OSY - Notes


V2V EDTECH LLP
Online Coaching at an Affordable Price.
OUR SERVICES:
• Diploma in All Branches, All Subjects
• Degree in All Branches, All Subjects
• BSCIT / CS
• Professional Courses
+91 93260 50669
V2V EdTech LLP
v2vedtech.com
v2vedtech

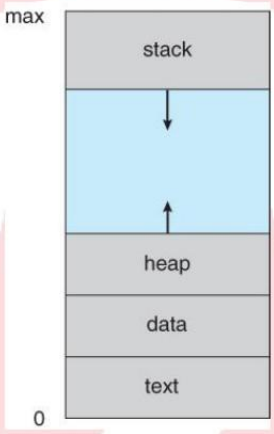
UNIT 2: Process Management

➢ 2.1 Processes

Define : Process, PCB (W – 22)

- **Process :**

Process is a program in execution. Process is also called as job, task and unit of work. The execution of a process must progress in a sequential fashion. A process is an instance of an executing program, including the current values of the program counter, registers and variables. Logically each process has its separate virtual CPU. A process is an activity and it has a program, input, output and a state.



Process in memory

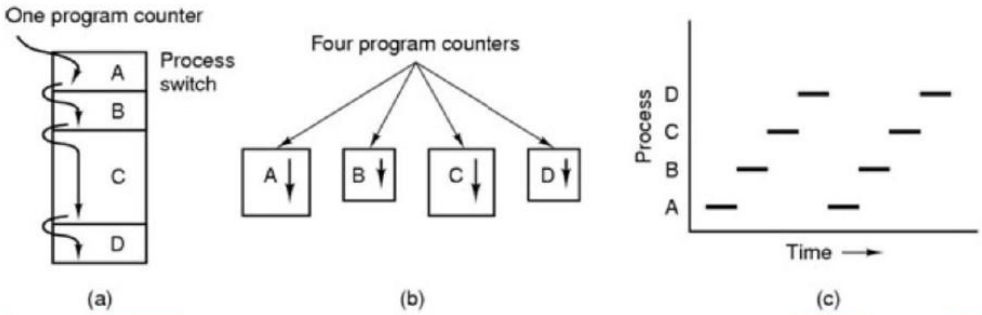
Each process has following sections:

- **Text section :** A Text section contains the program code.
- **Data section :** A Data section contains global and static variables.
- **Heap :** The heap is used for dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- **Stack :** The stack is used for local variables. A process stack which contains the temporary data. Space on the stack is reserved for local variables when they are declared and the space is freed up when the variables go out of scope.
- **Program counter :** Program counter contains the contents of processor's registers.

- **Process Model:**

Explain in detail Process Model

In the process model, an operating system organizes operations into a number of sequential processes. A process is defined as an executing program, which includes its current state (program counter, register values, variables). Conceptually, each process has its own "virtual CPU." However, in reality, there's only one physical CPU. The illusion of multiple CPUs is achieved by rapidly switching the CPU back and forth between processes. This rapid switching is known as multiprogramming.



(a) illustrates four programs (A, B, C, D) and their respective program counters. In a multiprogramming environment, these programs appear to be running in parallel.

(b) depicts how the CPU switches between these four processes, each with its own independent flow of control. This means each program executes independently of the others, even though they share the same CPU.

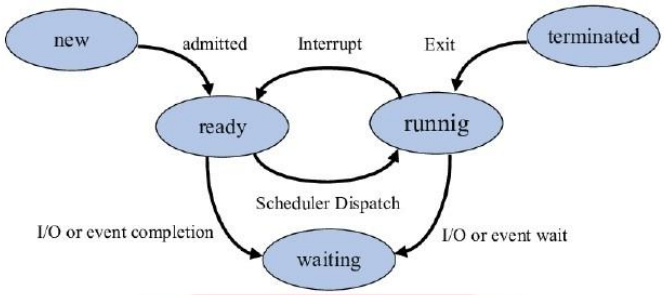
(c) shows the actual execution of processes over time. While it might appear that all processes are running simultaneously, in reality, only one process is executing at any given instant.

The CPU switches back and forth between processes, and the rate of these switches affects the perceived performance. Due to the overhead of CPU switching, the execution of the same processes might not be uniform or perfectly reproducible.

- **Process State :**

Draw and explain process state diagram (W – 19, S – 22, W – 22, S – 23, W – 23, S - 24)

The current activity of a process is known as its state. As a process executes, it changes state.



Process state diagram

A state diagram represents the different states in which a process can be at different times, along with the transitions from one state to another that are possible in the operating system.

Different process states are as follows :

1. **New:**

When a process enters into the system, it is in new state. In this state process is created. In new state the process is in job pool.

