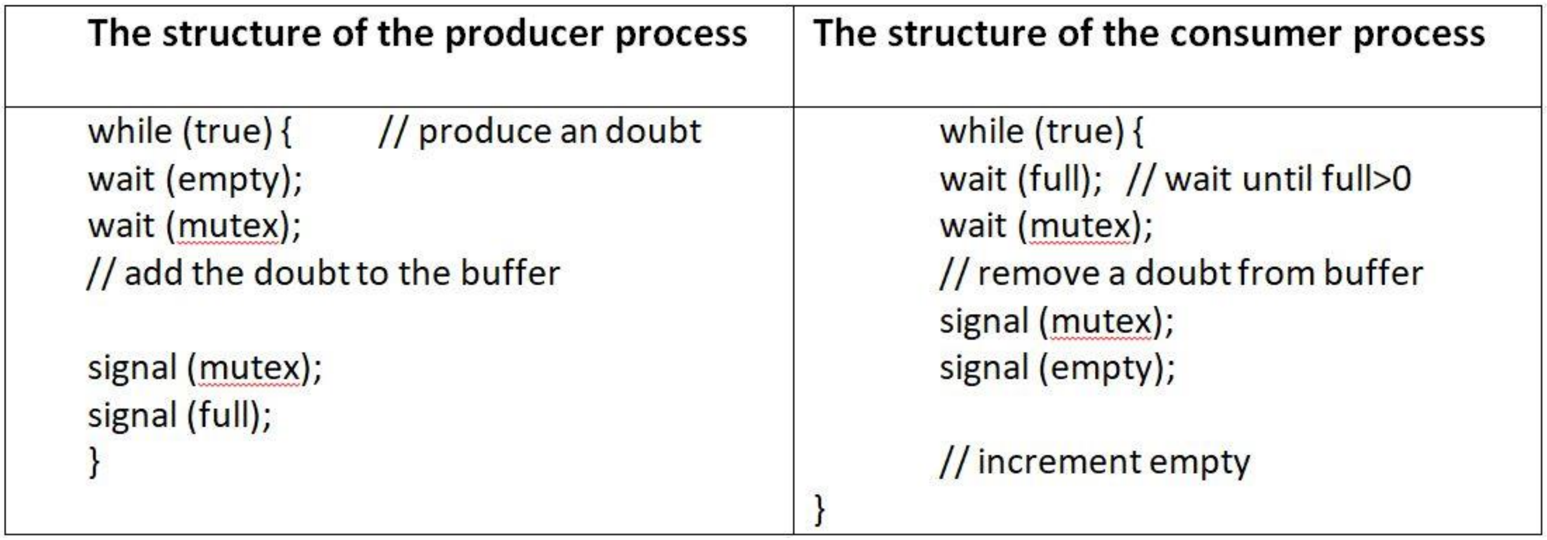
```
monitor DP
{
enum { THINKING; HUNGRY, EATING) state [5] ;
condition self [5];
void pickup (int i) {
state[i] = HUNGRY;
test(i);
if (state[i] != EATING) self [i].wait;
}

void putdown (int i) {
state[i] = THINKING;
// test left and right neighbors
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;
}
}
```

14 Discuss solution to the bounded buffer problem using semaphores.

Assume that there are n buffers, each capable of holding a single item. We use three semaphores: empty and full to count the empty and full buffers and mutex to provide mutual exclusion for operations on the buffer pool.
mutex is initialized to 1, empty is initialized to n and full is initialized to 0.

1. N buffers, each can hold one item
2. Semaphore mutex initialized to the value 1
3. Semaphore full initialized to the value 0
4. Semaphore empty initialized to the value N.



The structure of the producer process	The structure of the consumer process

while (true) {   // produce an doubt
wait (empty);
wait (mutex);
// add the doubt to the buffer
 	
signal (mutex);
signal (full);
}
	while (true) {
	wait (full); // wait until full>0
	wait (mutex);
	// remove a doubt from buffer
	signal (mutex);
	signal (empty);
	
	// increment empty
	}

15 Describe readers-writers problem and illustrate solution for it.

A data set is shared among a number of concurrent processes

Readers – only read the data set; they do not perform any updates
Writers – can both read and write.
Problem – allow multiple readers to read at the same time. Only one single writer can access the shared data at the same time.

Shared Data

- Data set
- Semaphore mutex initialized to 1.
- Semaphore wrt initialized to 1.
- Integer readcount initialized to 0.

The structure of a writer process

```
do {
```

```
wait (wrt) ;
// writing is performed
signal (wrt) ;
} while (true)
```

The structure of a reader process

```
do {
wait (mutex) ;
readcount ++ ;
if (readercount == 1) wait (wrt) ;
signal (mutex)
// reading is performed
wait (mutex) ;
readcount - - ;
if redacount == 0) signal (wrt) ;
signal (mutex) ;
} while (true)
```