Operating Systems CSO-231 Mid-semester Assignment.

Name :- Shruti T. Avhad Roll no:- 20074030 Branch:- CSE(IDD)

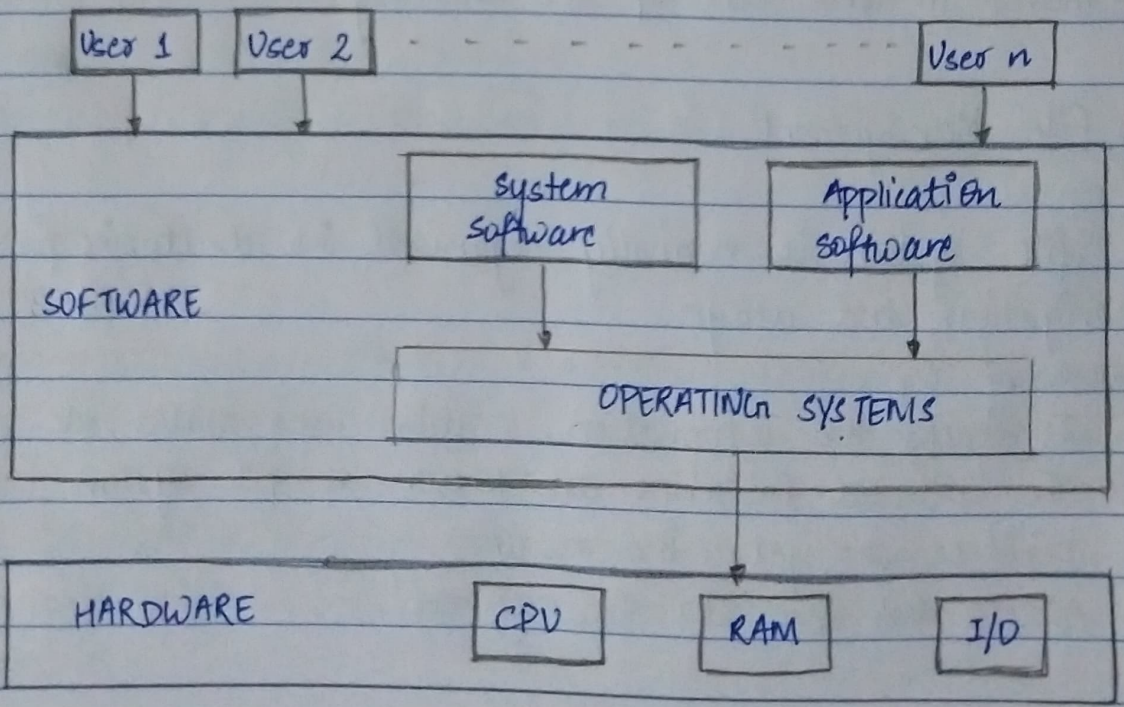
(Q1.)

Ans-

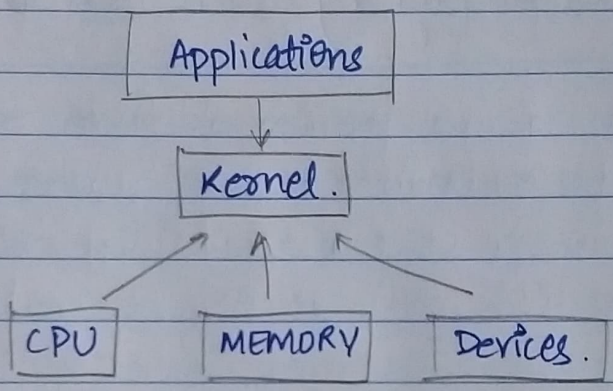
An Operating system (OS) is a system software that is a Resource Allocator. It controls and manages the execution of programs and provides abstraction to users from low level implementations of computer systems providing an interface. It acts as an intermediary between the computer user and computer hardware and also provides a basis for application programs.

When it comes to defining an operating system, there is no universally accepted definition of OS because it performs a varied range of tasks and keeps evolving. As the fundamental goal of a computer is to execute user programs, hence comp. hardware is developed but a hardware is not an easy to use tool, so application programs are developed. And they require controlling, managing I/O devices and allocating resources, so these functions are brought together in one software, The OS.

