JGi JAIN DEEMED-TO-BE UNIVERSITY

SCHOOL OF
COMPUTER
SCIENCE AND IT

Module 2
Process Management

Syllabus

Processes: Process concept, processes, Operations on processes, Inter process communication, Communication in client-server systems.

Threads: Introduction to Threads, Single and Multithreaded processes and its benefits, User and Kernel threads, Multithreading models, Threading issues.

Process Scheduling: Basic concepts, Scheduling criteria, Scheduling Algorithms

What is a Process?

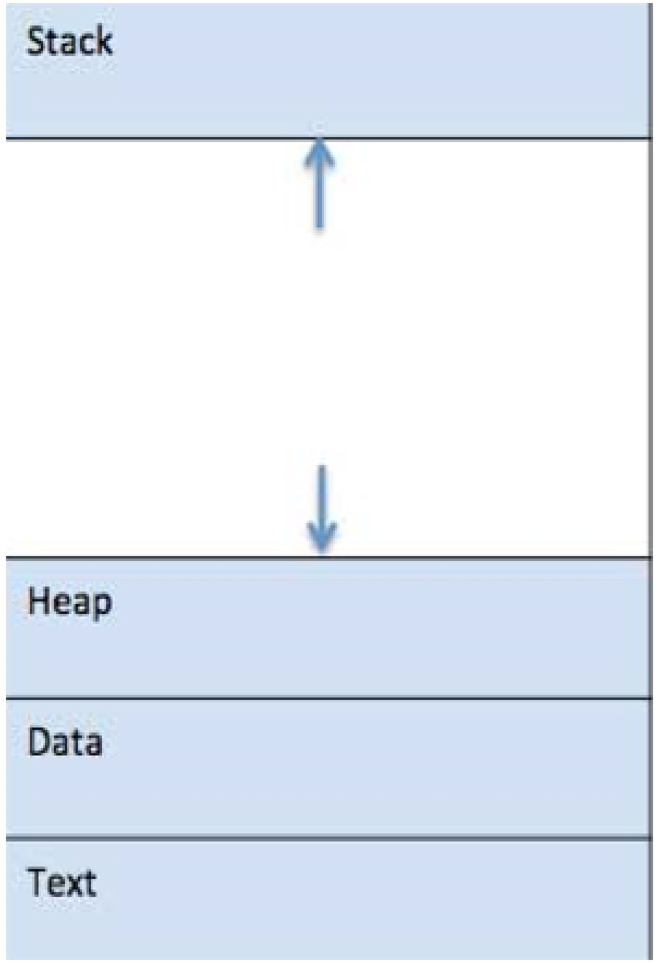
- A process is defined as a system which describes the fundamental unit of work to be carried out in the system.
- A process is a program in execution. It is the unit of work in most systems
- A process will need certain resources—such as CPU time, memory, files, and I/O devices —to accomplish its task.
- These resources are allocated to the process either when it is created or while it is executing.

Process Layout Inside Memory

- A process usually involves the process stack that includes temporary data (such as return addresses, function parameters, and local variables) and a data section which consists of global variables.

- A process may also contain a heap a memory and that is dynamically allocated when the process runs.

- Programs are the passive entity, such as a file consisting of information stored on the disk which is often known as an executable file, whereas a process is an active entity.



Stack

Heap

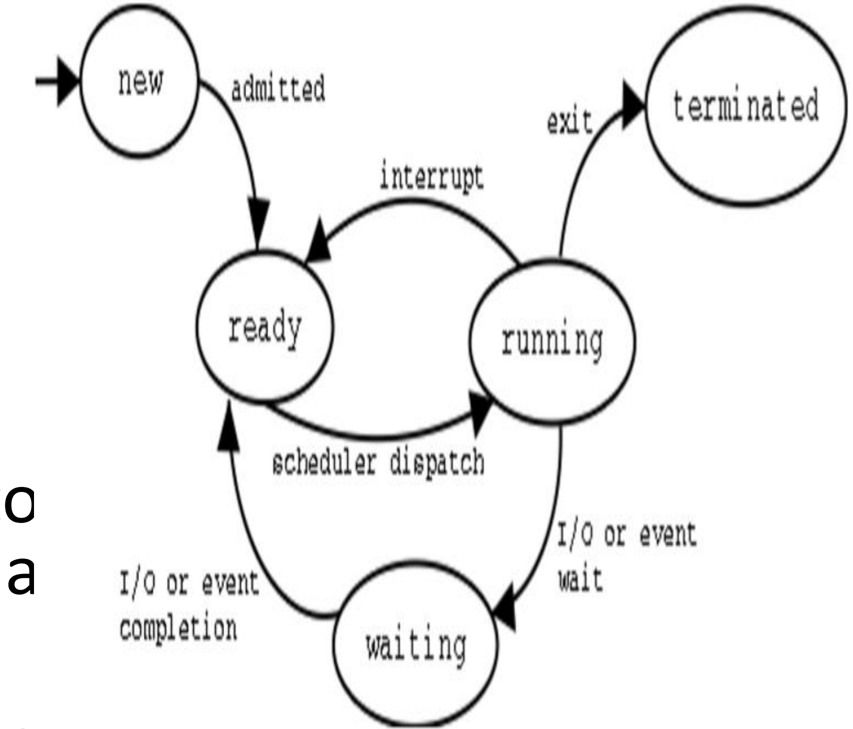
Data

Text

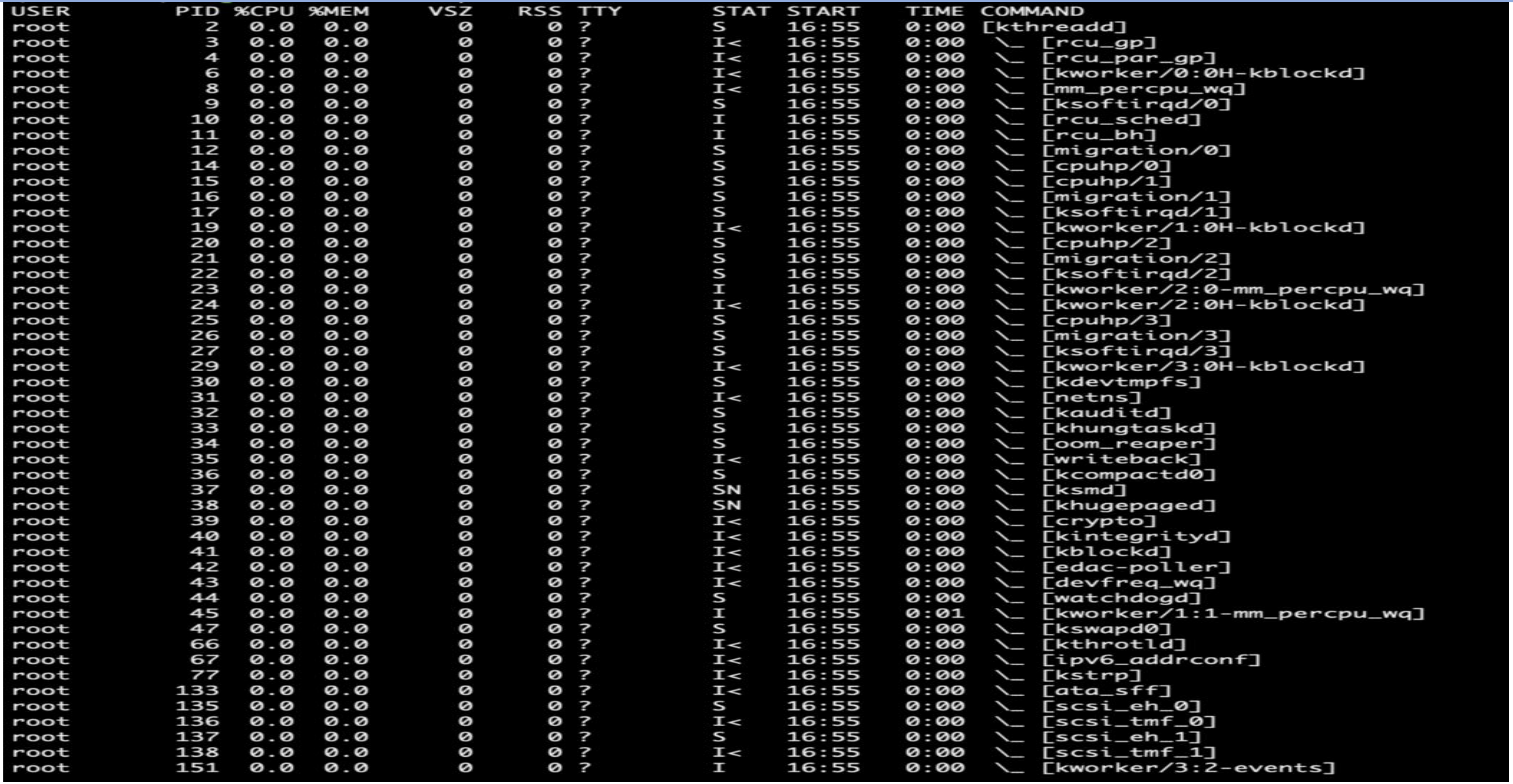
Process States

The state of a process changes, as it executes.

- New. The process is being created.
- Running. Instructions are being executed.
- Waiting. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- Ready. The process is waiting to be assigned to a processor.
- Terminated. The process has finished execution



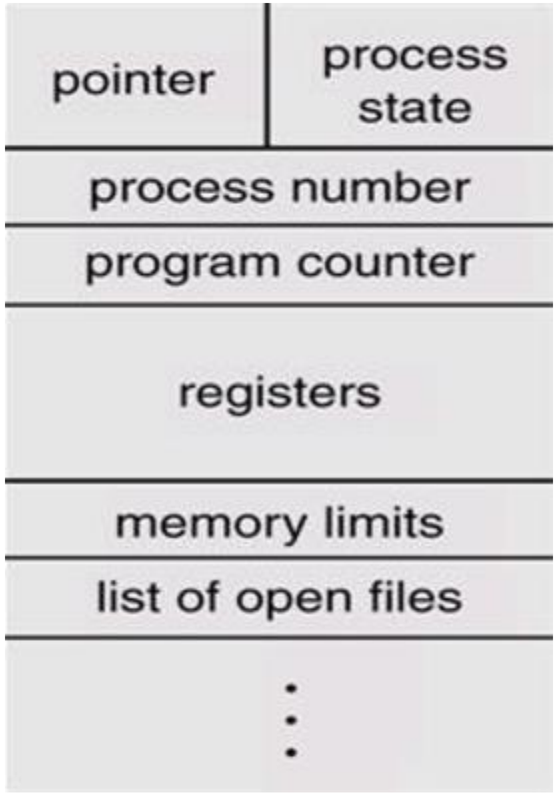
- Process State Codes:
  - D - uninterruptible sleep (usually IO)
  - I - Idle kernel thread
  - R - running or runnable (on run queue)
  - S - interruptible sleep (waiting for an event to complete)
  - T - stopped by job control signal
  - t - stopped by debugger during the tracing
  - X - dead (should never be seen)
  - Z - defunct ("zombie") process, terminated but not reaped by its parent



```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           2  0.0  0.0      0     0 ?        S    16:55   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kworker/0:0H-kblockd]
root           8  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [mm_percpu_wq]
root           9  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [ksoftirqd/0]
root          10  0.0  0.0      0     0 ?        I    16:55   0:00  \_ [rcu_sched]
root          11  0.0  0.0      0     0 ?        I    16:55   0:00  \_ [rcu_bh]
root          12  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [migration/0]
root          14  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [cpuhp/0]
root          15  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [cpuhp/1]
root          16  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [migration/1]
root          17  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [ksoftirqd/1]
root          19  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kworker/1:0H-kblockd]
root          20  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [cpuhp/2]
root          21  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [migration/2]
root          22  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [ksoftirqd/2]
root          23  0.0  0.0      0     0 ?        I    16:55   0:00  \_ [kworker/2:0-mm_percpu_wq]
root          24  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kworker/2:0H-kblockd]
root          25  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [cpuhp/3]
root          26  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [migration/3]
root          27  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [ksoftirqd/3]
root          29  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kworker/3:0H-kblockd]
root          30  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [kdevtmpfs]
root          31  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [netns]
root          32  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [kauditd]
root          33  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [khungtaskd]
root          34  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [oom_reaper]
root          35  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [writeback]
root          36  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [kcompactd0]
root          37  0.0  0.0      0     0 ?        SN   16:55   0:00  \_ [ksmd]
root          38  0.0  0.0      0     0 ?        SN   16:55   0:00  \_ [khugepaged]
root          39  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [crypto]
root          40  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kintegrityd]
root          41  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kblockd]
root          42  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [edac-poller]
root          43  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [devfreq_wq]
root          44  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [watchdogd]
root          45  0.0  0.0      0     0 ?        I    16:55   0:01  \_ [kworker/1:1-mm_percpu_wq]
root          47  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [kswapd0]
root          66  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kthrotld]
root          67  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [ipv6_addrconf]
root          77  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [kstrp]
root         133  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [ata_sff]
root         135  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [scsi_eh_0]
root         136  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [scsi_tmf_0]
root         137  0.0  0.0      0     0 ?        S    16:55   0:00  \_ [scsi_eh_1]
root         138  0.0  0.0      0     0 ?        I<   16:55   0:00  \_ [scsi_tmf_1]
root         151  0.0  0.0      0     0 ?        I    16:55   0:00  \_ [kworker/3:2-events]
```

Process control block

- **Process state**: The state may be new, prepared, executing, waiting, halted and so on.
- **Program counter**: The program counter specifies the address of the next instruction to be carried out for this process.
- **CPU registers:** It consists of accumulators, stack pointers, index registers, and general purpose registers, any condition code information.