2. **Ready:**

When the process is loaded into the main memory, it is ready for execution. In this state the process is waiting for processor allocation.

3. **Running:**

When CPU is available, system selects one process from main memory and executes all the instructions from the process. So, when a process is in execution, it is in running state. In single user system, only one process can be in the running state. In multi-user system, there can be multiple processes which are in the running state.

4. **Waiting State:**

When a process is in execution, it may request for I/O resources. If the resource is not available, process goes into the waiting state. When the resource is available, the process goes back to ready state.

5. **Terminated state:**

When the process completes its execution, it goes into the terminated state. In this state the memory occupied by the process is released.

- **Process Control Block (PCB) :**

Explain PCB with diagram (W – 19, S – 22, W – 22, S – 24, W – 24, S - 25)

Process Control Block (PCB) is a data structure that contains information of the process related to it. It is also known as Task Control Block (TCB). The operating system groups all information that needs about a particular process into a data structure called as PCB. When a process is created, operating system creates a corresponding PCB and released whenever the process terminates. The basic purpose of PCB is to indicate the so far progress of a process.

Process Control Block

- **Pointer:** It is a stack pointer that is required to be saved when the process is switch from one state to another to retain the current position of the process.
- **Process State:** It indicates current state of a process. Process state can be new, ready, running, waiting and terminated.
- **Process Number:** Each process is associated with a unique number which is known process identification number.
- **Program Counter:** It indicates the address of the next instruction to be executed for the process.
- **Registers:** Registers includes accumulators, index registers, stack pointers and general purpose registers plus any condition code information.
- **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page table, segment tables. etc,
- **List of open files:** This information includes the list of files opened for a process.

Following are the functions of PCB :

- **Context switching:**
  Allows the operating system to save the current state of a process and load the state of another process to enable multitasking.

- **Process scheduling:**
  provides the necessary information for the scheduler to determine which process to run next.

- **Memory management:**
  Facilitates the allocation and deallocation of memory to processes and track their memory usage.

- **Resource Management:**
  Tracks and manages the resources allocated to a process, such as file descriptors and I/O devices.

- **Process synchronization and communication:**
  Help manage process synchronization, including handling inter-process communication and synchronization primitives.

- **Process Control:**
  Assists in starting, pausing, resuming, and terminating processes.

- **Security and Access Control:**
  Ensures processes have the necessary permissions to process certain resources and data.

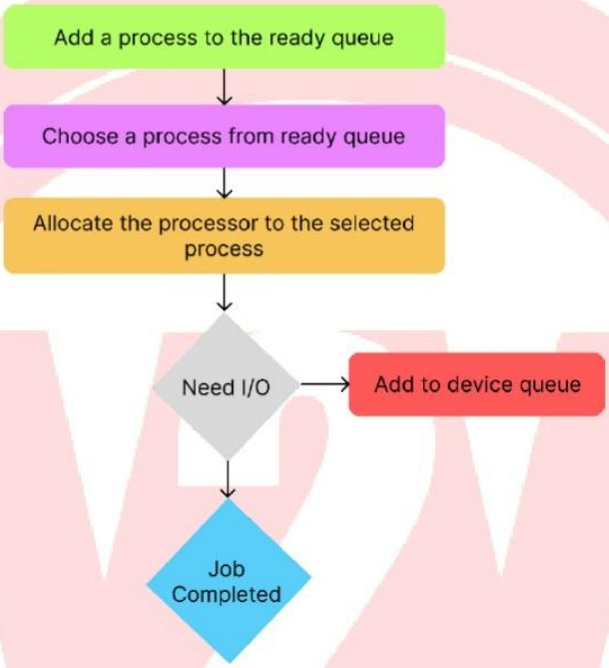
Difference between Program and Process : (W - 23)

Program	Process
It is a set of instructions that has been designed to complete a certain task.	It is an instance of a program that is currently being executed.
It is a passive entity.	It is an active entity.
It resides in the secondary memory of the system.	It is created when a program is in execution and is being loaded into the main memory.
It exists in a single place and continues to exist until it has been explicitly deleted.	It exist for a limited amount of time and it gets terminated once the task has been completed.
It is considered as a static entity.	It is considered as a dynamic entity.
It doesn't have a resource requirement.	It has a high resource requirement.
It requires memory space to store instructions.	It requires resources such as CPU, memory address, I/O during its working.
It doesn't have a control block.	It has its own control block, which is known as Process Control Block.

➢ 2.2 Process Scheduling

Explain the working of Process Scheduling

Procedure of determining the next process to be executed on the CPU is called process scheduling and the module of operating system that makes this decision is called the scheduler. Scheduling is that in which each process have some amount of time of CPU.