Ex. of OS :- Windows 11, Ubuntu, Kali linux, Mac OSX, etc.



The core component of an operating system is a Kernel.
It acts as a bridge between the software and hardware.
It always remains in memory and handles interaction between software and hardware.
The kernel allocates requests from applications to perform I/O to an appropriate device and provides convinient methods for using the device.



OS performs the following main functions:

1. Memory Management
2. Processor Management
3. Device Management
4. File Management
5. security
6. control over system performance.
7. Job accounting.
8. Error detecting aids.
9. Coordination between other software and users.

User's view:

for a user, there may be several cases like for a personal computer, performance is must and there is very less necessity to look into hardware resource utilization. Also there may be a case where many users are using the same system through different terminals or accessing the same server from distributed stations.

System's view:

from the system's view OS works as a resource allocator whose major work is to allocate the hardware resources to many application programs.

So, a system views an operating system as a resource allocator and user as a utilisation performation. Both the views have their own independent objects and the operating system serves both of them independently and very efficiently.

Q2.)

Ans- Services of an Operating System

An operating system acts as an interface between the user and hardware. It provides services to both users and to the application and system programs.

a) Memory Management

OS manages the available memory using various scheduling algorithms to ensure proper allocation of resources to processes in an efficient manner to increase throughput and prevent starvation as well as utilize CPU efficiently.

- It keeps track of memory currently being used and the process which is using them.
- Allocates and Deallocates memory space as needed.
- Decision making of which processes and data is to be moved in and out of the memory.

b) File Management

A file system is normally organised into directories for easy navigation and usage.

Work of OS:-

- It tracks the information, location, user status, etc. The collected facilities are known as file system.
- Decides who gets the resources.
- Allocates and deallocates these resources.

c) Processor Management

Multiprogramming OS uses the function of process scheduling that is when a process gets the processor and for how much time. Following are the major activities of OS with respect to processor management:

- OS keeps track of primary memory that is what part of it is in use by whom.
- Allocates the memory when a process requests it and de-allo-cates when a process no longer uses it.

d) Input/Output System Management

One of the purposes of an operating system is to securely hide the peculiarities of specific hardware devices from the users by use of I/O subsystem. The I/O subsystem consists of several components.

- A memory-management component that includes buffering, caching and spooling.
- A general device-driver interface.
- Drivers for specific hardware devices.

e) Communication

OS manages communication b/w all the processes. OS handles all routing and connection strategies, and problems of contention and security.

- It provides a medium of communication between processes.
- Implements communication by shared memory or message passing.

f) Error Handling

↳ OS is responsible for constantly checking for possible errors in CPU, I/O devices or anywhere else.

↳ Taking appropriate action to ensure correct and consistent computing.

g) Protection

Multi-users and multiprogramming system access to data is regulated by the OS by implementing timers, memory addressing hardware, device control registers, dual operating mode (kernel and user) etc.

↳ It controls the access of processes of users to the resources defined by a computer system,

↳ External I/O devices and system are safe from unauthorized access attempts,

↳ Provides Authentication facility.

h) Program Execution

OS handles many kinds of activities from user programs to system programs like printer spooler, name server, etc. Each of these activities are encapsulated as a process. A process includes complete execution context (execution code, registers, data, resources).

↳ OS Loads program in memory and create processes.

↳ Scheduling processes and threads on the CPUs

↳ Allocate resources to process

↳ suspending and resuming processes

↳ Provides mechanisms for process synchronizat", communication and deadlock handling.

(Q3.)

Ans- **Program:**

A program is an executable file which contains a particular set of instructions written to complete the specific job on your computer. Programs are stored on a disk or secondary memory on our PC or laptop.

Process:

A process is an executing entity of a specific program. It is considered an active entity that actions the purpose of application. Multiple processes may be related to the same program. It contains the program code and its activity. Multiple processes may be related to the same program.

Difference b/w a process and program.

Parameter	Process	Program
Resource Management	The resource requirement is quiet high in case of a process.	A program needs memory only for storage.
Control Block	It has its own control Block called PCB (process control Block.)	A program does not have any control blocks.
Entity type	A process is an active or a dynamic entity.	A program is a passive or static entity.