pointer | process state

process number

program counter

registers

memory limits

list of open files

•
•
•

Process control block

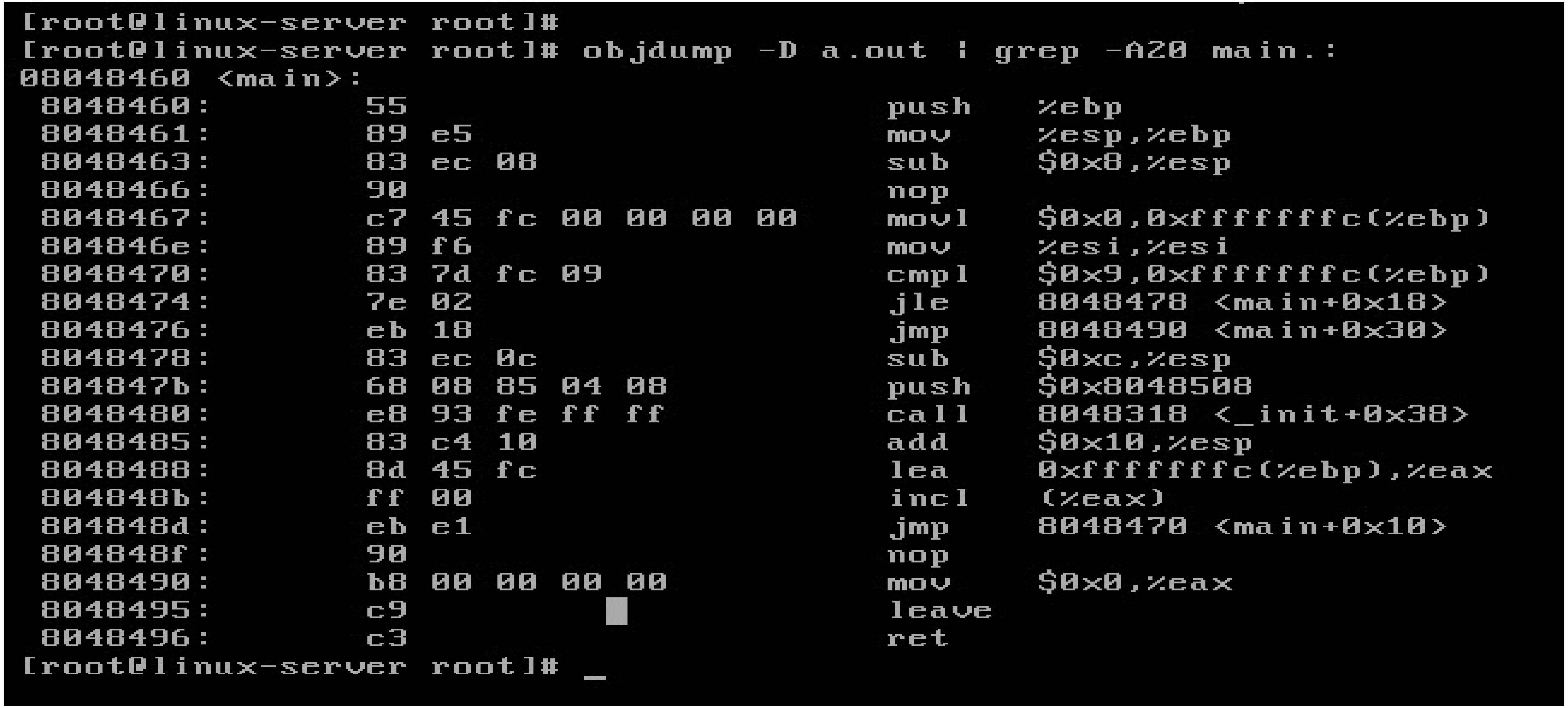
- **CPU scheduling information**: This information involves a pointer to scheduling queues, process priority, and other scheduling parameters.
- **Memory management information**: Information such as the value of the source and limit registers, the segment tables or the page tables depending on the memory system that is applied by the operating system are contained in it.
- **Accounting information**: This information involves the amount of CPU and real time used, account numbers, time limits, job or process numbers, etc.

Process control block

- **I/O status information**: It consists of the list of I/O devices allotted to the process, a list of open files, and so on.

Threads:

A single thread is executed by a program called process. For instance, when a process is running a word-processor program, a single thread of instructions is being carried out. This single thread of control supports the process to execute only one task at one time.

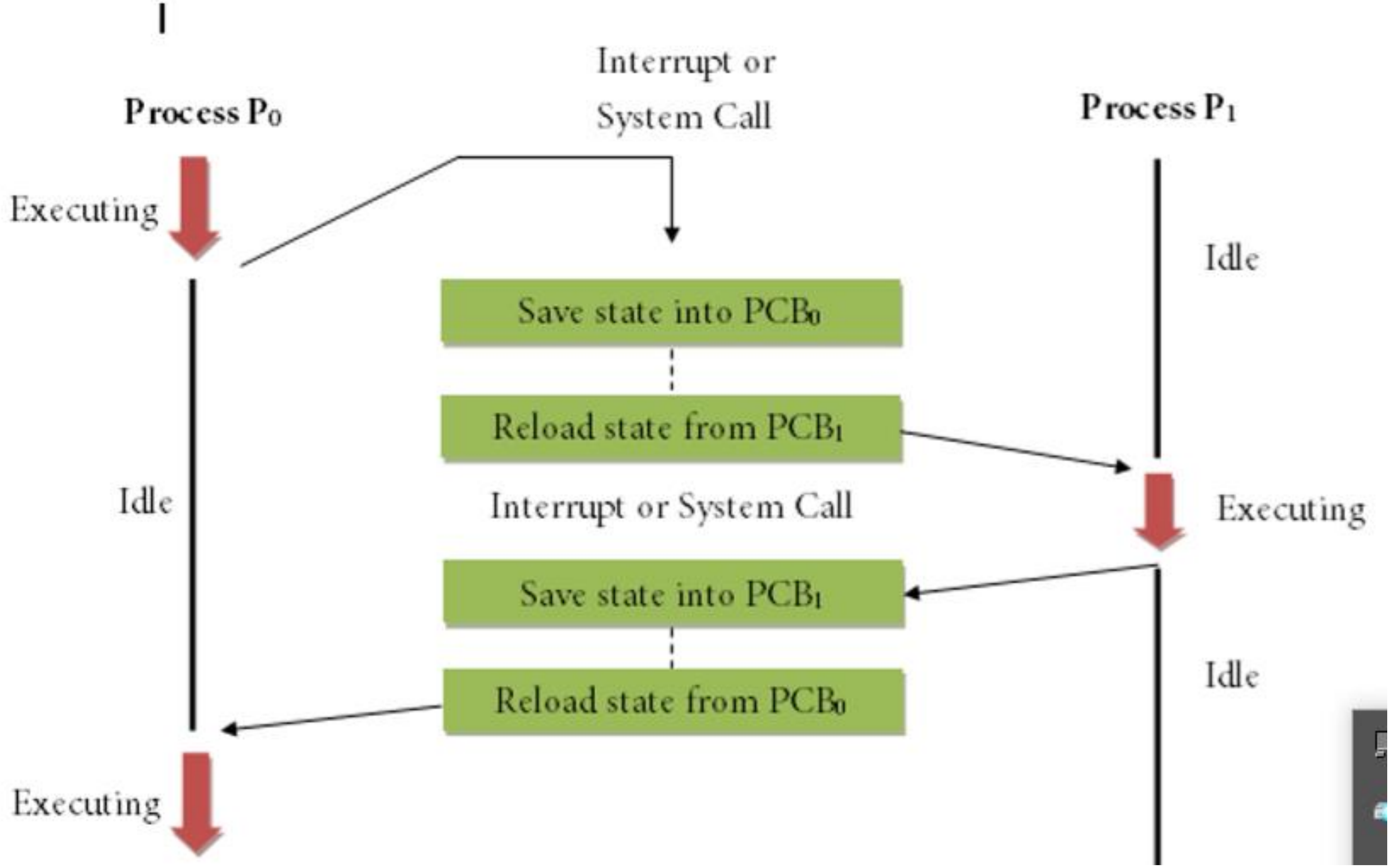


```
[root@linux-server root]#
[root@linux-server root]# objdump -D a.out | grep -A20 main.:
08048460 <main>:
 8048460:       55                      push   %ebp
 8048461:       89 e5                   mov    %esp,%ebp
 8048463:       83 ec 08                sub    $0x8,%esp
 8048466:       90                      nop
 8048467:       c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffc(%ebp)
 804846e:       89 f6                   mov    %esi,%esi
 8048470:       83 7d fc 09             cmpl   $0x9,0xfffffffc(%ebp)
 8048474:       7e 02                   jle    8048478 <main+0x18>
 8048476:       eb 18                   jmp    8048490 <main+0x30>
 8048478:       83 ec 0c                sub    $0xc,%esp
 804847b:       68 08 85 04 08          push   $0x8048508
 8048480:       e8 93 fe ff ff          call   8048318 <_init+0x38>
 8048485:       83 c4 10                add    $0x10,%esp
 8048488:       8d 45 fc                lea    0xfffffffc(%ebp),%eax
 804848b:       ff 00                   incl   (%eax)
 804848d:       eb e1                   jmp    8048470 <main+0x10>
 804848f:       90                      nop
 8048490:       b8 00 00 00 00          mov    $0x0,%eax
 8048495:       c9                      leave
 8048496:       c3                      ret
[root@linux-server root]# _
```

Context Switch

- A context switch is a technique to keep and restore the state or context of a CPU in Process Control block so that execution of a process can be resumed from the same point at a later time.
- By using this mechanism, a context switcher allows multiple processes to use a single CPU.
- When the scheduler exchanges the CPU from one executing process to another execution process, the **state from the current running process** is stored in the **process control block**.
- After this, the state which is needed for the process to run next is installed on its own PCB and applied to set the PC, registers, etc. At this point, the execution of the second process can start.

Context Switch



Context Switch

The following information given below is stored for the later use, when the process is switched:

- Scheduling information
- Currently used register
- Base and limit register value
- I/O State information
- Program Counter
- Changed State
- Accounting information

Operations on Process

Process Creation:

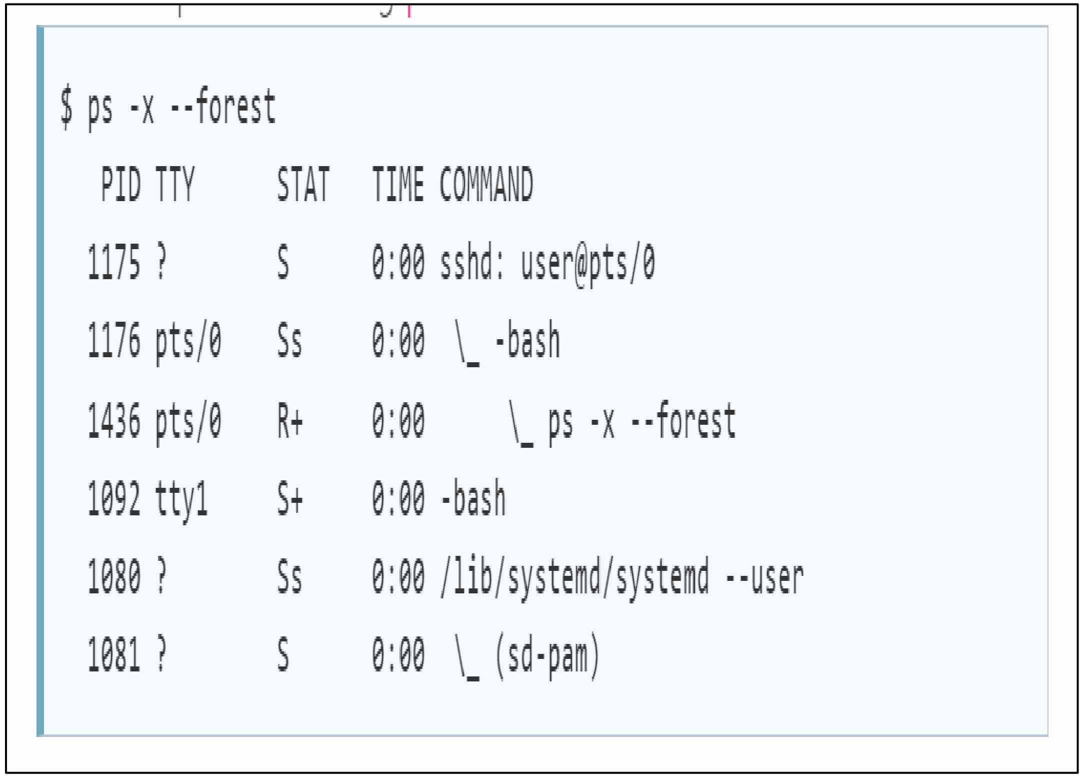
- It can be titled as system initialization, a user request to design a new process, execution of process creation system and lastly, a batch job initialization.

When a process generates a new process, there are two possibilities regarding execution:

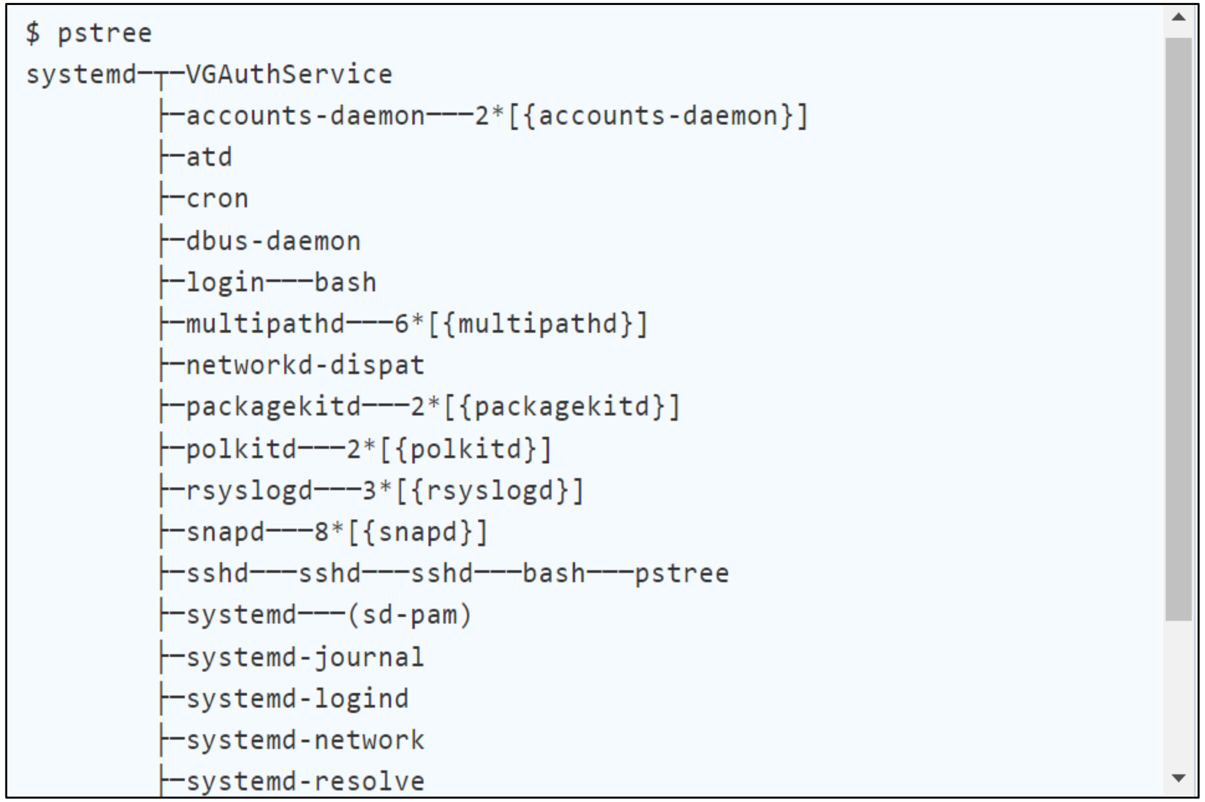
- Simultaneously the parent continues to carry out with its children.
- The parent remains, till some or all of its children have stopped.