Process Scheduling

Working of process scheduling:

1. **Add to Ready queue:**
   New or ready-to-resume processes are added to the ready queue, which holds all processes waiting for CPU time.

2. **Select Process:**
   The short-term scheduler selects a process from the ready queue based on a scheduling algorithm.

3. **Allocate CPU:**
   The selected process is given access to the CPU for execution.

4. **Check for I/O Requirements:**
   If the process needs to perform an I/O operation, it is moved to the device queue and waits until the I/O is complete.

5. **Job Completion check:**

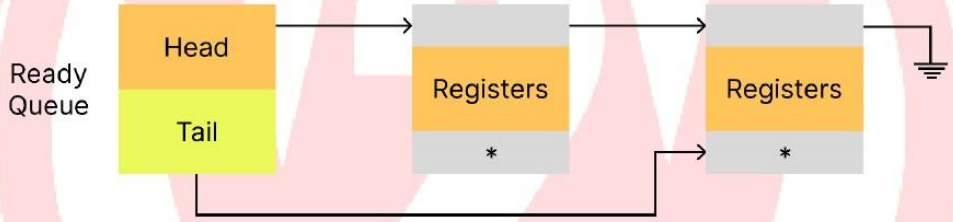
If no I/O is needed, the system checks whether the process has finished its execution.

6. **Process completion or Requeuing:**
   If completed, the process is terminated. If not completed and requires more CPU time, it is placed back in the ready queue.

- **Scheduling queues :**

With suitable diagram, describe use of scheduling queues in process scheduling. (S - 22)

For a uniprocessor system, there will never be more than one running process. If there are more than one process, the rest will have to wait until the CPU is free and can be rescheduled. The processes, which are ready and waiting to execute, are kept on a list called the ready queue. The list is generally a linked list. A ready queue header will contain pointers to the first and last PCBs in the list. Each PCB has a pointer field which points to the next process in the ready queue. There are also other queues in the system. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted or waits for the occurrence of a particular event, such as the completion of an I/O request.



Scheduling Queue

Example :

As 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 assigned the CPU and is executing, one of several events could occur :

1. The process could create a new sub-process and wait for the termination of the sub-process.
2. The process could issue an I/O request and then be placed in an I/O queue.
3. The process could be removed forcibly from the CPU, as a result of an interrupt, and again put in the ready queue.

In the first two cases, the process transition from the waiting state to ready state occurs. A process continues this cycle until it terminates. When the process terminates, it is removed from all queues and its PCB and resources are de-allocated.

- **Scheduler :**

State and describe types of scheduler (W – 19, S - 23)

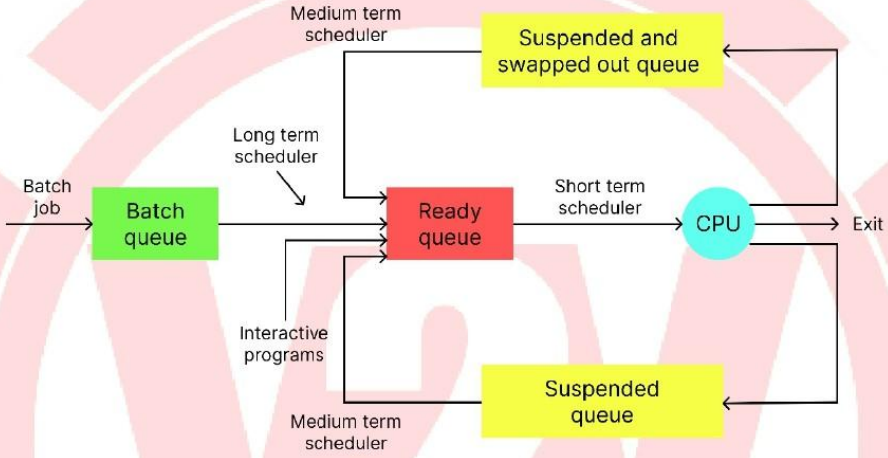
Schedulers are special system software's which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

Schedulers are of three types

a. Long Term scheduler

b. Short Term scheduler

c. Medium Term scheduler



Types of Schedulers:

1. **Long term scheduler:**
   Long term scheduler is also called Job scheduler. It determines which process are admitted to the system for processing. Processes are selected from the queue and loads into the main memory for execution.

2. **Medium term scheduler:**
   Medium term scheduler is part of swapping function. Sometimes it removes the process from memory. It also reduces the degree of multiprogramming.