Inter-relation	Multiple process can be on the same code, hence the same program.	A same process can run on different programs.
Overhead.	Process have considerable overhead.	No significant overhead cost.
Memory	It is loaded into the main memory.	A program resides in secondary memory.

Process Control Block (PCB)

A Process control Block is a Data structure that contains information of the process related to it. While creating a process the OS performs several operations. To identify the process, it assigns a process identification number (PID no.) to each process which is generally generated as mex of current PIDs (Mex = Minimum excludant.). PCB is used to track the process's execution status, it contains information about process state, program counter, stack pointer etc.

Pointer
Process State
Process Number
Process Counter
Registers
Memory Limits
Open File Lists
⋮

1. Pointer: It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
2. Process State: It stores the respective state of the process.
3. Process Number: Every Process is assigned with a unique id known as PID which stores the process identifier.
4. Program Counter: It stores the counter which contains the address of the next instruction that is to be executed for the process.
5. Register: There are the CPU registers which includes: accumulator, base, registers, and general purpose registers.
6. Memory Limits: This field contains the information about the memory management system used by OS. This may include the page tables, segment tables etc.
7. Open File List: This information includes the list of files opened for a process.

(Q4.)

Ans- Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data - that is, send data to and receive data from each other. While independent processes are not affected by execution of other processes, an operating system can be affected by other processes.

There are two fundamental models of interprocess communication:
1. shared memory
2. Message passing.

These both models are common in operating systems, and many systems implement both.

Shared Memory

Interprocess communication using shared memory requires communicating processes to establish a region of shared memory. Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment ~~Other processes that wish to communicate using this shared-memory segment~~ must attach it to their address space.

They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by these processes and are not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously. Eg.:- Producer Consumer scenario

Message Passing

Another way to achieve the same effect is for the operating system to provide the means for cooperating processes to communicate with each other via a message-passing facility. Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. It is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.

IPC is responsible for providing 2 operations:

- send (entity, message) and receive (entity, message) where message size is fixed or variable, the entity varies in case of Direct communication its identifier for the process for Indirect communication its the identifier for mailbox/port.

Direct communication involves processes communication by explicitly naming the processes and sending and receiving operation work where the entity is the process name.

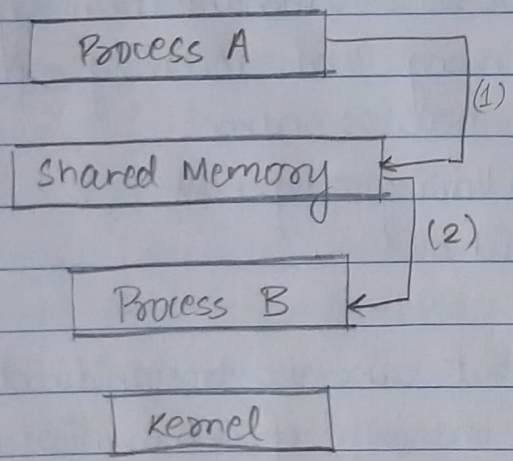
Communication link has the following properties:

↳ Links are automatically established.
↳ Each pair contains one link
↳ link is usually bidirectional but can also be unidirectional.
↳ Link is associated with exactly one pair of communicating processes.
↳ The link will cease to work if process name changes.

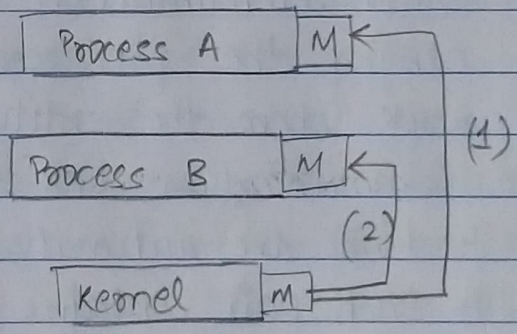
Indirect communication involves messages that are received and directed from mailboxes (ports). Operations involved are not much different from direct communication just involves creation of a mailbox and the entity in send and receive functions is mailbox.
There exists a unique id for each mailbox and communication is possible because of sharing of mailboxes.
It has following properties:

- ↳ Link maybe associated with many processes
- ↳ Link can be uni as well as bi directional.
- ↳ Link is established by processes sharing a common mailbox.
- ↳ Each pair of processes may share several communication link.