Also two possibilities may find its existence regarding the address space of these new processes:

- The child process is a replica of the parent process.
- It contains the same data and program as the parent.
- The child process has a new program loaded into it.



```
$ ps -x --forest
  PID TTY      STAT   TIME COMMAND
 1175 ?        S      0:00 sshd: user@pts/0
 1176 pts/0    Ss     0:00  \_ -bash
 1436 pts/0    R+     0:00      \_ ps -x --forest
 1092 tty1     S+     0:00 -bash
 1080 ?        Ss     0:00 /lib/systemd/systemd --user
 1081 ?        S      0:00  \_ (sd-pam)
```



```
$ pstree
systemd─┬─VGAuthService
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─atd
        ├─cron
        ├─dbus-daemon
        ├─login───bash
        ├─multipathd───6*[{multipathd}]
        ├─networkd-dispat
        ├─packagekitd───2*[{packagekitd}]
        ├─polkitd───2*[{polkitd}]
        ├─rsyslogd───3*[{rsyslogd}]
        ├─snapd───8*[{snapd}]
        ├─sshd───sshd───sshd───bash───pstree
        ├─systemd───(sd-pam)
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-network
        ├─systemd-resolve
```

Operations on Process

Process Termination :

- The goal of the processes is to execute the instructions provided by a program.
- When the process ends accomplishing the last statement, it gets cancelled. This process is known as process termination.

A parent may stop the execution of one of its children for various reasons, such as:

- The child has crossed the usage limit of some of the resources that it has been allotted. For this tracking mechanism of the usage limit, the parent must have a technique to inspect the usage/state of its children.
- No further necessity of the task that is given to the child.
- When the exiting of the parent happens, since the operating system does not allow the child to proceed further after the parent has been terminated.

Cooperating Process

- In the operating system, processes executing simultaneously can be either independent processes or cooperating processes.
- When the sharing of data does not occur between two processes, then that is called as independent.
- A process is known as a cooperating process if it can impact or get influenced by the other processes that are being carried out in the system.
- When the sharing of data takes place between different processes, then that is called as a cooperating process.

Reason for having Cooperating Process

- **Information sharing**: Since various users may be involved in the same part of information (for instance, a shared file), we must give an environment to support simultaneous access to such information.
- **Computation Speedup**: If we require a particular task to execute faster, we must snap it into subtasks, each of which will be carried out in parallel with the others.
- **Modularity**: The solution of a problem is divided into parts having well-defined interfaces, and where the parts execute in parallel. Divide system functions into separate processes or threads.
- **Convenience**: A user may be executing multiple processes to obtain a single objective, or where a utility may apply multiple components, which attach via a pipe structure that connects the stdout of one stage to stdin of the next, etc.

Inter Process Communication (IPC)

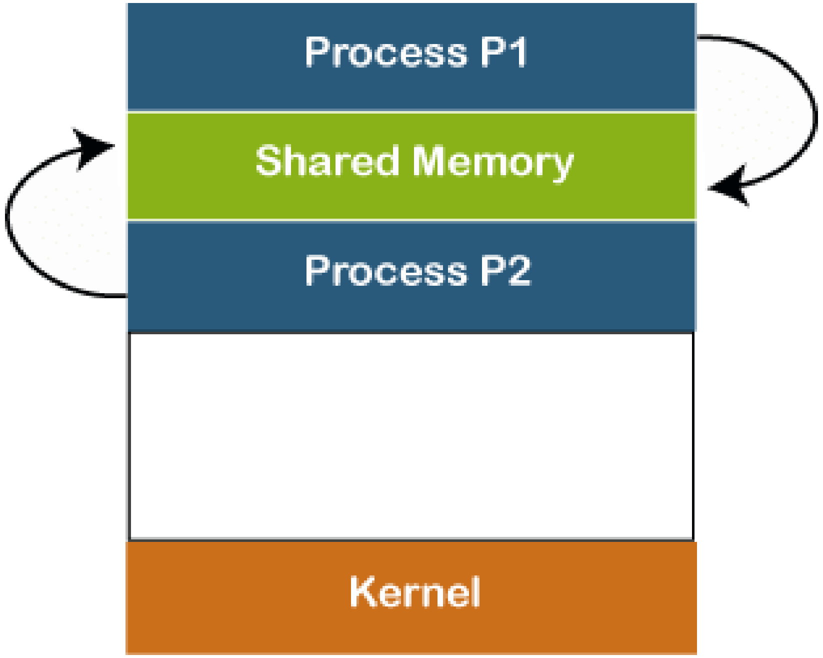
- The processes that are executing in a concurrent fashion, can either be cooperating processes or it can be an independent processes.

The two important models of communication for IPC are as follows:

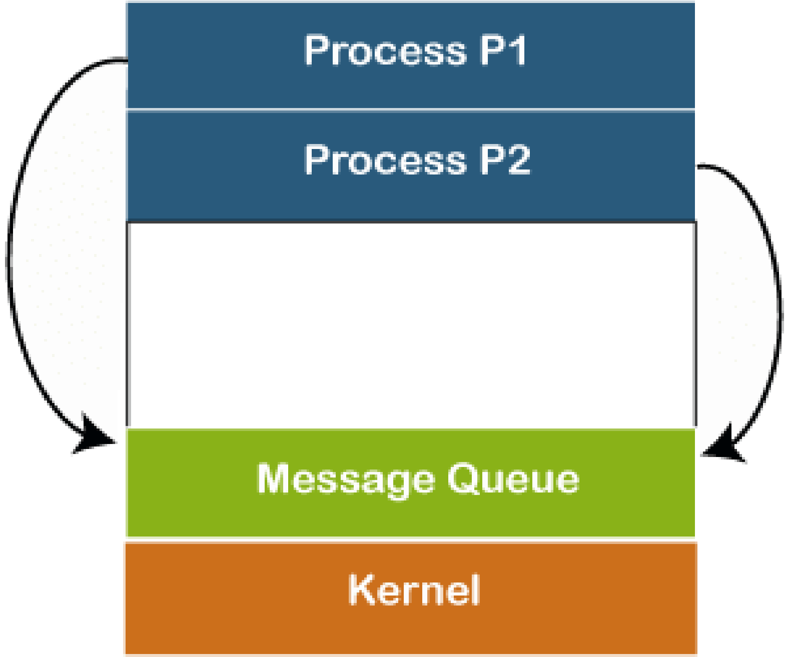
- **Shared Memory:** In this model, a region of the memory is established, which has been shared by the cooperating processes. All the processes that are connected to that shared region can exchange the information. They can also read and write in the shared region.

- **Message Passing**: In this model, the communication happens through the exchange of messages between all the cooperating processes.

Approaches to Interprocess Communication



Shared Memory



Message Queue

Shared Memory Model :

- In this IPC model, a shared memory region is established which is used by the processes for data communication.
- This memory region is present in the address space of the process which creates the shared memory segment.
- The processes who want to communicate with this process should attach it to their address space.
- Producer-Consumer Problem – unbounded buffer and bounded buffer

Message Passing Model :

- In this model, the processes communicate with each other by exchanging messages.
- For this purpose a communication link must exist between the processes and it must facilitate at least two operations send (message) and receive (message).
- Size of messages may be variable or fixed.
- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

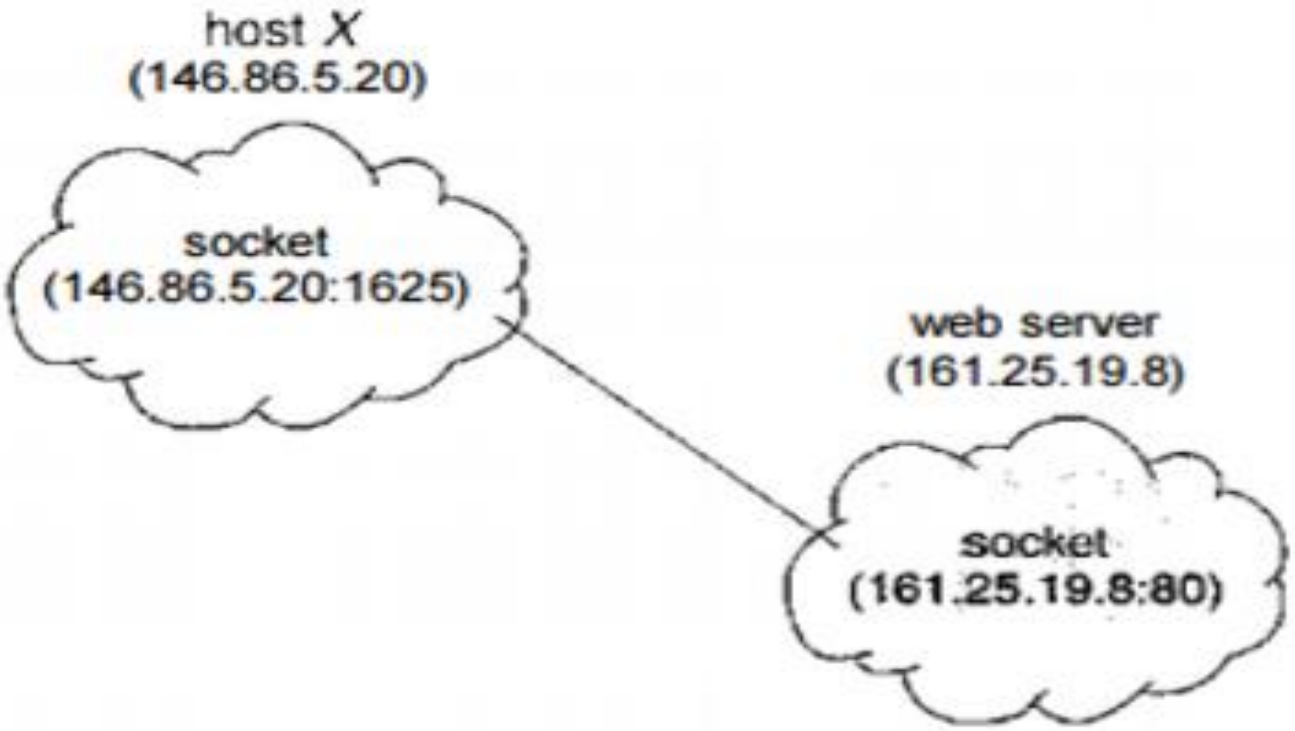
Communication in Client-Server System

Happens via **RPC (remote procedure calls), sockets, and Java's RMI (remote method invocation)**.

Sockets

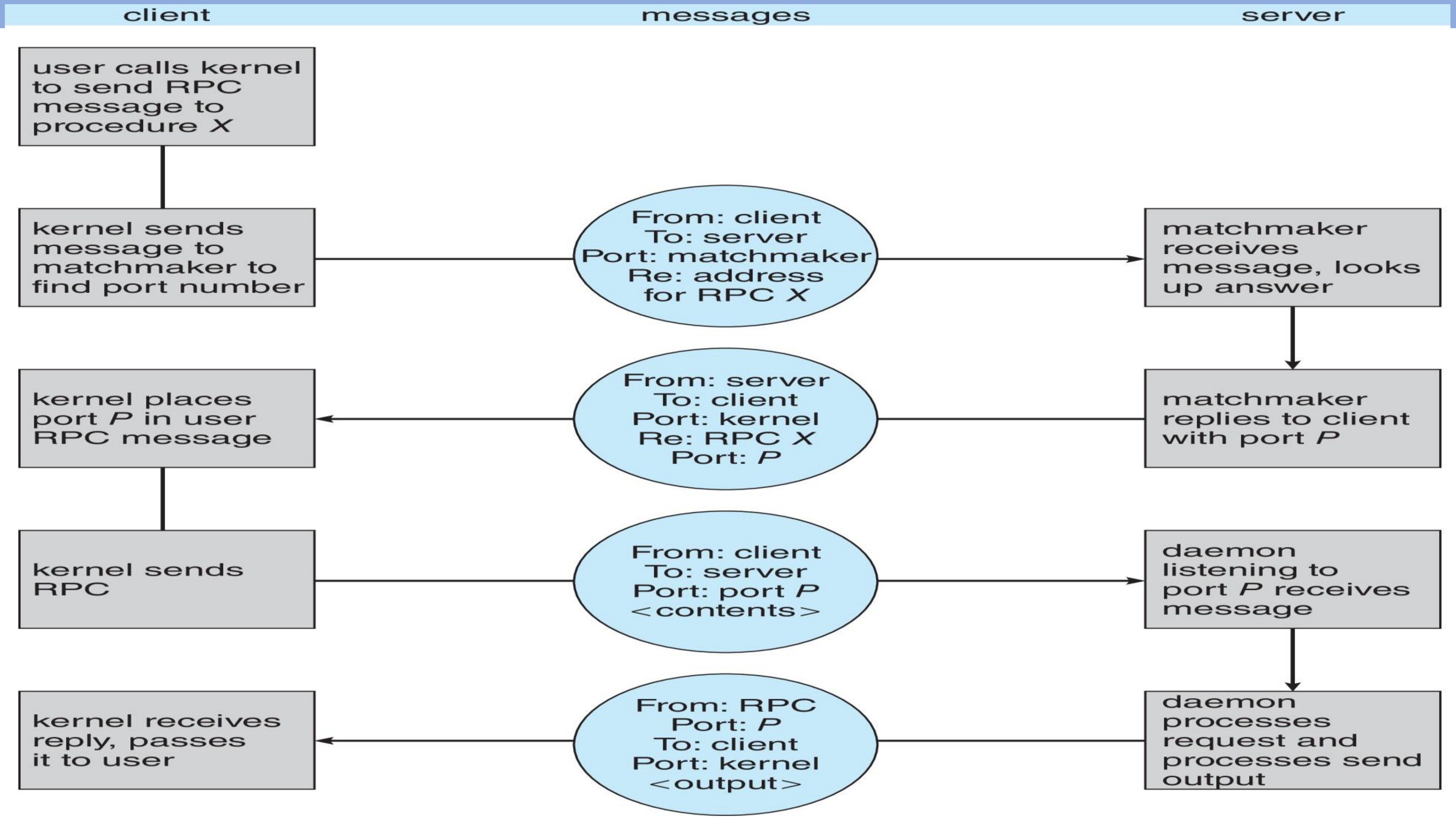
- A socket is considered to be an endpoint for the communication.
- A set of processes that are communicating over a network applies a set of sockets (one for each of the processes).
- A socket is recognised by an IP address along with a port number.
- The server waits for an incoming client request by following a specified port.
- Once a request is obtained, the server acquires a link from the client socket to establish a connection.