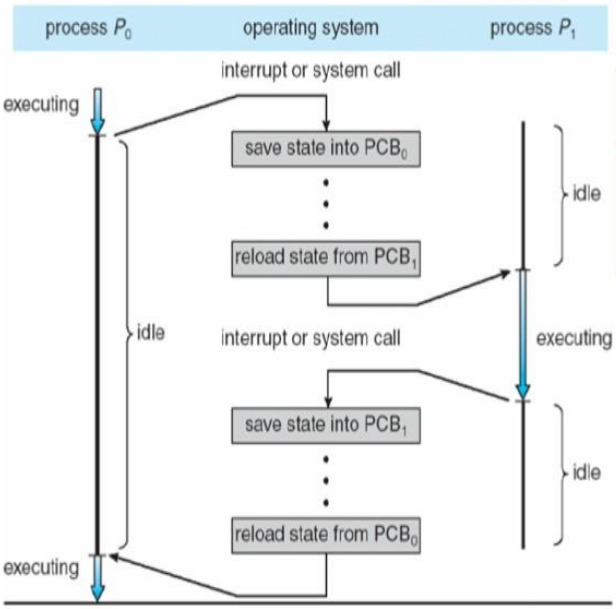
3. **Short term scheduler:**
   Short term scheduler is also called CPU scheduler. It selects the process from queue which are ready to execute and allocate the CPU for execution. Short term scheduler is faster than long term scheduler.

Difference between Long Term Scheduler and Short Term Scheduler (W - 22)

Long Term Scheduler	Short Term Scheduler
Job scheduler	CPU scheduler
Selects processes from job pool and loads them into memory for execution.	Selects processes from ready queue which are ready to execute and allocates CPU to one of them.
Job pool and ready queue	Ready queue and CPU
Executes much less frequently. It executes when memory has space to accommodate new processes.	Executes frequently. It executes when CPU is available for allocation.
Speed is less than short-term scheduler.	Speed is fast.
It controls the degree of multiprogramming.	It provides lesser control over degree of multiprogramming.
It chooses a good process that is a mix-up of input/output bound and CPU bound.	It chooses a new process for a processor quite frequently.
Used in batch processing systems.	Used in time-sharing systems.

- **Context switch :**

Explain working of CPU switch from process to process with neat labelled diagram(W - 23)
OR
Describe how context switch is executed by operating system (S - 24)



A context switching is a mechanism that store and restore the state or context of a CPU in process control block so that a process execution can be resumed from the same point at a late time. When the scheduler switches the CPU from one process to another process, the context switch save the contents of all process registers for the process being removed from the CPU in its process control block. Context switch includes two operations such as state save and state restore. State save operation stores the current information of running process into its PCB. State restore operation restores the information of process to be executed from its PCB.

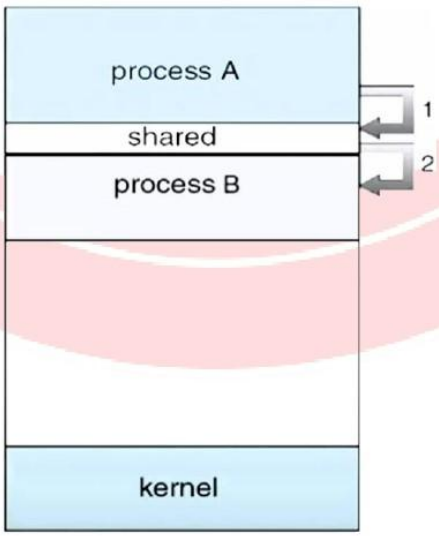
➢ 2.3 Inter Process Communication :

With neat diagram explain inter process communication model. (W – 19, S – 22, W – 22, S – 23, W – 23, S – 24, S - 25)

Cooperating processes require an inter-process communication (IPC) mechanism that will allow them to exchange data and information.

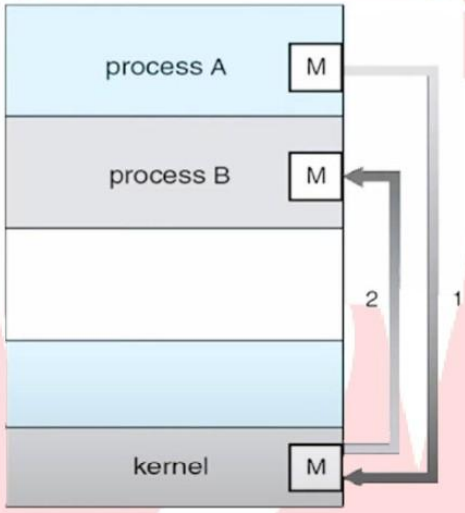
There are two models of IPC.
(a) Shared memory
(b) Message passing

(a) Shared memory :



IPC using shared memory requires a region of shared memory among the communicating process. 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 must attach it to their address space. Normally, the operating system does not allow one process to access the memory region of another process. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. Shared memory allow maximum speed and convenience of communication. Shared memory is faster than message passing.