Shared Memory



Memory Passing:

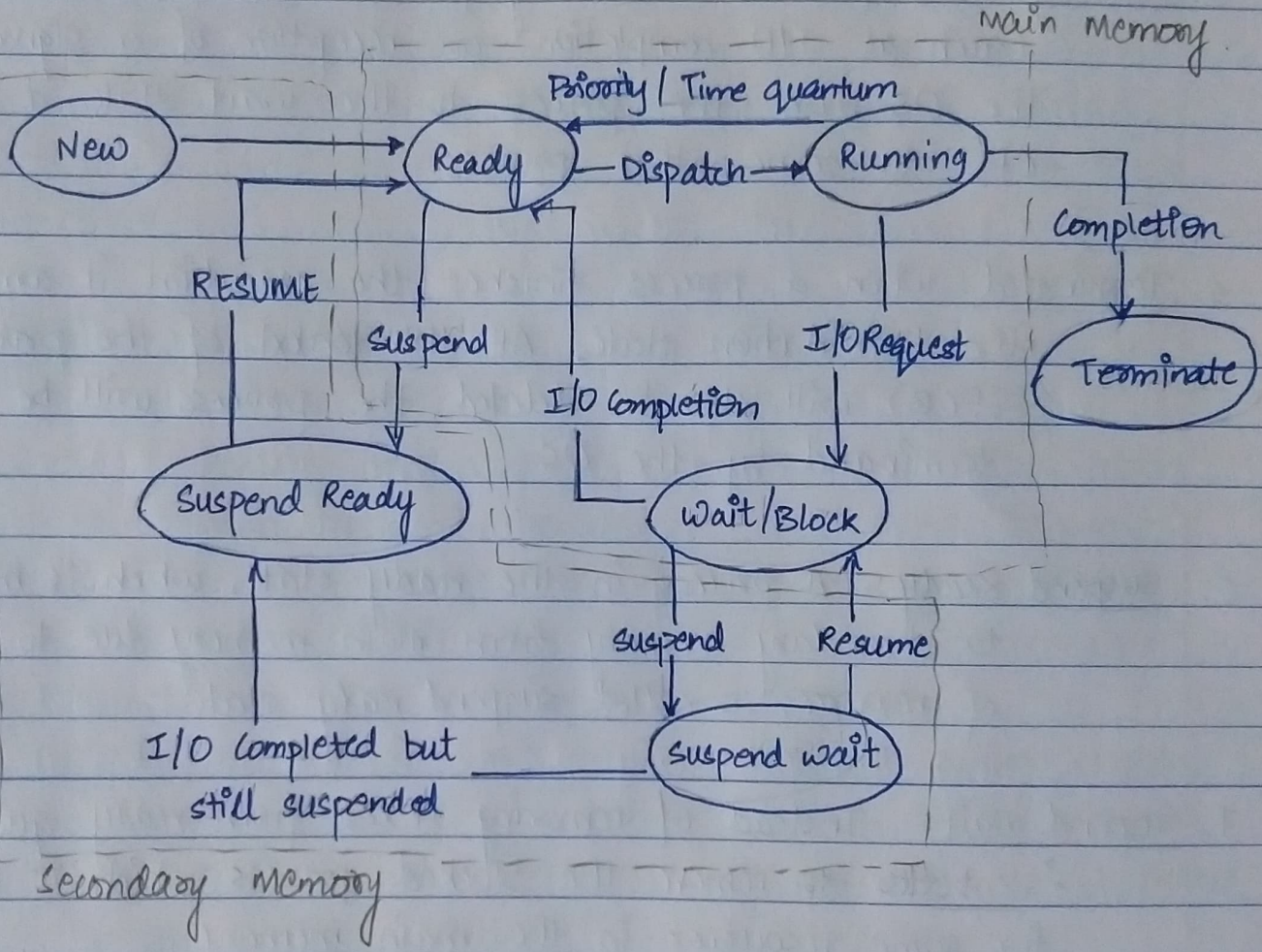


Some Ex. of IPC systems:

1. POSIX: used shared memory method.
2. Mach: Uses message passing.
3. Windos XP: Uses message passing using local procedural calls.