IPC via Sockets between client and server



Remote Procedure Calls(RPC)

- The Remote Procedure Calls was planned in a way to separate the usage of the procedure call mechanism between the systems including the network connections.
- Each message is directed to a Remote Procedure Calls daemon that is listening to a port on the remote system, and all of them have an identifier of the function and the parameters to execute and pass the function respectively.
- Then the function is accomplished as requested, and the output is then sent to the requester through a separate message.



client

messages

server

user calls kernel to send RPC message to procedure *X*

kernel sends message to matchmaker to find port number

From: client
To: server
Port: matchmaker
Re: address for RPC *X*

matchmaker receives message, looks up answer

matchmaker replies to client with port *P*

From: server
To: client
Port: kernel
Re: RPC *X*
Port: *P*

kernel places port *P* in user RPC message

kernel sends RPC

From: client
To: server
Port: port *P*
<contents>

daemon listening to port *P* receives message

daemon processes request and processes send output

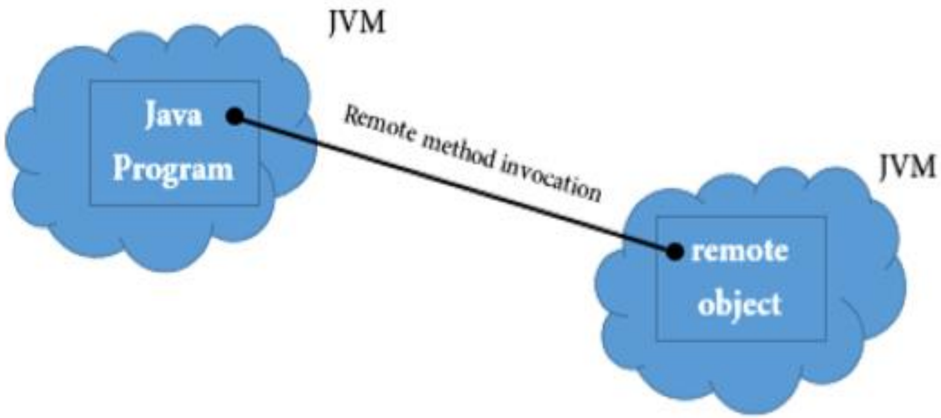
From: RPC
Port: *P*
To: client
Port: kernel
<output>

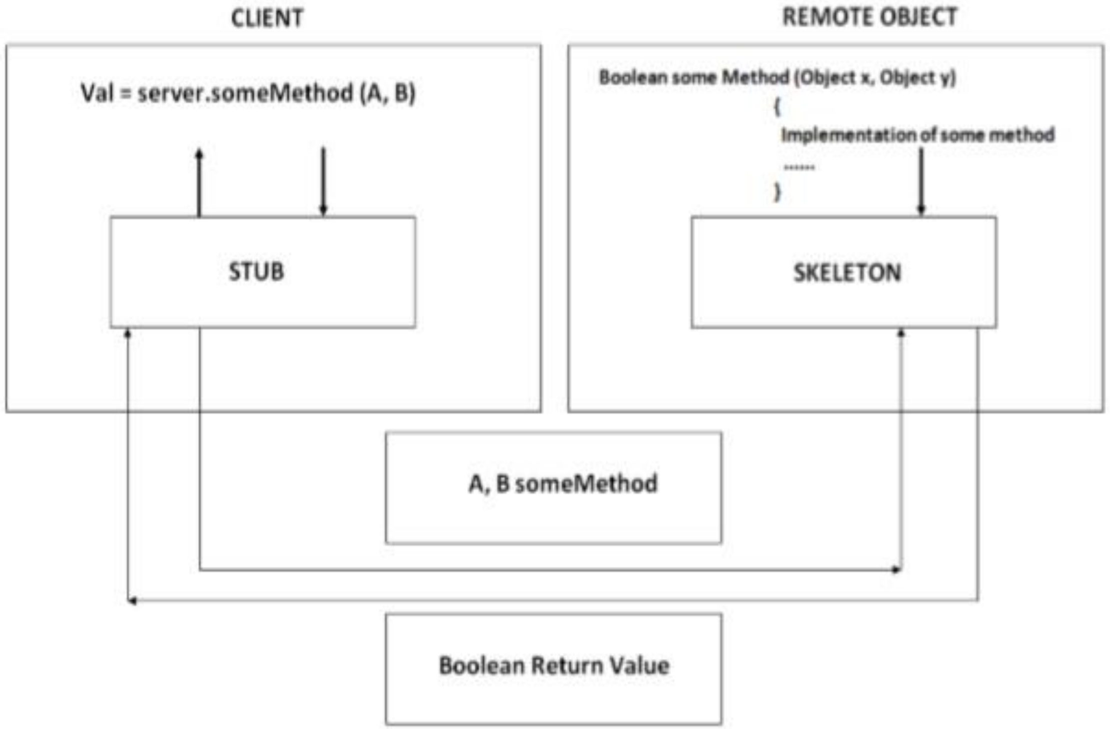
kernel receives reply, passes it to user

Remote Method Invocation (RMI)

- The Remote Method Invocation enables a thread to request a method on the remote object.
- The objects are viewed as remote if they are in a different JVM (Java virtual machine).
- Hence, the remote object can be on a remote host that is connected by a network or a different Java virtual machine on the same computer

Remote Method Invocation (RMI)



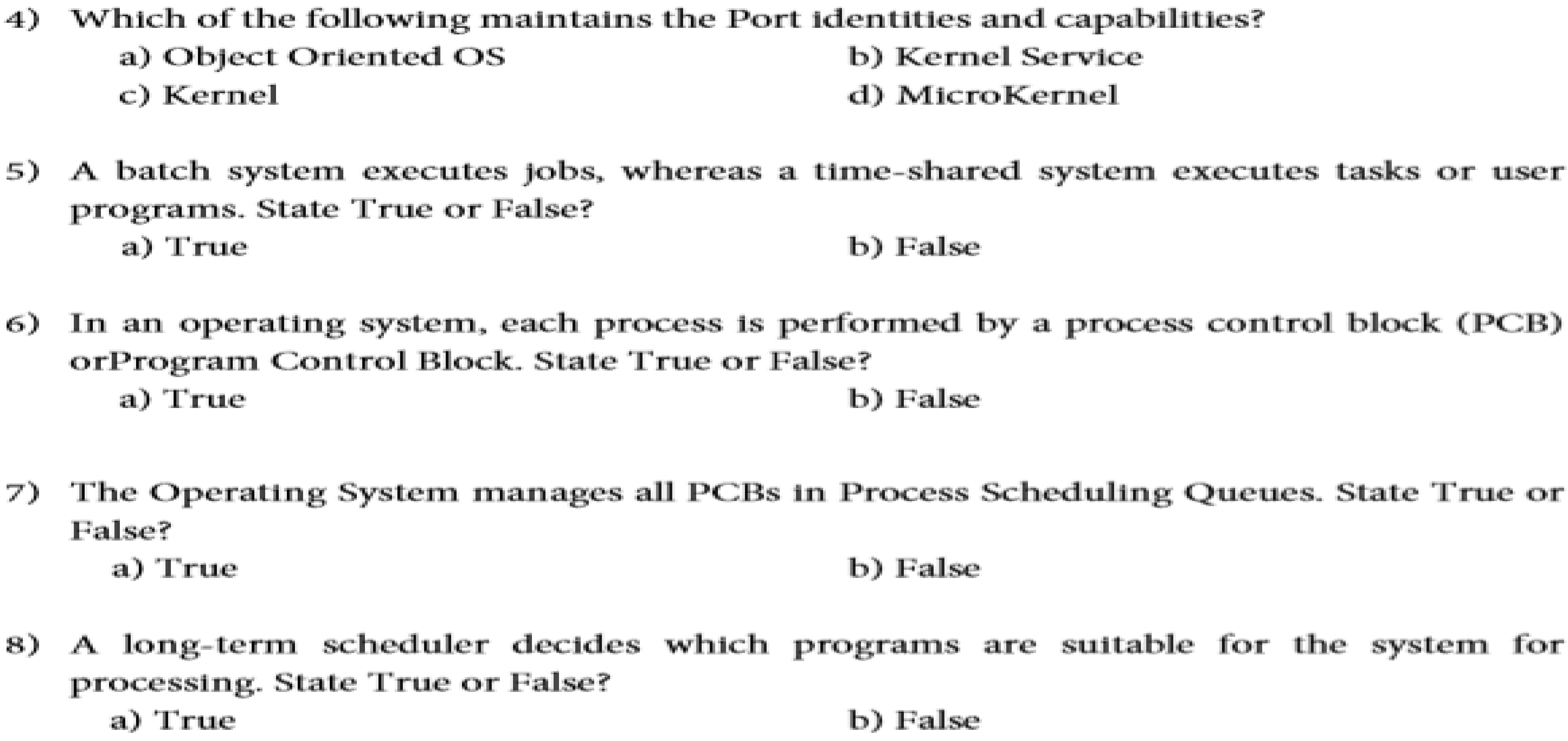

CLIENT
REMOTE OBJECT
Val = server.someMethod (A, B)
Boolean some Method (Object x, Object y)
{
Implementation of some method
......
}
STUB
SKELETON
A, B someMethod
Boolean Return Value

Self-assessment Questions

1) What happens when the process issues an I/O Request?
   - a) It is placed in an I/O Queue
   - b) It is placed in a waiting Queue
   - c) It is placed in the ready Queue
   - d) It is placed in the job Queue

2) If all the processes I/O bound, the ready queue will almost always be ______, and the short term scheduler will have a _____ to do.
   - a) Full, Little
   - b) Full, Lot
   - c) Empty, Little
   - d) Empty, Lot

3) In a time-sharing operating system, when the time slot given to a process is completed, the process goes from a running state to ____________ state?
   - a) Blocked
   - b) Ready
   - c) Suspended
   - d) Terminated



4) Which of the following maintains the Port identities and capabilities?

a) Object Oriented OS	b) Kernel Service
c) Kernel	d) MicroKernel

5) A batch system executes jobs, whereas a time-shared system executes tasks or user programs. State True or False?

a) True	b) False

6) In an operating system, each process is performed by a process control block (PCB) orProgram Control Block. State True or False?

a) True	b) False

7) The Operating System manages all PCBs in Process Scheduling Queues. State True or False?

a) True	b) False

8) A long-term scheduler decides which programs are suitable for the system for processing. State True or False?

a) True	b) False

Threads

- A thread is the smallest unit of processing that can be performed in an OS.
- In most modern operating systems, a thread exists within a process - that is, a single process may contain multiple threads.
- A thread shares the data section, code section, and other resources of the operating system with the other threads from the same process. A process has a single thread.
- If any process has multiple threads, then it can perform multiple tasks.
- A thread is essential for sharing address space within a process because the cost of communication between threads is low. It uses same code section, data section, and OS resources. So that threads are also known as a lightweight process

Threads

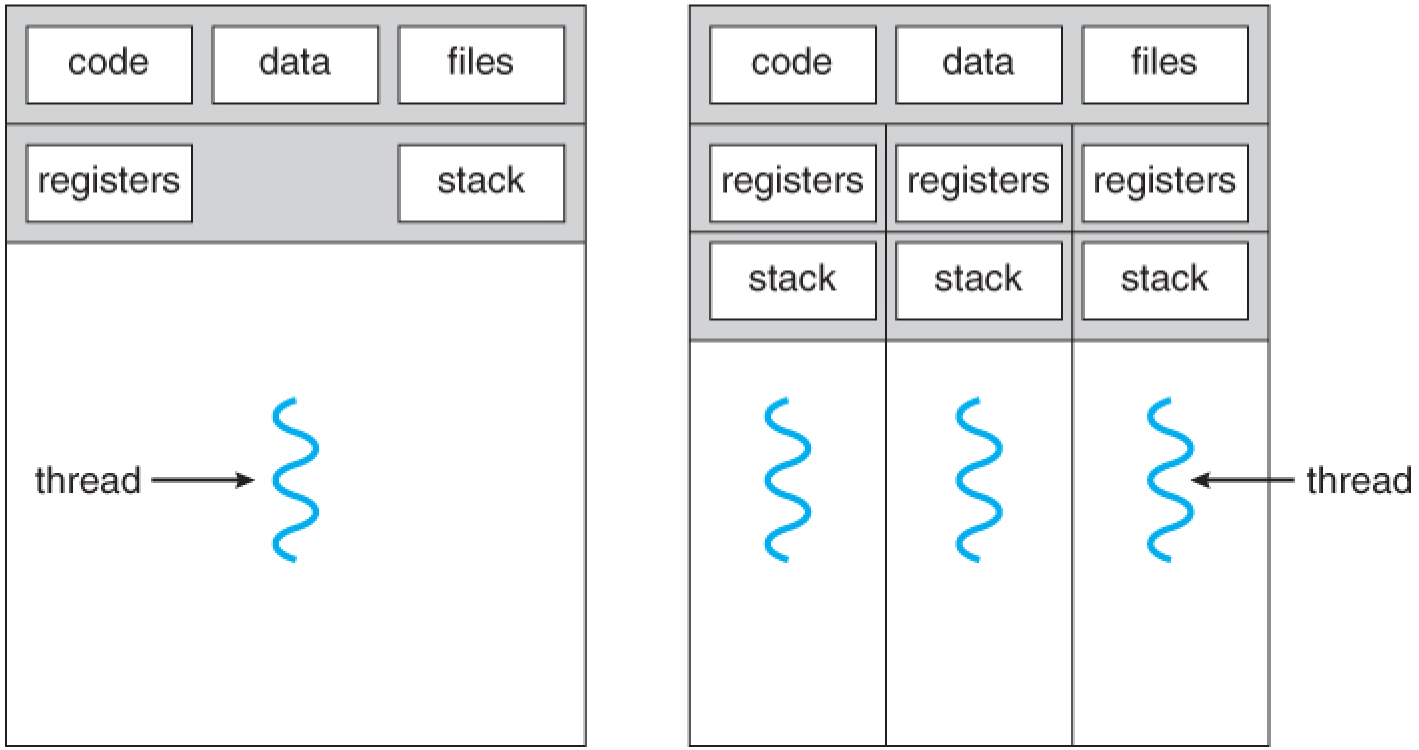
- The implementation of single threaded process within a server can accept only one client at a time and can halt the other users requesting services for a long time.