(b) Message passing:



In this model, communication takes place by exchanging messages between co-operating processes. It allows processes to communicate and synchronize their action without sharing the same address space. It is particularly useful in a distributed environment when communication process may reside on a different computer connected by a network. Communication requires sending and receiving messages through the kernel. The processes that want to communicate with each other must have a communication link between them.

Comparison between Shared Memory Model and Message Passing Model:

Shared Memory Model	Message Passing Model
The shared memory region is used for communication.	A message-passing facility is used for communication.
It is used for communication between processes on a single processor or multiprocessor system.	It is typically used in a distributed environment.

The code for reading and writing the data from the shared memory should be written explicitly by the application programmer.	No such code is required here as the message-passing facility provides a mechanism for communication and synchronization of actions performed by the communicating processes.
Here the processes need to ensure that they are not writing to the same location simultaneously.	It is useful for sharing small amounts of data as conflicts need not be resolved.
Faster communication strategy.	Relatively slower communication strategy.
No kernel intervention.	It involves kernel intervention.
It can be used in exchanging larger amounts of data.	It can be used in exchanging small amounts of data.
Data from a client process may need to be transferred to a server process for modification before being returned to the client.	Web browsers, Web servers.

➢ 2.4 Threads:

- **Benefits of Thread :**

Describe benefits of Thread .

1. **Responsiveness:**
   Multithreading allows an application to remain responsive to the user even when parts of it are blocked or performing long operations. For example, a multithreaded web browser can still allow user interaction in one thread while an image is loading in another. This increases the overall responsiveness experienced by the user.

2. **Resource Sharing:**
   Threads, by default, share the memory and resources of the process they belong to. This shared memory space allows for efficient code sharing, enabling an application to have multiple threads of activity all operating within the same address space.

3. **Economy:**
   Allocating memory and resources for process creation is costly. Multithreading offers an economical alternative because threads share the resources of the process to which they belong. It is more efficient to create and manage threads than processes. For instance, in Solaris 2, creating a process is about 30 times slower than creating a thread, and context switching between threads is approximately five times faster.

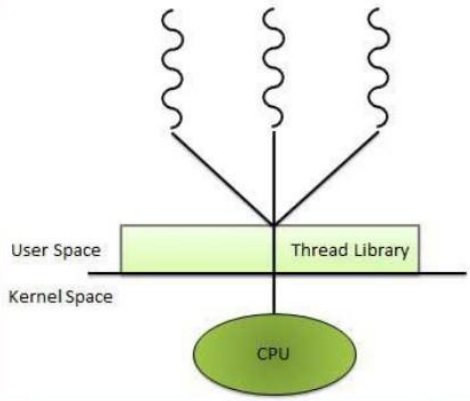
4. **Utilization of Multiprocessor Architectures:**
   Multithreading greatly enhances the benefits of multiprocessor architectures. In such systems, each thread can run in parallel on a different processor, leading to increased concurrency. While a single-threaded process can only utilize one CPU regardless of how many are available, multithreading on a multi-CPU machine can achieve true parallelism. Even in a single-processor architecture, the CPU can rapidly switch between threads, creating the illusion of parallelism.

- User and Kernel level threads

Explain user level thread and Kernel level thread with its advantages and disadvantages (S - 24)

User Level Thread :

The threads implemented at the user level are known as user level threads. In a user thread, all of the work of thread management is done by the application and the kernel is not aware of the existence of threads.



User Level Thread

The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. User level threads are generally fast to create and manage.

Advantages of User Level Threads:

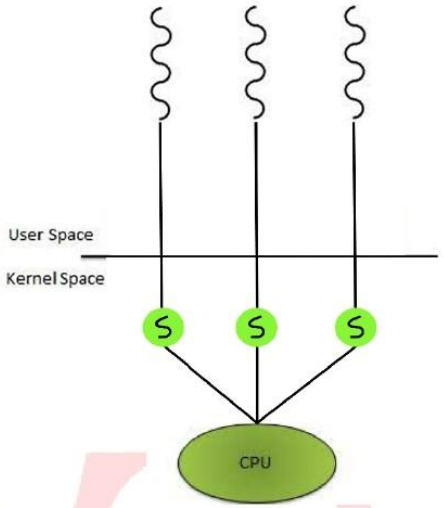
- User level thread can run on any operating system.
- A user thread does not require modification to opera operating systems.
- User level threads are fast to create and manage.
- User thread library easy to portable.