Q5.)

Ans- A process passes through different states as it executes. These states may be different in different operating systems. Following is the process-state diagram:-



1. New: A process which is going to be picked up by the OS into the main memory is called a new process.
2. Ready: Whenever a process is created, it directly enters the ready state where it waits for the CPU to be assigned. It is ready for execution and resides in main memory.

3. Running: Instructions are being executed. And the max. no. of running processes present at particular time is equal to no. of processors in the system.

4. Waiting: The process is waiting for some event to occur (such as I/O completion or reception of a signal). When the OS move the process to the wait state it assigns CPU to some other process.

5. Terminated: When a process finishes the execution, it comes in the termination state, All the context of the process (PCB) will also be deleted, the process will be terminated by the OS.

6. Suspend Ready: A process in the ready state, which is moved to secondary memory from main memory due to lack of resources is called suspend ready state.

7. Suspend wait: Instead of removing process from ready queue, it is better to remove the blocked process which is waiting for some resources in the main memory.
Since, it is already waiting for some resource to get available, hence it is better if it waits in secondary memory and make room for high priority process.
These processes complete their execution once the main memory gets available and their wait is finished.

Q6.)

Ans- Six state Process Model:

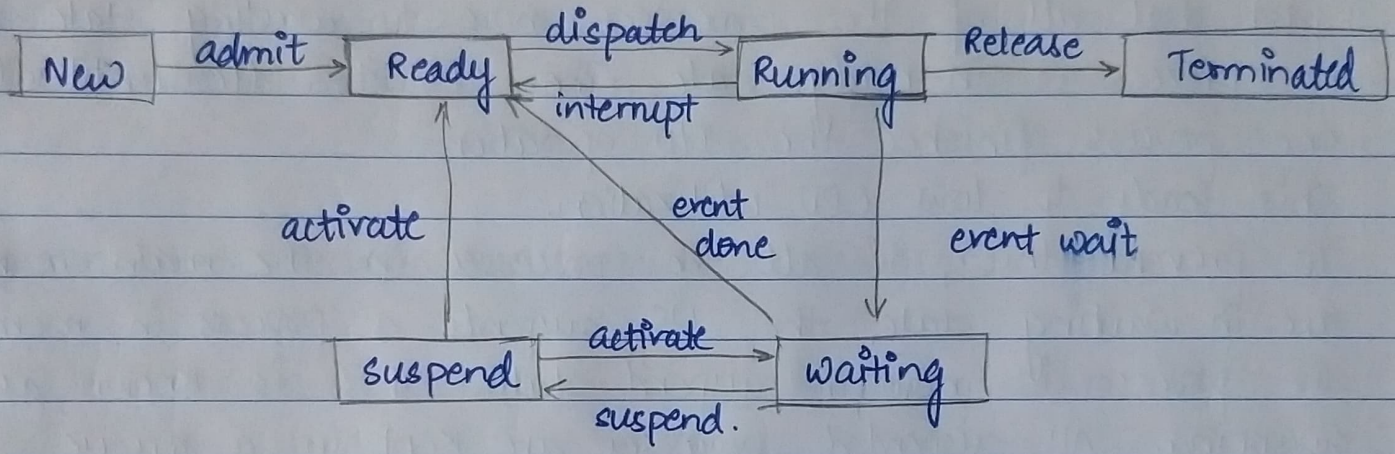
It is commonly a five state process model with suspended state. There is one flaw in the five state model, as we know, the processor is much faster than I/O devices.
∴ a situation may occur where the processor executes so fast that all of the processes more to waiting state and no process is in ready state. The CPU sits idle until atleast one process finishes the I/O operation.
This leads to low CPU utilization.
To prevent this, if all the processes in the main memory are in waiting state, the OS suspends a process in memory. The process is now in suspend state. This is known as swapping. All suspended processes are kept in a queue.
The memory of the swapped process is freed. CPU can now bring some other process in the main memory.
Bringing a process from the suspend queue is prefered.

This scenario is similar to traffic controlling on roads. The vehicles on some of the roads are made stopped while vehicles on the other road can run. Likewise, here the suspend state works.