The benefits of single thread process are:

- It is not a complex process.
- Fewer overheads added by a thread to an application.

Multi Thread Process and Benefits

- Multithreading is the process of a CPU (Central Processing Unit) to handle several processes or threads simultaneously.
- This concept is used by the operating system to control its utilization simultaneously by multiple users.
- It handles the multiple requests made by a single user on a single program.



single-threaded process

multithreaded process

Benefits of Multithreading

- **Responsiveness:** Multi-threading collective application may provide a program to carry on execution even if part of it is closed or is operating a lengthy operation, thereby enhancing responsiveness to the user.
- **Resource Sharing:** Typically, threads give the memory and the resources of the process in which they exist.
- **Economy**: It is expensive allotting memory and resources for process creation. As threads provide resources of the process in which they exist, it is more cost effective to make and context-switch threads.
- **Utilization of multiprocessor architectures:** The advantages of multithreading can be largely enhanced in a multiprocessor structure, where threads can be executed parallel on various processors

User and Kernel Threads

User Thread: In user thread, the thread management kernel does not know the occurrence of threads. The application begins with a single thread.

Advantages

- Thread switching does not need Kernel mode authorisation.
- User level thread can execute on any operating system.
- Scheduling can be applied specifically to the user level thread.

Disadvantages

- In a normal operating system, most system calls are prevented.
- The multi-threaded application is unable to take advantage of multiprocessing

User and Kernel Threads

- In kernel thread process, thread management is accomplished by the Kernel.
- Kernel threads are provided directly by the operating system.

Advantages

- The kernel can simultaneously organize multiple threads from the same process on multiple processes.
- If one thread in a process is obstructed, the Kernel can set up another thread for the same process.
- Kernel routines can be multi-threaded.

User and Kernel Threads

The disadvantages of kernel threads are:

- Usually, Kernel threads are moderate to design and handle than the user threads.
- Mode switch to the kernel is required within the same process for the transfer of control from a thread to another.

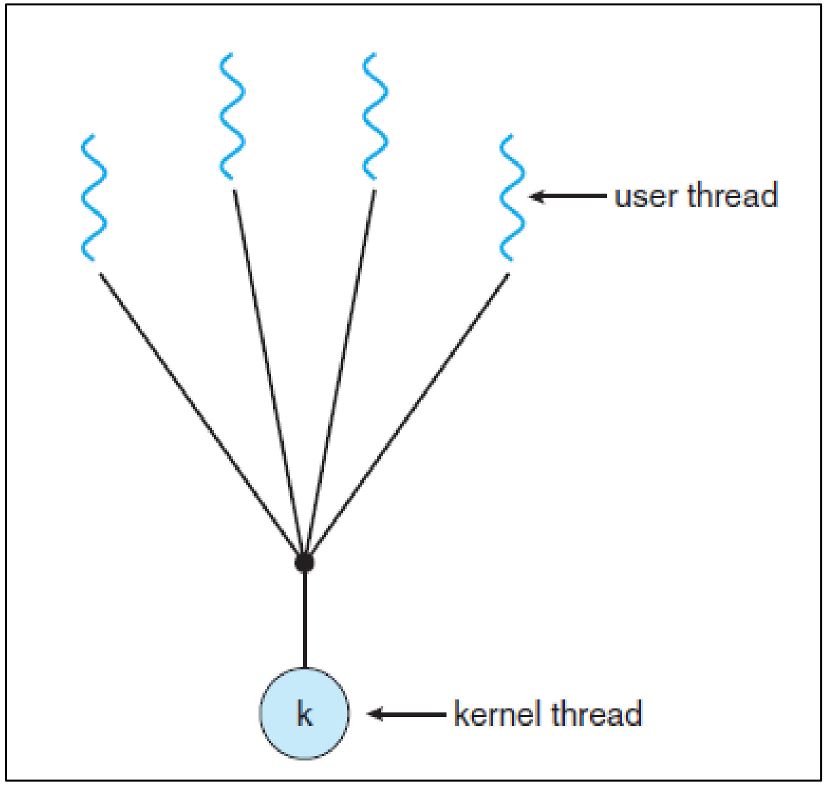
Multi-Threading Model

- The two systems i.e. combined user level thread and Kernel level thread are provided by some operating system.
- Solaris is an example of this combined approach.
- In this combined approach, the same application consisting of multiple threads can execute in parallel on multiple processors, and there is no need of blocking the entire process by a blocking system call

Types of Multi-Threading Model

Many to One model: In the many-to-one model, many user-level threads to one kernel thread.

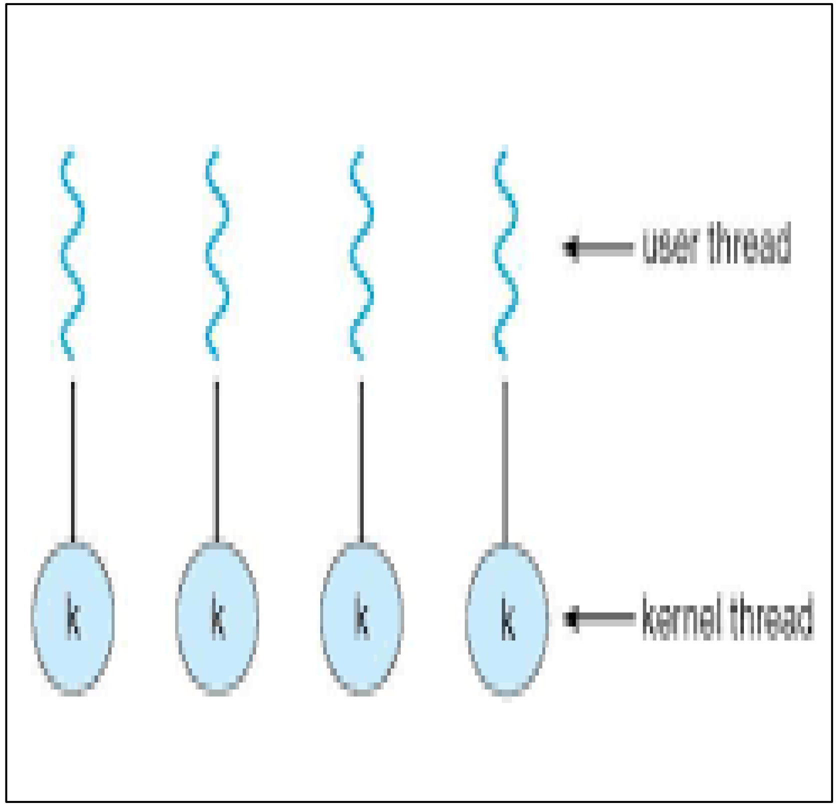
- Thread management is done by the thread library in user space so it is efficient, but the entire process will block if a thread makes a blocking system call.
- Only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.



Types of Multi-Threading Model

One to One model:

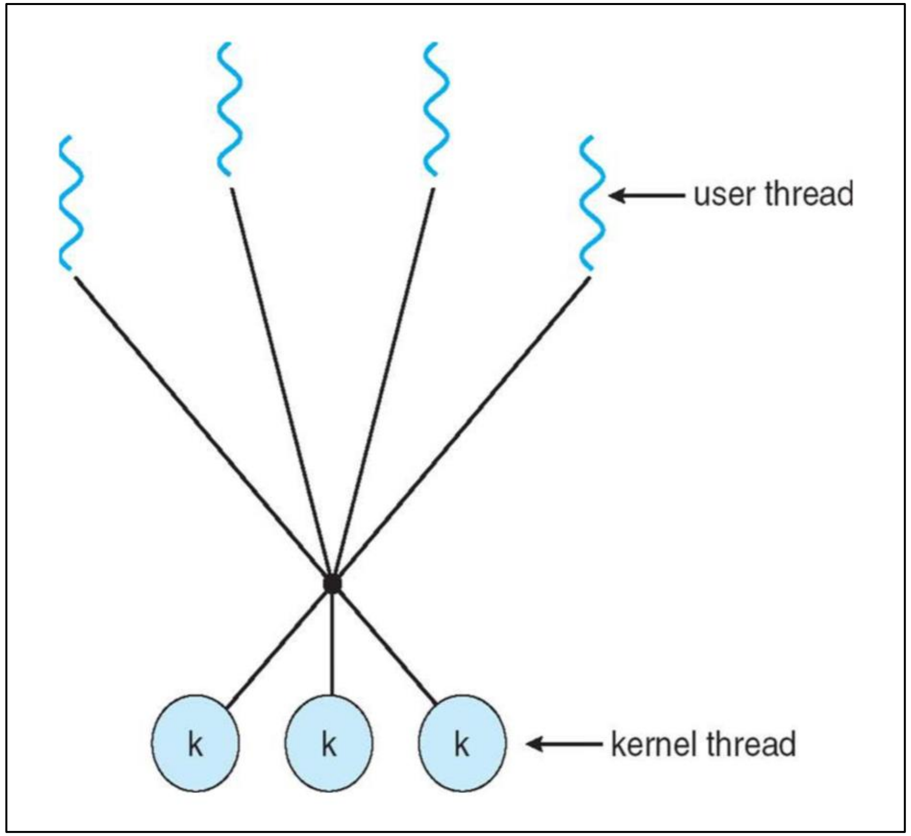
- The one-to-one model designs each user thread to a kernel thread.
- It gives more compatibility and concurrency than the many-to-one model by providing another thread to execute when a thread creates a blocking system call.
- It also allows multiple threads to run in parallel on multiprocessors.
- Drawback: creating a user threads requires creating the corresponding kernel thread, which burden the performance of an application



Types of Multi-Threading Model

Many to Many model: In the many-to-many model, many user-level threads are collected to a smaller or equal number of kernel threads.

- An application may be given more kernel threads on a multiprocessor as compared to a uniprocessor.
- The developers can design as much user threads according to their need, and the equivalent kernel threads can execute in parallel on a multiprocessor.



Threading issue

Cancellation: Thread cancellation is the process of cancelling a thread before it has accomplished.

A thread which is to be aborted is sometimes specified as the target thread. Termination of a target thread may take place in two different scenarios:

- **Asynchronous cancellation**: One thread instantly cancels the target thread.
- **Deferred cancellation**: The target thread regularly analyses whether it should cancel or provide it an opportunity to cancel itself in an orderly fashion

Threading issue

- **Signal Handling:** A signal is applied in UNIX systems to inform a process that a specific event has taken place. Signals can be synchronous or asynchronous.
- A signal is developed by the existence of a particular event.
- A developed signal is passed to a process.
- Once delivered, the signal must be controlled.

Every signal can be controlled by one of the following possible **handlers**:

- A default signal handler
- A user-defined signal handler

Threading issue

- **Thread Pools**
- **Thread-Specific Data**

Self-assessment Questions

9) What happens when the thread is blocked?

a) Thread moves to the ready queue

b) Thread remains blocked

c) Thread completes

d) A new thread is provided

10) Which of the following is terminated when the termination of process takes place?

a) First thread of the process

b) First two threads of the process

c) All threads within the process

d) No Thread within the process

11) Which of the following is not a valid state of the thread?

a) Running

b) Parsing

c) Ready

d) Blocked

Process Scheduling

- The process scheduling is defined as the action of the process manager which manages the replacement of the functioning process from the CPU and the choosing of another process by a specific strategy.

- Process scheduling is an important part of multiprogramming operating systems.

Process Scheduling

Scheduling Queues:

- The Operating System manages all PCBs in Process Scheduling Queues.

- For each of the process states, the OS keeps a separate queue and PCBs of all processes in the same execution state are put in the same queue.

- When the state of a process is modified, its PCB is separated from its current queue and switched to its new state queue

Process Scheduling Queues

- **Job queue**: This queue maintains all the processes in the system.
- **Ready queue**: This queue maintains a set of all processes residing in main memory, prepared and waiting to be carried out. A new process is always kept in this queue.
- **Device queue**: The processes which are stopped due to the absence of an I/O device constitute this queue.

Schedulers

- **Schedulers**: Schedulers are special system software which manages process scheduling in various ways.

- Their main task is to choose the right jobs to be put into the system and to determine which process to execute.

Types of Schedulers

Schedulers are of three types:

- **Long-Term Scheduler**: It is also known as a **job scheduler**. Long-Term schedulers are those schedulers whose decision will have a long-term effect on the performance. The duty of the long-term scheduler is to bring the process from the JOB pool to the Ready state for its execution.
- **Short term Scheduler**: It is also known as **CPU scheduler**. The main objective of this scheduler is to enhance system performance following the selected set of criteria. It is the transformation of the ready state to running state of the process. CPU scheduler chooses a process among the processes that are prepared to carry out and allocates CPU to one of them.

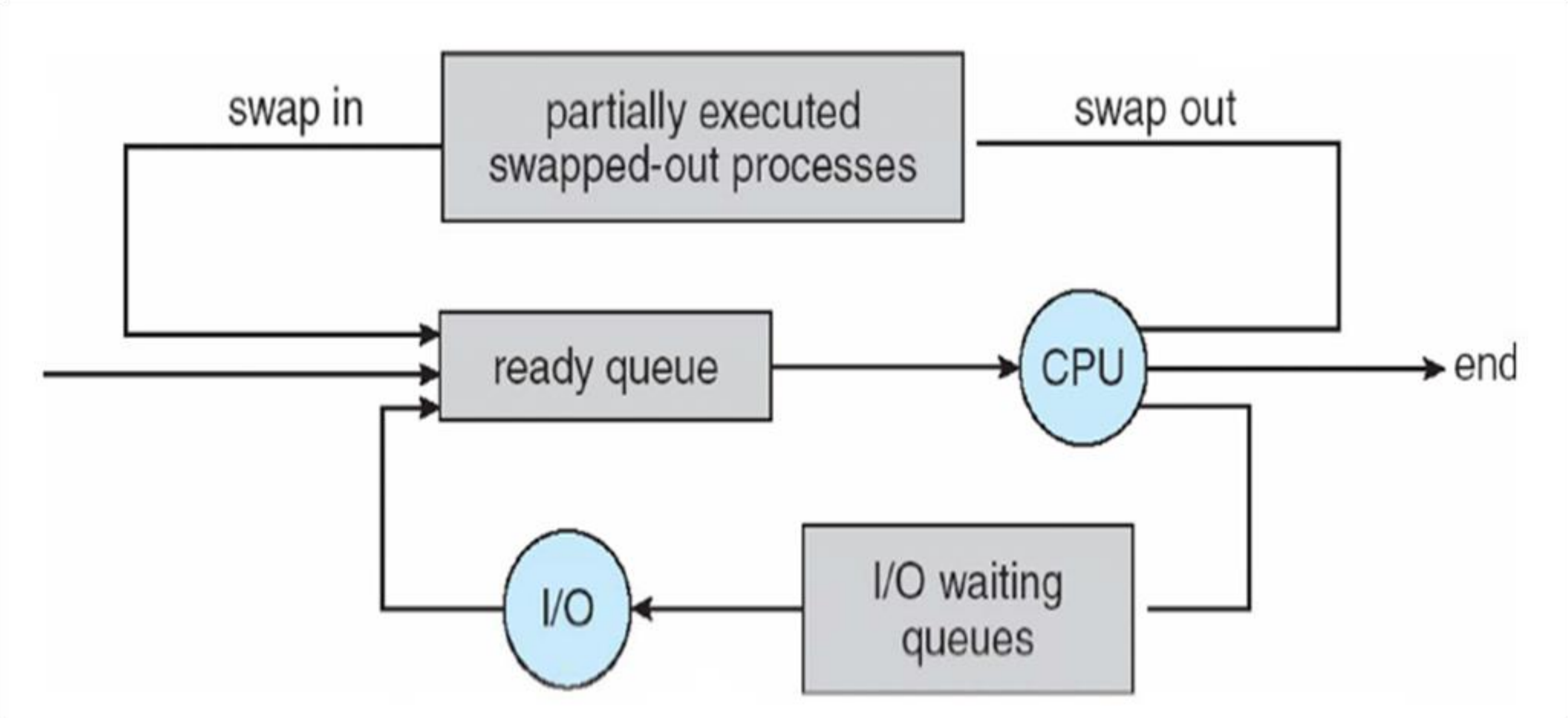
Types of Schedulers

- **Medium term scheduler**: Medium-term scheduling is considered as a component of **swapping**. In this scheduler, the processes are removed from the memory. It decreases the degree of multiprogramming. It is also responsible for controlling the swapped out processes.

