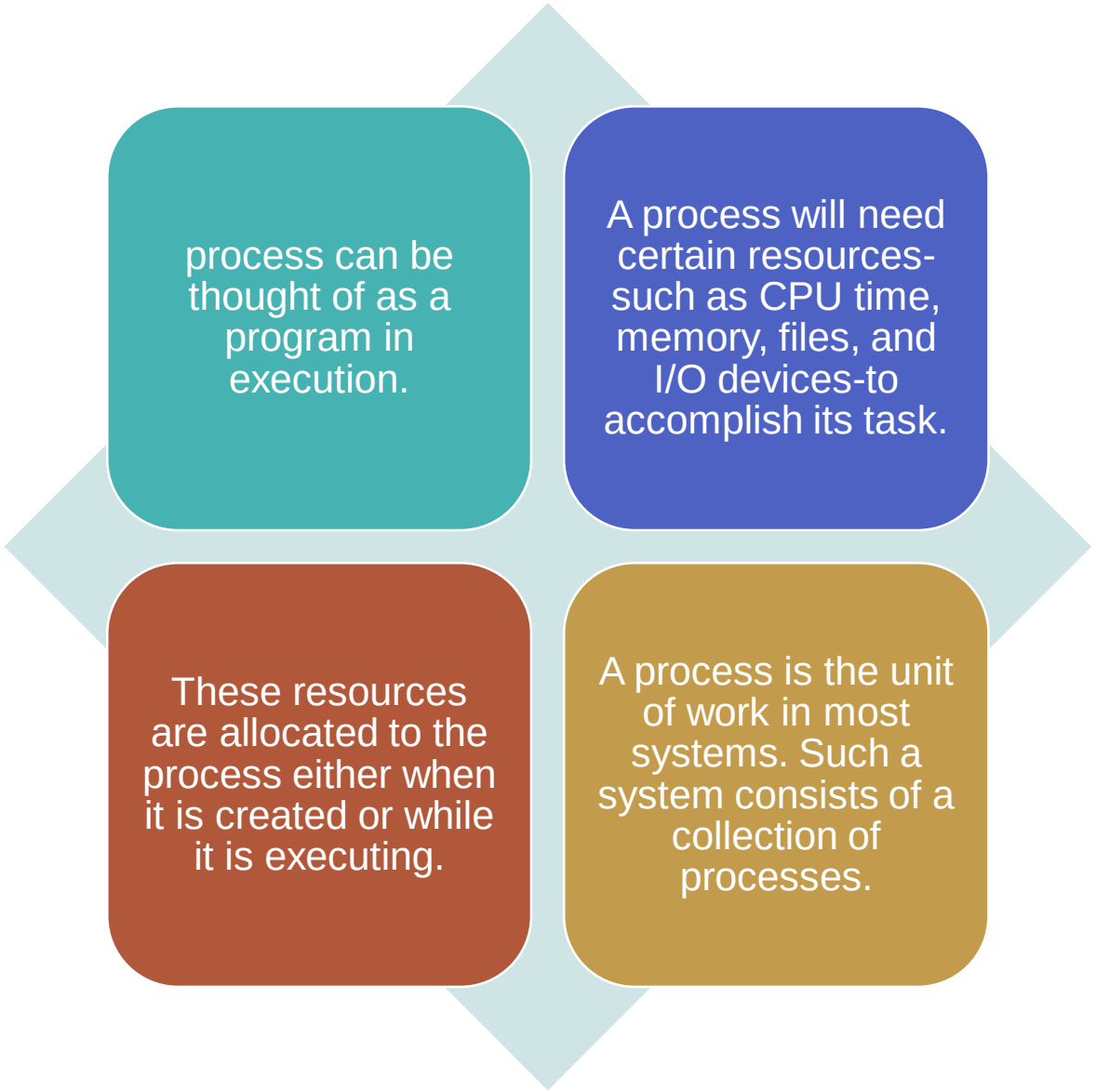
Process Management

Process Management



- A process is a program in execution.
- It includes:

✓The code being executed

✓The data and variables it uses

✓The resources it has (CPU, memory, files)

✓The state it's in (running, waiting, etc.)

Process state

- State of a process is defined in part by the current activity of that process. Each process may be in one of following states
- New: The process is being created.
- Ready: The process is waiting to be assigned to a processor.
- 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).
- Terminated: The process has finished execution.