Disadvantages User Level Threads :

- At most, one user level thread can be in operation at one time, which limits the degree of parallelism.
- If a user thread is blocked in the kernel, the entire process (all threads of that process) are blocked.
- It is not appropriat for a multiprocessor system.

kernel level Threads :

The threads implemented at kernel level are known as kernel threads. Kernel threads are supported directly by the operating system.



Kernel Level Thread

The kernel performs thread creation, scheduling and management in kernel space. Because thread management is done by the operating system, kernel threads are generally slower to create and manage than are user threads. However, since the kernel is managing the threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution. Also in a multiprocessor environment, the kernel can schedule threads on different processors.

Advantages of Kernel Level Threads :

- kernel can Simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the kernel can schedule another thread of the same process.

Disadvantages of kernel Level Threads :

- The kernel -level threads are slow and inefficient.
- Transfer of control from one thread to another within same process requires a mode switch to the kernel.

Comparison between User Level Threads and Kernel Level Threads :

User Level Thread	Kernel Level Thread

User level threads are implemented by user.	Kernel level threads are implemented by Operating System (OS).
User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
User level threads can run on any operating system.	Kernel level threads are specific to the operating system.
OS doesn't recognize user level threads.	Kernel level threads are recognized by OS.
Context switch time is less.	Context switch time is more.
No hardware support is required for context switching.	Hardware support is needed.
If one user-level thread performs a blocking operation then the entire process will be blocked.	If one kernel thread performs a blocking operation then another thread can continue execution.
User-level threads can be created and managed more quickly.	Kernel-level threads take more time to create and manage.
In user-level threads, each thread has its own stack, but they share the same address space.	In kernel-level threads have their own stacks and their own separate address spaces, so they are better isolated from each other.
Example: Java Thread, POSIX thread, Mach C-Threads.	Example: Window Solaris.

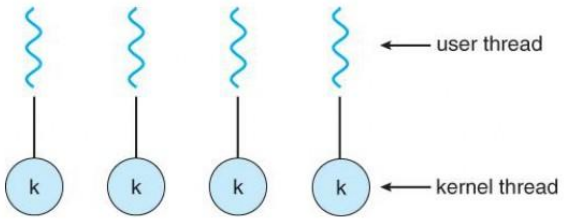
- **Multithreading Models :**

Explain multithreading model in detail. (W – 19, S - 22)

Many system provide support for both user and kernel threads, resulting in different multithreading models.

Following are three multithreading models:

1. One – to - One Model :



The one-to-one model maps each user thread to a kernel thread. It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call; it also allows multiple threads to run in parallel on multiprocessors. Windows NT, Windows 2000 and 08/12 implement the one-to-one model.

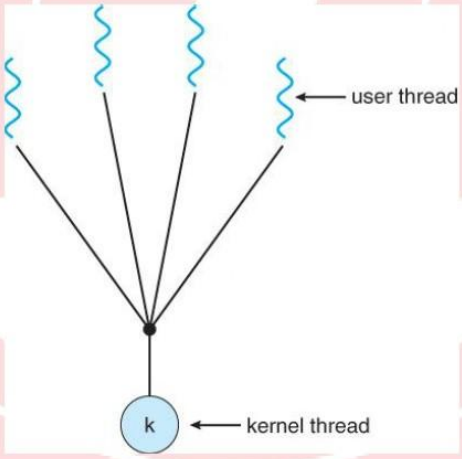
Advantages of one-to-one Model:

- More concurrency, because of multiple threads can run in parallel on multiple CPUs.
- Less complication in the processing.

Disadvantages of one-to-one Model:

- Every time with user's thread, kernel thread is created.
- Limiting the number of total threads.
- Kernel thread is an overhead.
- It reduces the performance of system.

2. Many – to – One Model :



The many-to-one model maps many user level threads to one kernel thread. Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call. Only one method thread can access the kernel at a time. Multiple threads are unable to run in parallel on multiprocessors.

Advantages of many-to-one Model:

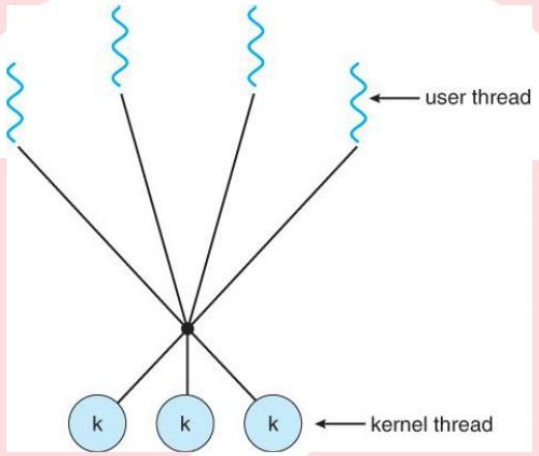
- Totally portable.