Swapping

- A running process can be turned into suspended if it makes an I/O request.
- A suspended process cannot cause any progress towards completion.
- In this situation, to remove the process from memory and create space for other processes the suspended process is shifted to the secondary storage.
- This process is known as swapping, and the process is declared to be rolled out or swapped out

Medium Term Scheduler



Types of Schedulers

Types of Schedulers: There are three types of Scheduler:

1. Short Term Scheduler
2. Medium Term Scheduler
3. Long Term Scheduler

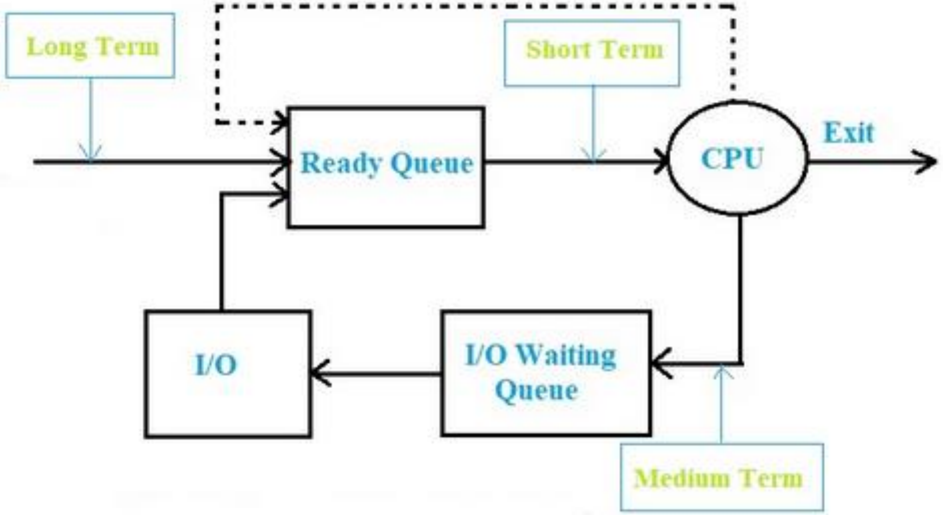


Fig)- Schedulers

CPU Scheduling

- CPU scheduling is a process which permits one process to utilize the CPU while the carrying out of another process is in a waiting state due to deficiency of any resource like I/O etc., thereby using of CPU fully.
- The aim of CPU scheduling is to create the system efficient, fast and fair.

CPU Scheduling

CPU-I/O Burst Cycle:

- The progress of CPU scheduling relays upon a noticed property of processes.
- Process execution contains a cycle of **CPU execution and I/O wait**.
- Processes substitute between these two cases. Process execution starts with a CPU burst.
- That is supported by an I/O burst, which is supported by another CPU burst, then another I/O burst, and so on.

CPU Scheduler

- Whenever the CPU turns unproductive, the operating system must choose one of the processes in the ready queue to be carried out.
- The selection process is executed by the short-term scheduler or CPU scheduler.
- The **scheduler** chooses a process from the processes in memory that are prepared to carry out and assigns the CPU to that process.
- The ready queue is not certainly a **First-in First-out (FIFO)** queue

Preemptive and Non-preemptive Scheduling

CPU-scheduling decisions can occur under the following **four** factors:

- When a process shifts from the **running state to the waiting state**. For example, on account of an I/O request or an appeal of wait for the completion of one of the child processes) .
- When a process shifts from the **running state to the ready state**. For example, when an interruption takes place.
- When a process shifts from the **waiting state to the ready state**. For example, at the termination of I/O.
- When a process **completes**.
- When scheduling occurs only under **factors 3 and 4**, we can state that the scheduling scheme is **non-preemptive**; otherwise, it is **pre-emptive**

Scheduling Criteria

- **CPU utilization**: We should keep using the CPU as much as possible. Generally, CPU utilization can extend from 0 to 100 percent.
- **Throughput**: Throughput is defined as the number of processes that is carried out per time unit.
- **Waiting Time:** time spend in waiting

Scheduling Criteria

- **Turnaround time**: From the perspective of a particular process, the important criterion is how long it requires carrying out that process.

  Turnaround time = Exit time - Arrival time

- **Response time**: The time from the compliance of a request till the first response is created. This method is called response time.

Response time = Time at which the process gets the CPU for the first time - Arrival time

Scheduling Algorithms

- CPU Scheduling manages the problem of deciding which of the processes in the prepared queue is to be allotted to the CPU.

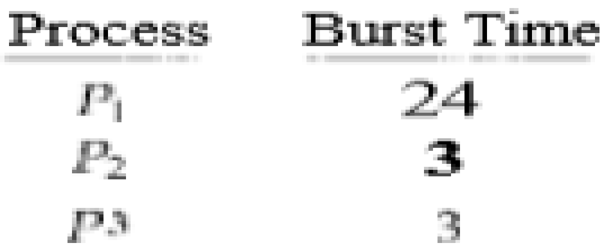
CPU Scheduling Algorithms:

1)First-Come, First-Served (FCFS) scheduling :

- The Jobs are carried out on a first come first serve basis.
- It is a **non-preemptive** a scheduling algorithm.
- Inefficient in performance as the average wait time is high

Scheduling Algorithms –FCFS example

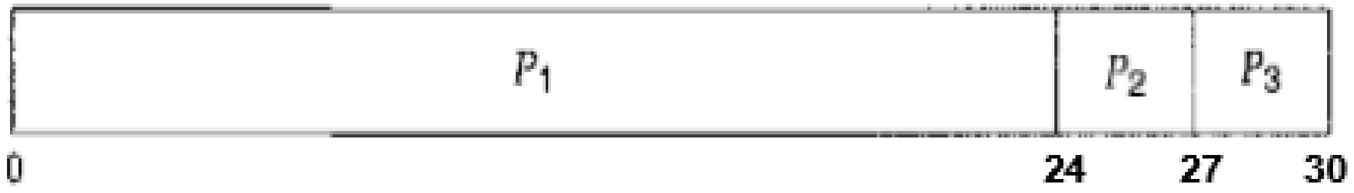
- The average waiting time under the FCFS policy, however, is often quite long. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:



Process	Burst Time
P_1	24
P_2	3
P_3	3

If the processes arrive in the order P1, P2, P3, and are served in FCFS order, we get the result shown in the following Gantt chart (next slide)

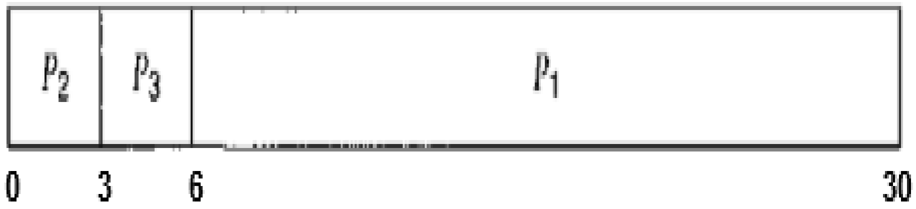
Scheduling Algorithms –FCFS example



The waiting time is 0 milliseconds for process Pi, 24 milliseconds for process Pn, and 27 milliseconds for process Pj. Thus, the average waiting time is (0 + 24 + 27)/3 = 17 milliseconds. If the processes arrive in the order P2, P3, P1, however, the results will be as shown in the following Gantt chart (next slide)

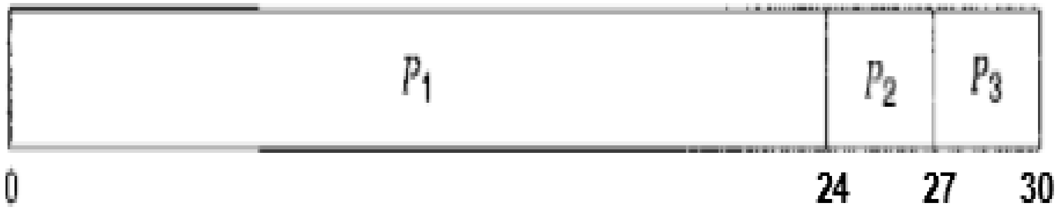
Scheduling Algorithms –FCFS example

- The average waiting time is now (6 + 0 + 3)/3 = 3 milliseconds.



The average waiting time is now **(6 + 0 + 3)/3** = 3 milliseconds. This reduction is substantial. Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the process's CPU burst times vary great.

Scheduling Algorithms –FCFS example



Response Time of p1=0 p2=24   p3 =27

Average response Time= 0+ (24-0)+ (27-0)

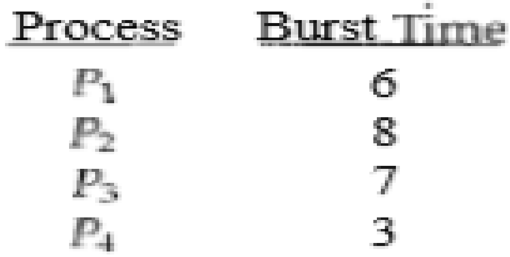
Throughput= 3/30 =1/10

Average Turn around time= E.t-A.t = (24-0) + (27-0)+ (30-0)/3

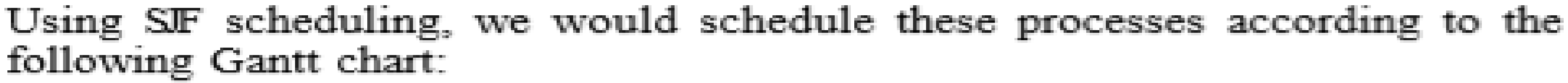
Shortest-Job-First (SJF) Scheduling

- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If the next CPU bursts of two processes are **the same, FCFS scheduling is used** to break the tie.
- Note that a more appropriate term for this scheduling method would be the shortest-next-CPU-burst algorithm, because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

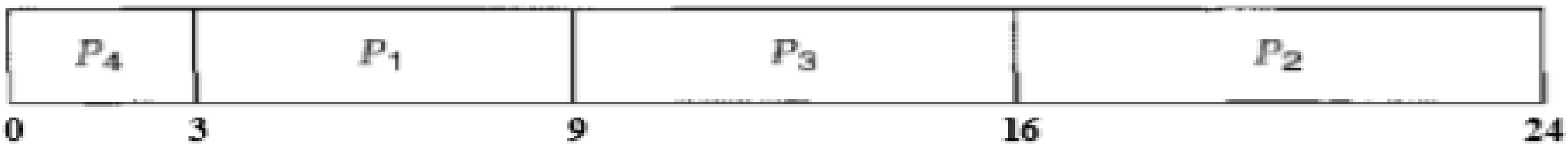
Shortest-Job-First (SJF) Scheduling



Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3



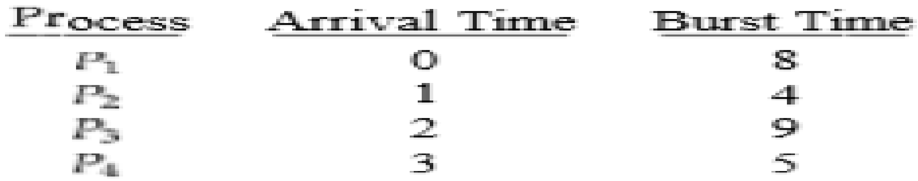
Using SJF scheduling, we would schedule these processes according to the following Gantt chart:



The waiting time is 3 milliseconds for process P1, 16 milliseconds for process P2, 9 milliseconds for process P3, and 0 milliseconds for process P4. Thus, the average waiting time is (3 + 16 + 9 + 0)/4 - 7 milliseconds

SJF with Arrival Time/Preemptive

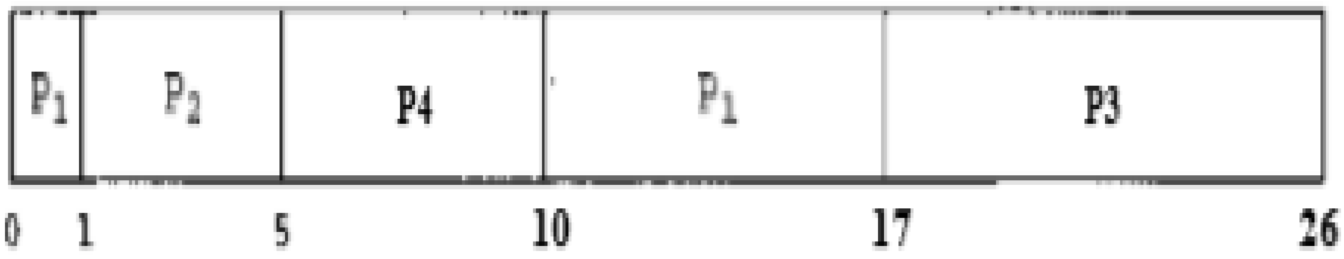
- **Preemptive** SJF scheduling is sometimes called shortest-remaining-time-first scheduling.



Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

If the processes arrive at the ready queue at the times shown and need the indicated burst times, then the resulting preemptive SJF schedule is as depicted in the following Gantt chart (next slide)

SJF with Arrival Time



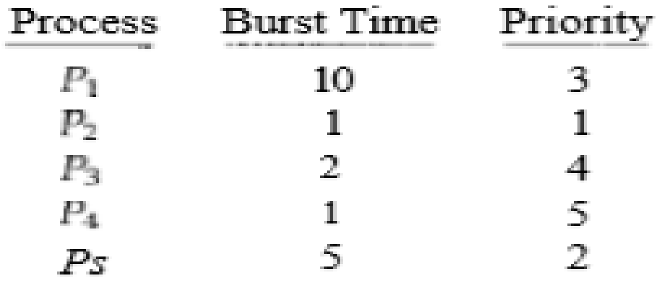
Process P1 is started at time 0, since it is the only process in the queue.
Process P2 arrives at time 1.
The remaining time for process P1 (7 milliseconds) is larger than the time required by process P2 (4 milliseconds), so process P1 is preempted, and process P2 is scheduled.
The average waiting time for this example is ((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3))/4 = 26/4 = 6.5 milliseconds.
Nonpreemptive SJF scheduling would result in an average waiting time of 7.75 milliseconds.

Priority Scheduling

- Priority scheduling can be either **pre-emptive** or **non-preemptive** algorithm, and in batch systems, it is considered as one of the most common scheduling algorithms.
- Each process is assigned a priority.
- The process which has the **highest priority** needs to be executed first.
- Processes with the **same priority** are executed on a **first come first serve basis.**
- Prioritization can be done based on the memory demand, time necessities or some other resource need.

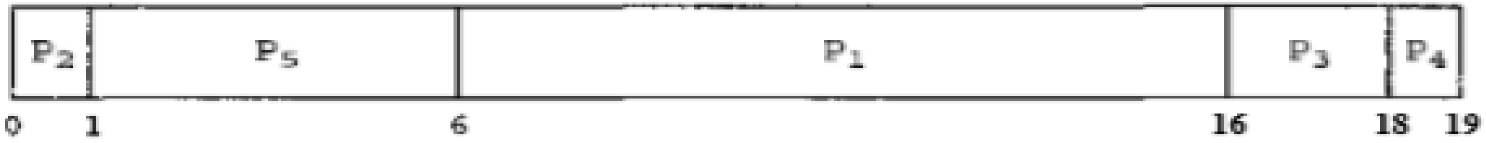
Priority Scheduling (Non-preemptive)

Considering lower value as high priority process



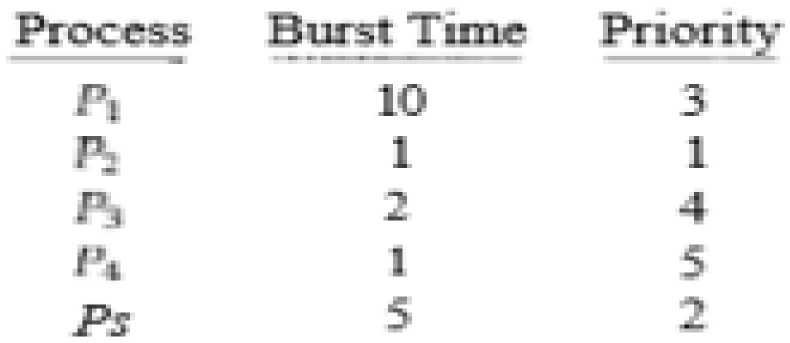
Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Using priority scheduling, we would schedule these processes according to the following Gantt chart:



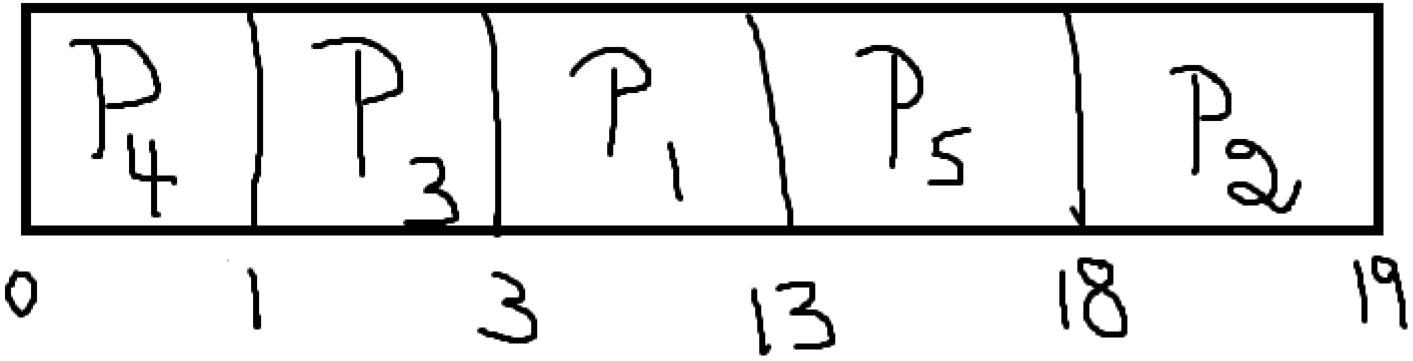
The average waiting time is 8.2 milliseconds.

Priority Scheduling (Non-preemptive)

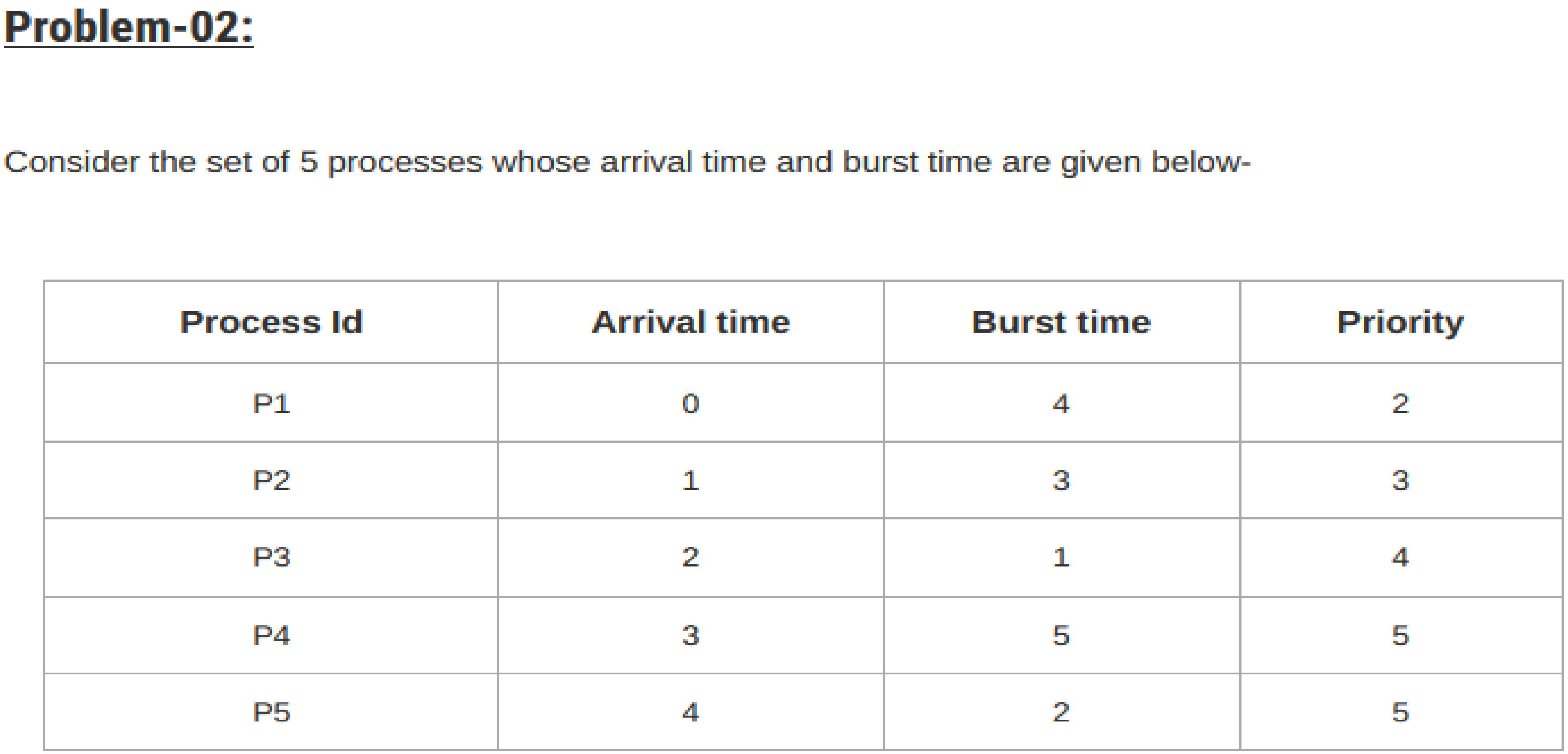


Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Considering higher value as high priority process



Priority Scheduling (preemptive)



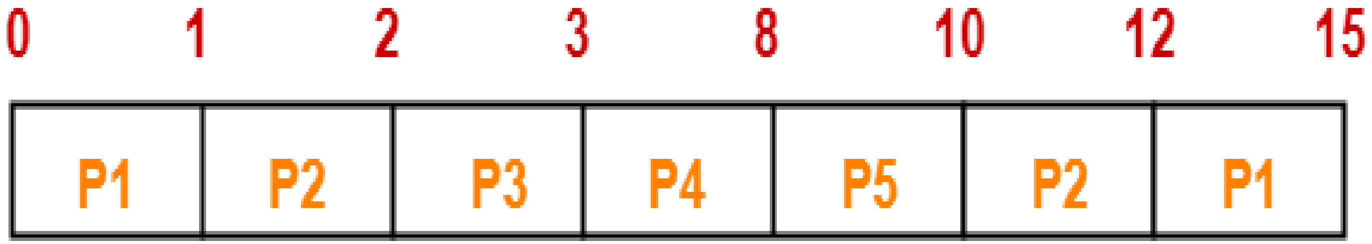
Problem-02:

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

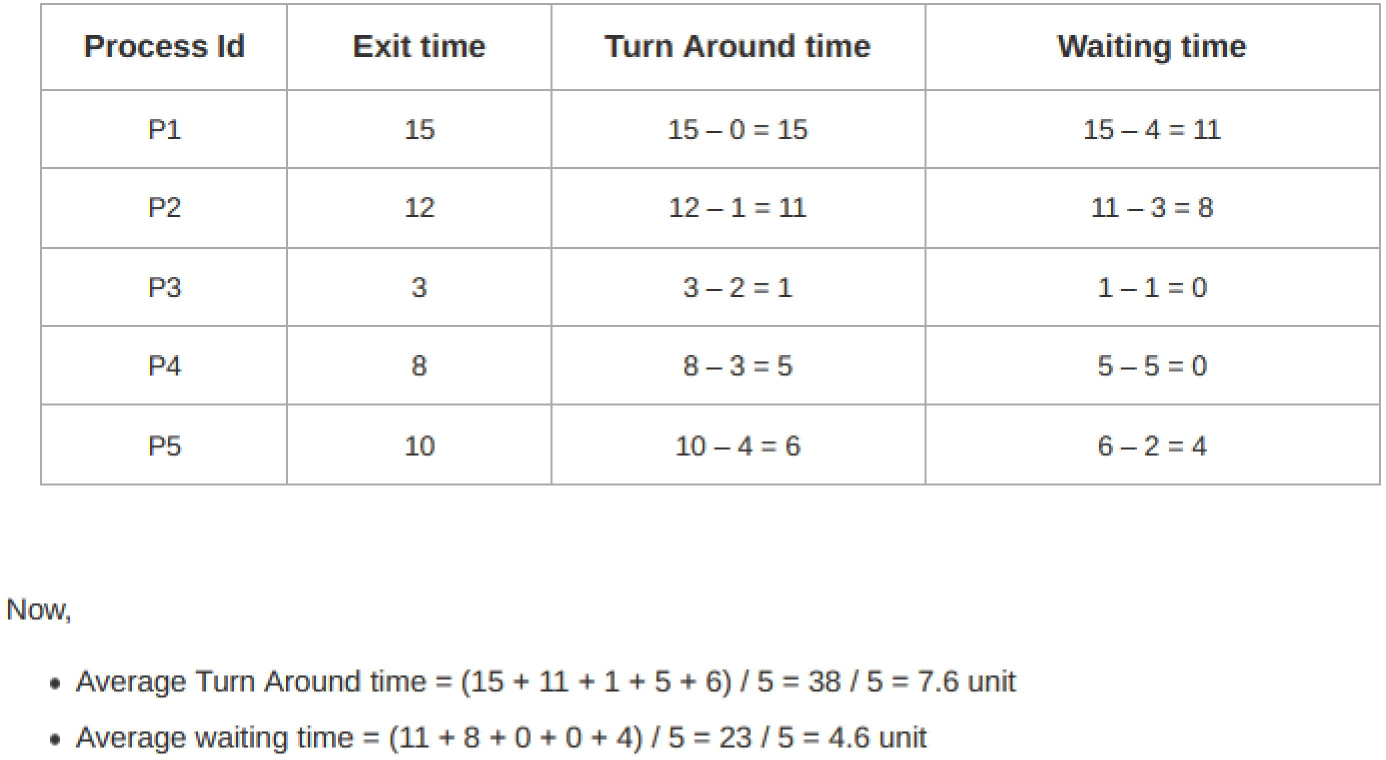
Priority Scheduling (preemptive)

Gantt Chart-



Gantt Chart

Priority Scheduling (preemptive)



Process Id	Exit time	Turn Around time	Waiting time
P1	15	15 – 0 = 15	15 – 4 = 11
P2	12	12 – 1 = 11	11 – 3 = 8
P3	3	3 – 2 = 1	1 – 1 = 0
P4	8	8 – 3 = 5	5 – 5 = 0
P5	10	10 – 4 = 6	6 – 2 = 4

Now,

- Average Turn Around time = (15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6 unit
- Average waiting time = (11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6 unit

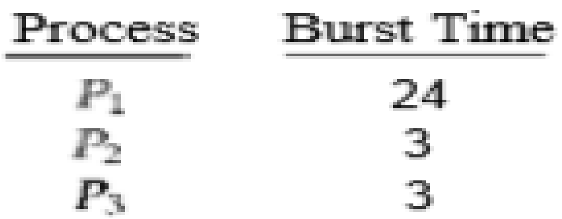
Priority Scheduling

- A major problem with priority scheduling algorithms is indefinite **blocking**, or **starvation**.
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes waiting indefinitely. In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.
- A solution to the problem of indefinite blockage of low-priority processes is **aging**. **Aging** is a technique of gradually increasing the priority of processes that wait in the system for a long time.