Process state transition diagram

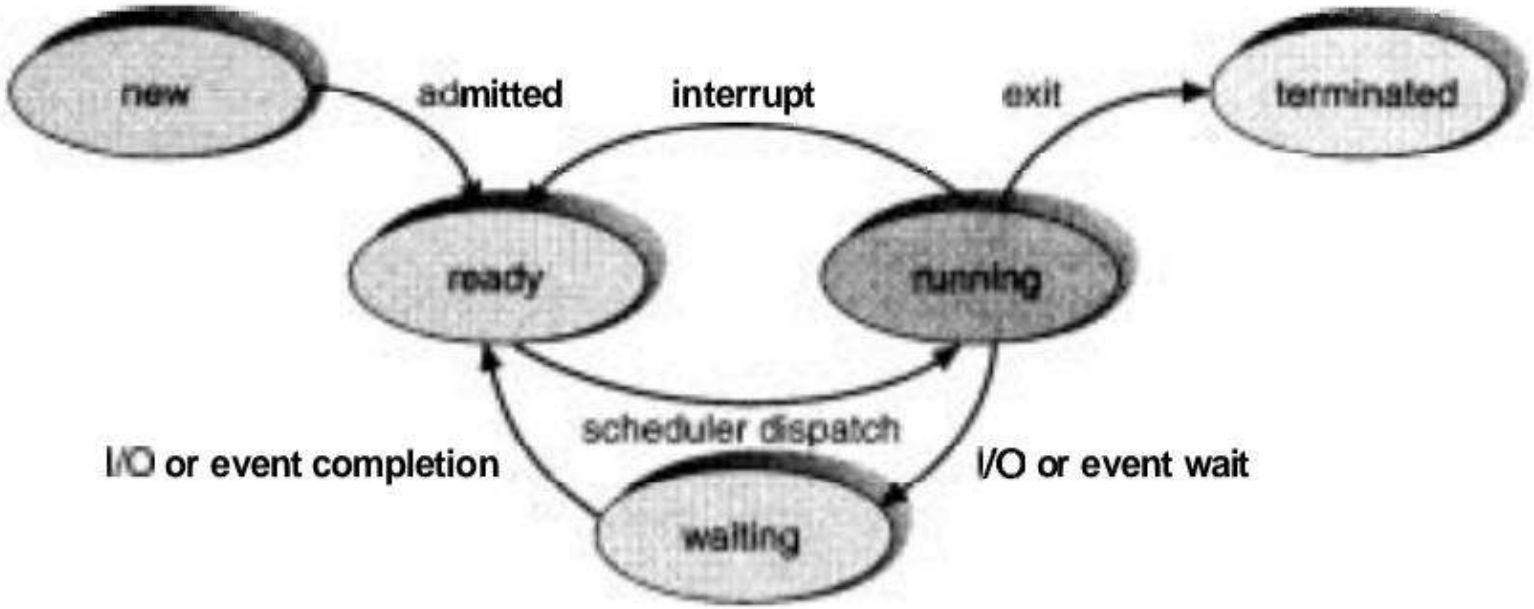


Figure 4.1 Diagram of process state.

- **New State**
- The new state is the initial state of a process.
- When a program in secondary memory is initiated for execution, the process is said to be in a new state.
- **Ready State**
- Once the process is loaded into the main memory and is prepared for execution, it transitions from the new state to the ready state.
- At this point, the process is waiting for the processor to execute it.
- In a multiprogramming environment, several processes may be in the ready state simultaneously.

- **Run State**
- Once the CPU is allocated for execution, the process transitions from the ready state to the run state.
- **Terminate State**
- Upon completion of a process's execution, it transitions from the run state to the terminate state.
- Process is killed as well as PCB(process control block) is deleted.
- The resources allocated to the process will be released or deallocated.

- **Block or Wait State**
- If a process requires an Input/Output operation or a blocked resource during execution, it transitions from the run state to the block or wait state.
- The process continues to wait in the main memory and does not require CPU.
- Once the I/O operation is completed or the resource becomes available, the process transitions to the ready state.

Process state transition diagram

- Only one process can be running on any processor at any instant, although many processes may be ready and waiting.

Process Control Block

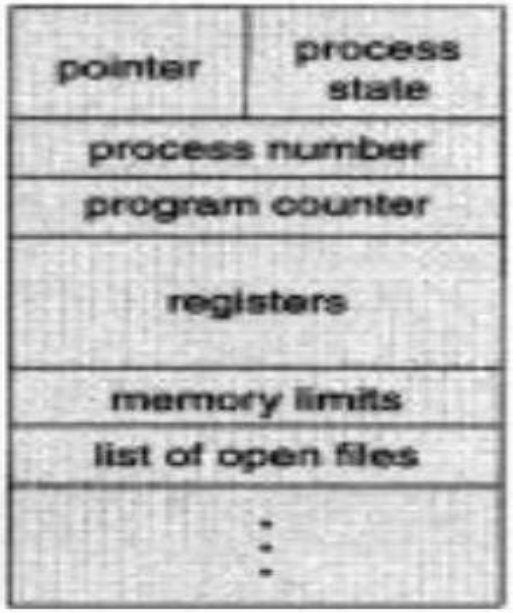


Figure 4.2 Process control block (PCB).