State Transitions:
Waiting → suspend:
The OS moves a process from the waiting state to a suspend state if all the processes in the main memory are in the waiting state.

Suspend → Ready: when sufficient memory is available, the OS mores a process from the suspend state back to the main memory for execution.

Suspend → Waiting: The process bought by OS from secondary memory to main memory might still be waiting for some event.



Sin state process model Transition Diagram.

Seven State Process Model:

It is commonly known as five state process model with two suspended states.

There is one major drawback of in the previous process state model. That is, the CPU may swap a process that is still waiting for event completion from secondary memory back to the main memory. There is no point in moving a blocked process back to the main memory. The performance suffers. To avoid this, we divide the suspend state in 2 states:

1) Blocked / suspend: The process is in secondary memory but not yet ready for execution.
2) Ready / suspend: The process is in secondary memory and ready for execution.

State Transitions:

Waiting → Blocked / suspend: If all the processes in the main memory are in the waiting state, the processor swaps out atleast one waiting process back to secondary memory to free memory to bring another process.

Blocked / suspend → waiting: This transition might look unreasonable but if the priority of a process in Blocked / suspend state greater than process in ready / suspend state then CPU may prefer process with higher priority.

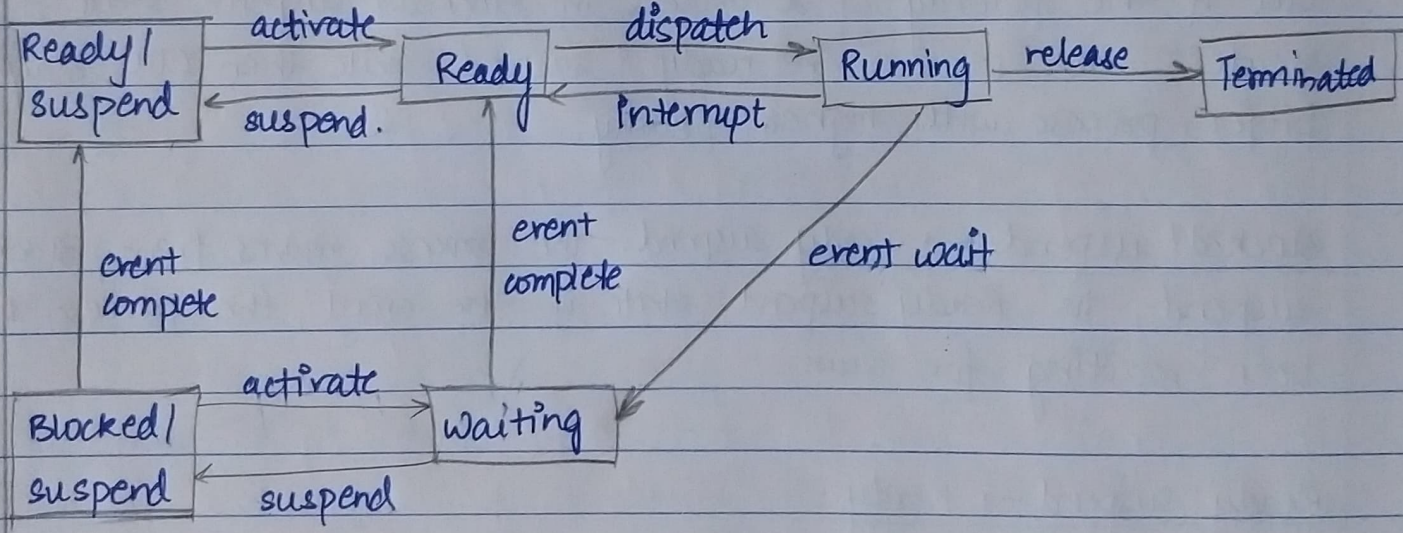
Blocked / suspend → Ready suspend: The process moves from Blocked / suspend to Ready suspend state if the event, the process has been waiting for occurs.

Ready / suspend → Ready:
The OS moves a process from secondary memory to the main memory when there is sufficient space available. Also, if there is high priority process in ready / suspended to face main memory for a higher priority process.

Ready → Ready/suspend:
The OS moves a process from the ready state to ready/suspended to free main memory for a higher priority process.