Round Robin(RR) Scheduling

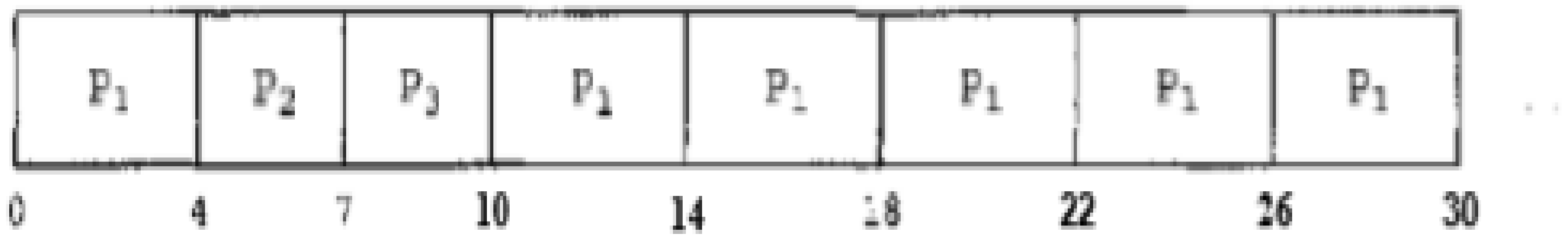
- The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.
- It is similar to **FCFS scheduling, but preemption** is added to switch between processes.
- A small unit of time, called a time **quantum** or **time slice**, is defined.
- A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
- To implement RR scheduling, we keep the ready queue as a FIFO queue of processes.
- New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

Round Robin(RR) Scheduling



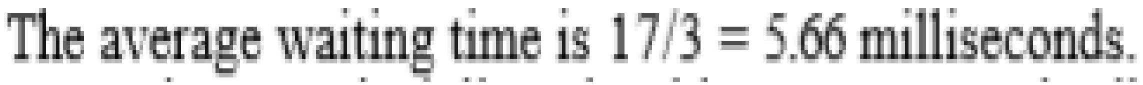
Process	Burst Time
P_1	24
P_2	3
P_3	3

Time quantum=4 ms



P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

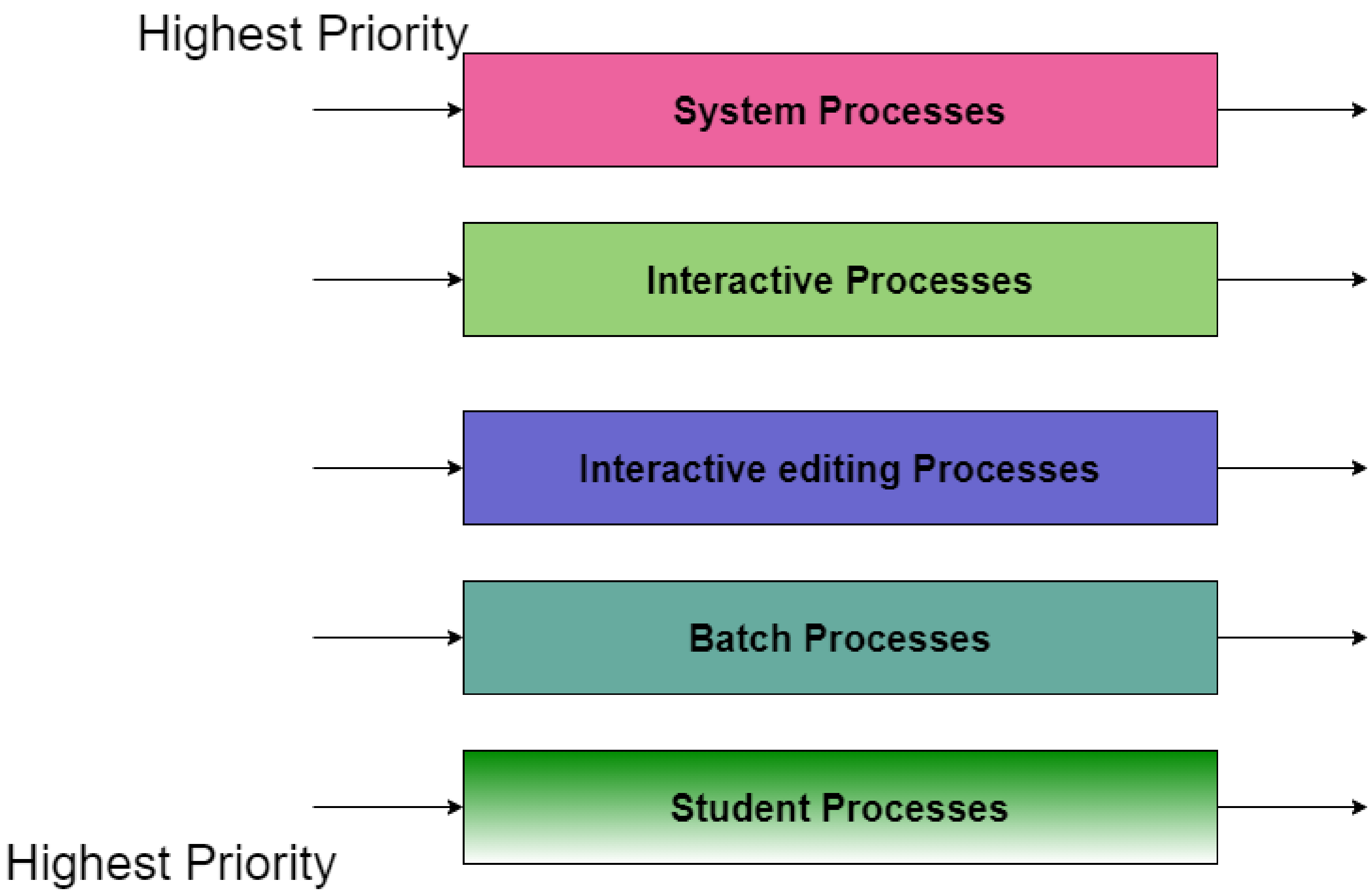
0 4 7 10 14 18 22 26 30



The average waiting time is 17/3 = 5.66 milliseconds.

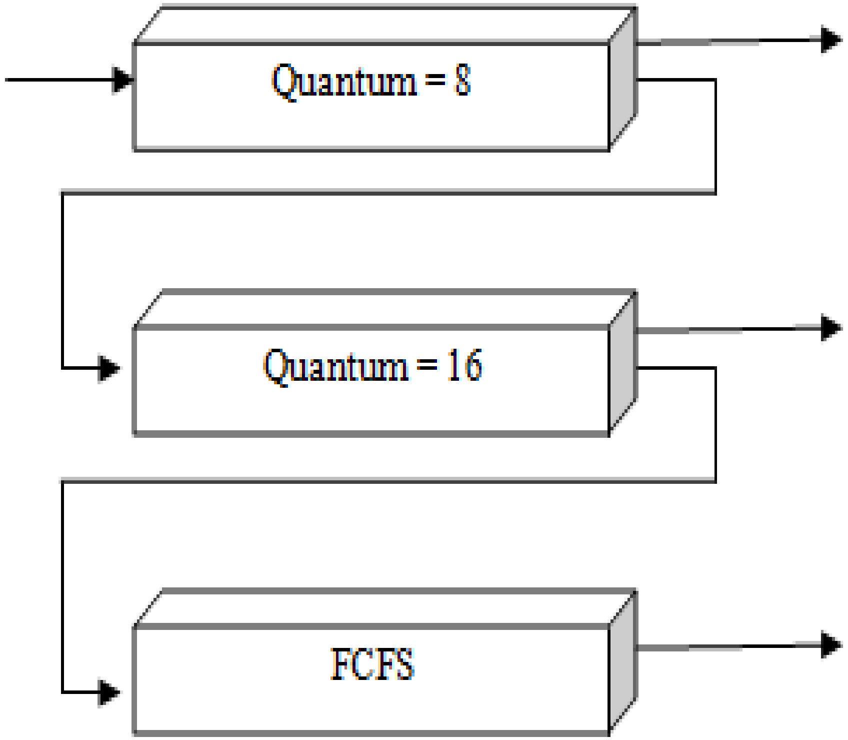
Multiple-Level (Multilevel)Queue Scheduling

- Multiple-level queues are a dependent scheduling algorithm.
- They allow the use of their existing algorithms to group and schedule jobs with common features.
- Multiple queues are supported for processes with common features. Each queue can obtain its own scheduling algorithms.
- Priorities are given to each queue.


Highest Priority
System Processes
Interactive Processes
Interactive editing Processes
Batch Processes
Student Processes
Highest Priority

Multilevel Feedback-Queue Scheduling

- This Scheduling is like Multilevel Queue(MLQ) Scheduling but in this process can move between the queues.
- **Multilevel Feedback Queue Scheduling (MLFQ)** keep analyzing the behavior (time of execution) of processes and according to which it changes its priority.
- Example: queue 1 and 2 follow round robin with time quantum 8 and 16 respectively and queue 3 follow FCFS



Multilevel Feedback-Queue Scheduling

One implementation of MFQS is given below –

1. When a process starts executing then it first enters queue 0.
2. In queue 0 process executes for 8 unit and if it completes in this 8 unit or it gives CPU for I/O operation in this 8 unit than the priority of this process does not change and if it again comes in the ready queue than it again starts its execution in Queue 0.
3. If a process in queue 0 does not complete in 8 unit then its priority gets reduced and it shifted to queue 1.
4. Above points 2 and 3 are also true for queue 1 processes but the time quantum is 16 unit. In a general case if a process does not complete in a time quantum than it is shifted to the lower priority queue.
5. In the last queue, processes are scheduled in FCFS manner.
6. A process in lower priority queue can only execute only when higher priority queues are empty.
7. A process running in the lower priority queue is interrupted by a process arriving in the higher priority queue.

Multiple Processor Scheduling and Benefits

- Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system.

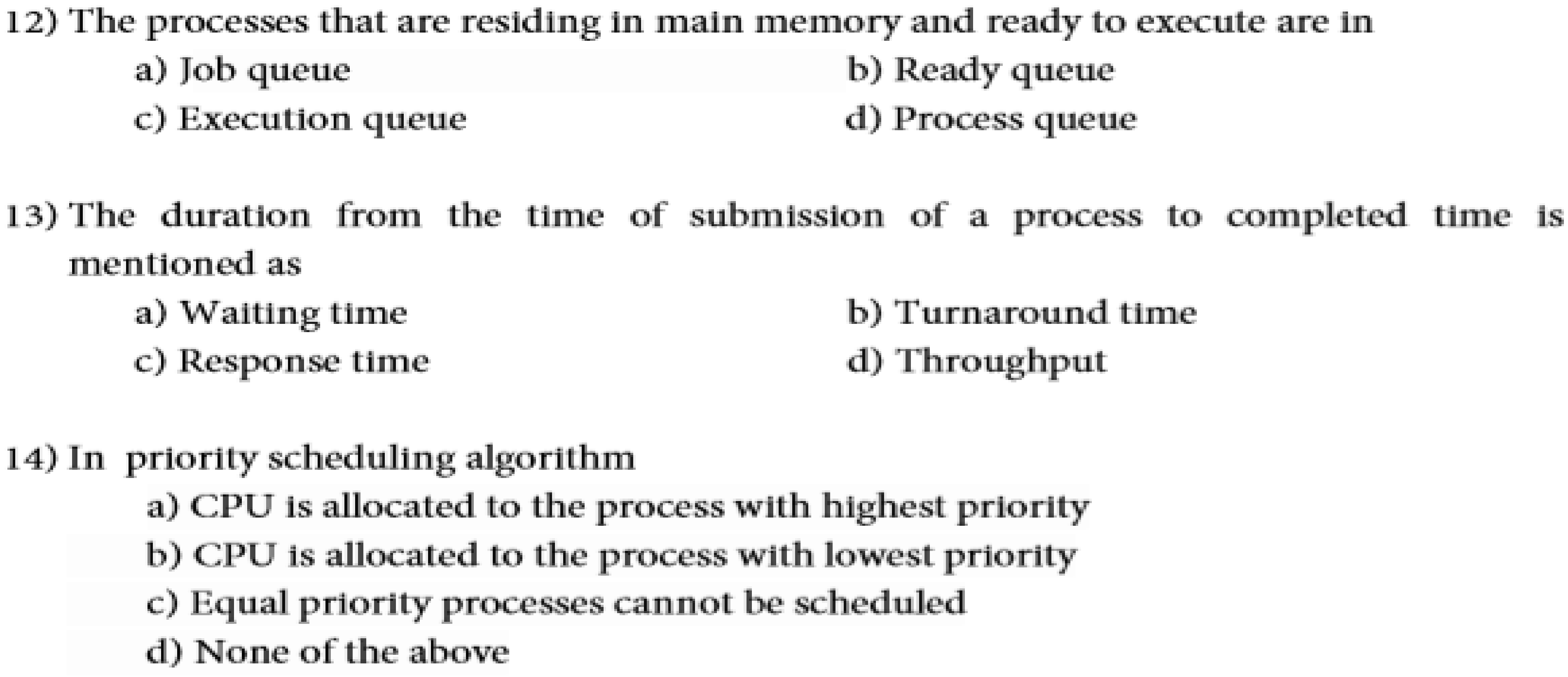
Benefits:

- **Increased Throughput**: More work can be completed in a unit time, by increasing the number of processors.
- **Cost Saving**: It shares the memory, buses, peripherals, etc. Therefore multiprocessor system saves money as compared to multiple single systems.
- **Increased Reliability:** In this system, due to the workload getting distributed among several processors results in increased reliability. Even if one processor fails then, its failure might slightly slow down the speed of the system, but the system will work smoothly

Real Time Scheduling

- The term scheduling analysis in real-time computing system comprises of the analysis and testing of the **scheduler system** and the algorithms that are used in the real-time applications.
- A real-time scheduling system is formed of the **scheduler, clock, and the processing hardware elements.**
- In a real-time system, a process or task has agenda; tasks are received by a real-time system and finished according to the task deadline depending on the nature of the scheduling algorithm.

Self-assessment Questions



12) The processes that are residing in main memory and ready to execute are in

a) Job queue b) Ready queue
c) Execution queue d) Process queue

13) The duration from the time of submission of a process to completed time is mentioned as

a) Waiting time b) Turnaround time
c) Response time d) Throughput

14) In priority scheduling algorithm

a) CPU is allocated to the process with highest priority
b) CPU is allocated to the process with lowest priority
c) Equal priority processes cannot be scheduled
d) None of the above