- Efficient system in terms of performance.
- One kernel thread controls multiple user threads.

Disadvantages of many-to-one Model:

- Cannot take advantage of parallelism.
- One block call blocks all user threads.

3. Many – to – Many Model :



The many-to-many model maps multiple user level threads to a smaller or equal number of kernel threads. The number of kernel threads may be specific to either a particular application or a particular machine. This model allows developer to create as many threads. Concurrency is not gained because the kernel can schedule only one thread at a time. Solaris 2, IRIX, HP-UX and Tru64 UNIX support this model.

Advantages of Many-to-Many model:

- Many threads can be created as per user's requirement.
- Multiple kernel or equal to user threads can be created.

Disadvantages of Many-to-Many Model:

- True concurrency cannot be achieved.
- Multiple threads of kernel is an overhead for operating system.
- Performance is less.

Differences between Process and Thread: (W - 23)

Process	Thread
A process is a program under execution i.e. an active program.	A thread is a subset of the process.
Process runs in separate memory space.	Thread runs within the process in a shared memory space.
Process is heavy weight	It is a lightweight
The process has its own Process Control Block, Stack, and Address space.	Thread has Parents PCB, its own Thread Control Block, and Stack and Common Address Space.
Context switching between the process is more expensive.	Context switching between threads of the same process is less expensive.
Processes are independent.	Threads are dependent.
Process is controlled by the operating system.	Threads are controlled by a programmer

➢ 2.5 Execute process commands like :

Write syntax for following commands: i)Sleep ii)Kill (W – 19, W - 22)

Describe use of ps and wait commands with suitable example. (S - 22)

Explain following commands with their syntax i) kill ii) Sleep iii) Wait iv) Exit (W - 23)

- **top**

```
top - 14:39:18 up 36 min,  1 user,  load average: 0.08, 0.12, 0.22
Tasks: 266 total,   1 running, 265 sleeping,   0 stopped,   0 zombie
%Cpu0  :  2.0 us,  1.7 sy,  0.0 ni, 96.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  2.7 us,  1.0 sy,  0.0 ni, 96.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu2  :  1.7 us,  1.3 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
GiB Mem :      3.8 total,      0.9 free,      1.1 used,      1.9 buff/cache
GiB Swap:      2.0 total,      2.0 free,      0.0 used.      2.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    932 root      20   0 1068.9m  40.9m  18.8m S   0.0   1.0   0:05.49 snapd
   2815 bosko     20   0 2426.1m 228.0m  69.8m S   0.0   5.8   0:05.39 dropbox
   2784 root      20   0 2028.5m 157.1m  35.0m S   0.0   4.0   0:03.08 mysqld
   2756 bosko     20   0  957.5m  76.5m  47.7m S   0.0   1.9   0:02.80 snap-store
   6555 bosko     20   0  798.2m  49.7m  37.4m S   0.0   1.3   0:02.42 gnome-termin+
   1813 bosko     20   0  154.5m   2.6m   2.3m S   0.3   0.1   0:02.17 VBoxClient
   1982 root      20   0   58.3m   4.5m   3.5m S   0.0   0.1   0:01.65 redis-server
  63283 bosko     20   0   14.5m   3.9m   3.1m R   0.0   0.1   0:01.47 top
```

The top stands for table of processes. As the name suggests, it displays a list of processes in table form. The top command is used to show the Linux processes. It provides a dynamic real-time view of the running

system. Usually, this command shows the summary information of the system and the list of processes or threads which are currently managed by the Linux Kernel.

Syntax: top

- **ps**

Explain 'PS' command with any four options. (W - 19)

What is the use of PS command? Write long forms of UID, PID in the output of this command. (W - 23)

It is used to display character slices of a process. This command when executed without options, it lists the process associated with a user at a particular terminal.

Syntax: $ ps [options]

Example: $ ps

Output:

PID	TTY	TIME	CMD
12330	pts/0	00:00:00	bash
21621	pts/0	00:00:00	ps

Each line in the output shows PID, the terminal with which the process is associated, the cumulative processor time that has been consumed since the process has been started and the process name.

UID - (Owner) User-id
PPID - Parent Process-id

Options :

1. **-f:**

It is used to display full listing of attributes of a process. It includes UID (User ID),PID (Process ID), PPID (Parent ID), C (Amount of CPU time consumed by the process) and STIME (chronological time that has

elapsed since the process started). TTY (The terminal device from which the process was launched), TIME (The cumulative CPU time required to run the process, CMD (The name of the program that was started.