New → Ready/suspend:
The OS may move a new process to ready/suspended if the main memory is full.

Seven state Process Model Transition Diagram



Q7.)

Ans- A thread is a basic unit of CPU utilization, it comprises of a thread ID, a program counter (PC), a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals. A traditional process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.

Advantages of using Threads

1. Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Additionally, context switching is generally faster between threads than between processes.
2. Hardware level threads that provide hyper threading enables the core to save time during memory stall as the processor works at a faster speed than memory and while it is stalled using other available thread it can do other jobs.
3. Threads allow us to do task parallelism which is different from concurrency where several tasks make progress but at a given time only one task is progressing. While is task parallelism many tasks are progressing at the same time making efficient use to today's multicore systems.
4. Many applications can also take advantage of multiple threads, including basic sorting, trees, and graph algorithms.

5. Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. A single-threaded application would be unresponsive to the user until the operation had been completed.

Issues associated with threading:

1) The fork() and exec() system calls:-

The semantics of the fork() and exec() system calls change in multithreaded program.

Issue: If one thread in a program calls fork(). Does the new process duplicate all threads, or is the new process single-threaded.

solution: Some UNIX systems have chosen to have two versions of fork(). one that duplicates all threads and another that duplicates only threads invoked in fork() sys. call.

2) Thread Cancellation:-

Thread cancellation means terminating thread before completion.
Thread to be cancelled is called target thread.

Cancellation Types:

1. Asynchronous:- Terminates target thread immediately.
2. Deferred:- Allow target thread to periodically check if it should be cancelled.

When target thread has resources it becomes troublesome with asynchronous cancellation.

3.) Signal Handling:
Signals are used in UNIX system to notify a process that a particular event has occured. Now in when a multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all or a single thread.

4.) Scheduler Activations:-

In multithreaded programs there must be communication between kernel and thread library.
Scheduler Activation is a scheme which pooride this communication.
It allow user level threads to act like virtual processors.
Further kernel must inform application about event that procedure is known as upcall, handled by thread library with an upcall handler.

5.) Thread-local storage:

In some circumstances thread need its own specific data (like a unique identifier). Most thread libraries support thread specific data.

Q8.)

Ans-

1.) A sequential code:

In a program where each step being executed relies heavily on the results from previously executed steps, multi-threading just won't work.

for example, if we have a program which loads an image, rotates it, then scales it, then turns it into black and white, then saves it again, each stage really needs the previous one to be finished before it goes further.

2.) A program Bounded by a single resource:

If our application is bound by a single resource - using hard disk drive, and all the tasks we would use that same resource, we'll just be adding contention. For instance, suppose we had an application which collected all the names of files on your disk. splitting this task into multiple threads won't help, it might be troublesome because then it would be asking the file system for lots of different directories, all at same time, so the program won't progress steadily.

Q9.)

Ans:- System calls used by OS to execute the program:

1. execve (" ./a.out")
2. brk (NULL)
3. arch_prctl()
4. mmap()
5. access ("/etc/ld.so.preload")
6. openat ("/etc/ld.so.cache")
7. newfstatat()
8. mmap()
9. close (3)
10. openat ("/lib/x86_64-linux-gnu/libc.so.6")
11. read()
12. pread64() * 3
13. newfstatat()
14. pread64()
15. mmap()
16. mprotect()
17. mmap() * 4
18. close (3)
19. mmap()
20. arch_prctl (ARCH_SET_FS)
21. set_tid_address()
22. set_robust_list()
23. mprotect() * 3
24. prlimit64()
25. munmap()
26. getrandom()

27. brk(NULL) *2
28. openat(AT_FDCWD, "FILEA", O_RDONLY)
29. openat(AT_FDCWD, "FILE", O_WRONLY | O_CREAT | O_TRUNC, 0666)
30. newfstatat(3, "", {st_mode=S_IFREG|0661, st_size=51, ...}, AT_EMPTY_PATH)
31. read(3, "copy this text to another"..., 4098)
32. newfstatat(4, "", {st_mode=S_IFREG|0661, st_size=0, ...}, AT_EMPTY_PATH)
33. read(3, "", 4096)
34. close(3)
35. write(4, "copy this text to another"..., 51)
36. close(4)
37. exit_group(0)