Example: $ ps -f

Output:

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	19:58	?	00:00:01	/sbin/init
root	2	0	0	19:58	?	00:00:00	[kthreadd]
root	3	2	2	19:58	?	00:00:00	

2. -u:

show the activities of any specified user at any time.

Example: $ ps -u abc

Output:

PID	TTY	TIME	CMD
1053	?	00:00:00	systemd
1062	?	00:00:00	(sd-pam)
1074	tty1	00:00:00	zsh

3. -a:

It shows the process of all users.

Example: $ ps -a

Output:

PID	TTY	TIME	CMD
27011	pts/0	00:00:00	man
27016	pts/0	00:00:00	less
27499	pts/1	00:00:00	ps

4. -e:

It displays processes including user and system processes.

Example: $ ps -e

Output:

PID	TTY	TIME	CMD
1	?	00:00:05	systemd
2	?	00:00:00	kthreadd

3	?	00:00:00	csmd
7	?	00:00:01	gjs
8	?	00:00:00	Gsd-sound

Below is a list of all ps commands:

① **-a:** Displays all processes except session headers and processes without a terminal.

② **-c:** Displays additional scheduler information about the process.

③ **-d:** Displays all processes except session header.

④ **-e:** Displays all processes.

⑤ **-f:** Displays a full format listing

⑥ **-j:** Displays job information.

⑦ **-l:** Displays a long listing

⑧ **-w:** Use wide output format, for unlimited width displays.

⑨ **-y:** Don't show process flags.

⑩ **-A:** Displays all processes.

⑪ **-F:** Use extra full output.

⑫ **-H:** Display processes in a hierarchical format (showing parent processes).

⑬ **-L:** Displays process threads.

⑭ **-M:** Display security information about the process.

⑮ **-N:** Display the opposite of the specified parameters.

⑯ **-U userlist:** Displays processes owned by a userid listed in userlist.

⑰ **-V:** Displays the version of ps.

⑱ **-Z:** Display the security context information.

- **kill**

Syntax: kill pid

Kill command is used to stop execution of particular process by sending an interrupt signal to the process.

- **wait:**

Syntax: wait [pid]

wait command waits for running process to complete and return the exit status. The process is identified by pid (process ID). If pid is not given, wait waits for all currently active child processes and the return status is zero.

- **sleep:**

Syntax: sleep NUMBER [SUFFIX]

The sleep command pauses the execution for specified time in command. It pauses execution for amount if time defined by NUMBER, and SUFFIX can be "s" for seconds (default), "h" for hours, "m" minutes, and "d" for days.

- **exit:**

Syntax: exit

used to quit the shell

OR

Syntax: exit [n]

The terminal window will close and return a status of n. It also returns value as which is available to script's parent.

- **nice**

The Linux nice command allows us to launch processes with altered priorities, which affects process scheduling. The "nice command is used to start a new process with a specific priority, known as the "nice value". A higher nice value lowers the process's priority, while a lower (negative) nice value increases it.

Syntax: nice -n [nice value] [command/process]

The -n flag is used to specify the Linux nice value ranging from -20 for most/highest favourable scheduling to +19 for least/lowest favourable scheduling.

Give Commands to perform following task:

i) To add delay in script

ii) To add terminate a process.

i) To add delay in script

To add delay in script sleep command is used.

sleep Command:

The sleep command is used to delay for a specified amount of time. The sleep command pauses for an amount of time defined by NUMBER. SUFFIX may be "s" for seconds (the default), "m" for minutes, "h" for hours, or "d" for days.

Syntax: sleep NUMBER [suffix]

Example: sleep 10

Delay for 10 seconds.

ii) To terminate a process

To terminate a process exit Command is used.

exit Command:

The exit Command terminates a script, just as in a c Program. It can also return a value, which is available to the script's parent process. Issuing the exit Command at the shell prompt will cause the shell to exit.

Syntax: exit

Example: exit

To exit from running State exit command is used

Write the output of following commands

i) wait 2385018

ii) sleep 9

iii) ps -u Asha

i) wait 2385018

Wait command waits until the termination of specified process ID 2385018.

ii) sleep 9

Sleep command is used to delay for 9 seconds during the execution of a process. i.e. it will pause the terminal for 9 seconds.

iii) ps -u Asha.

PS command with -u is used to display data / processes for specific user Asha.

Write Unix command for following:

i)create a folder OSY

ii) create a file FIRST in OSY folder

iii) List/display all files and directories.

iv) Write command to clear the screen

Ans

i) create a folder OSY:

$mkdir OSY

ii)create a file FIRST in OSY folder:

$cd OSY

$cat>FIRST or $ touch FIRST

iii) List/display all files and directories:

$ls

iv) to clear screen:

$clear