- Process ID (PID): A unique identifier for the process.
- Process State: Indicates the current status of the process (e.g., running, ready, waiting).
- Program Counter: Points to the next instruction to be executed in the process.
- CPU Registers: Stores the values of CPU registers used by the process during execution.
- CPU Scheduling Information: Includes priority, scheduling parameters, and other data needed for scheduling decisions.
- Memory Management Information: Contains details about the memory allocated to the process.
- Accounting Information: Tracks resource usage like CPU time, memory usage, and I/O operations.
- I/O Status Information: Keeps track of the I/O devices allocated to the process.
- Open Files: A list of files currently opened by the process.
- Pointers: Links to other PCBs, especially in the context of parent-child relationships or scheduling queues.

CPU registers

- Registers are the fastest memory in a computer system.They allow the CPU to access data quickly without needing to fetch it from slower memory like RAM.

Register Type	Function
Accumulator (ACC)	Stores results of arithmetic and logic operations.
Program Counter (PC)	Holds the address of the next instruction.
Instruction Register (IR)	Holds the current instruction being executed.
Memory Address Register (MAR)	Holds the address in memory to read/write data.
Memory Data Register (MDR)	Holds data being transferred to/from memory.
General Purpose Registers (e.g., AX, BX, R1, R2)	Temporarily store data, variables, or results.
Stack Pointer (SP)	Points to the top of the stack in memory.

Process Control Block

- **Process state:** The state may be new, ready, running, waiting, halted, and so on.
- **Program counter:** The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers:** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information.

Process Control Block

- **CPU-scheduling information:** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information**: This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system

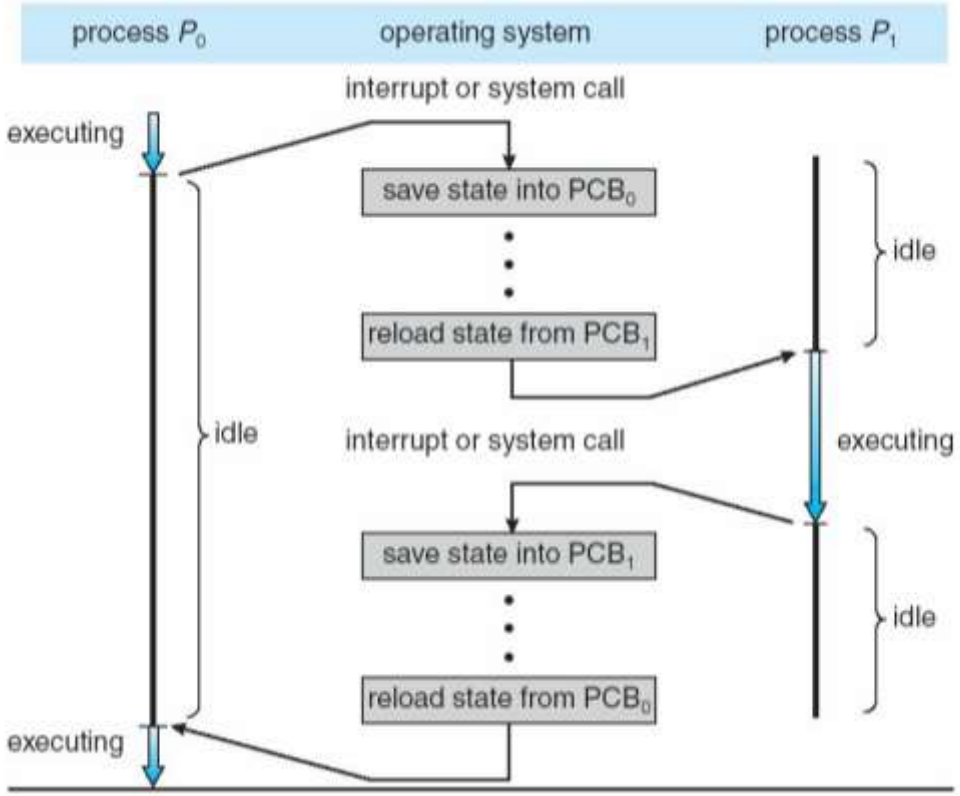
Process Control Block

- **Accounting information:** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I /O status information:** The information includes the list of I/O devices allocated to this process, a list of open files, and so on.

Context Switching

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- The more complex the OS and the PCB, the longer the context switch time dependent on hardware support
- Some hardware provides multiple sets of registers per CPU ie; multiple contexts loaded at once (no. of registers)

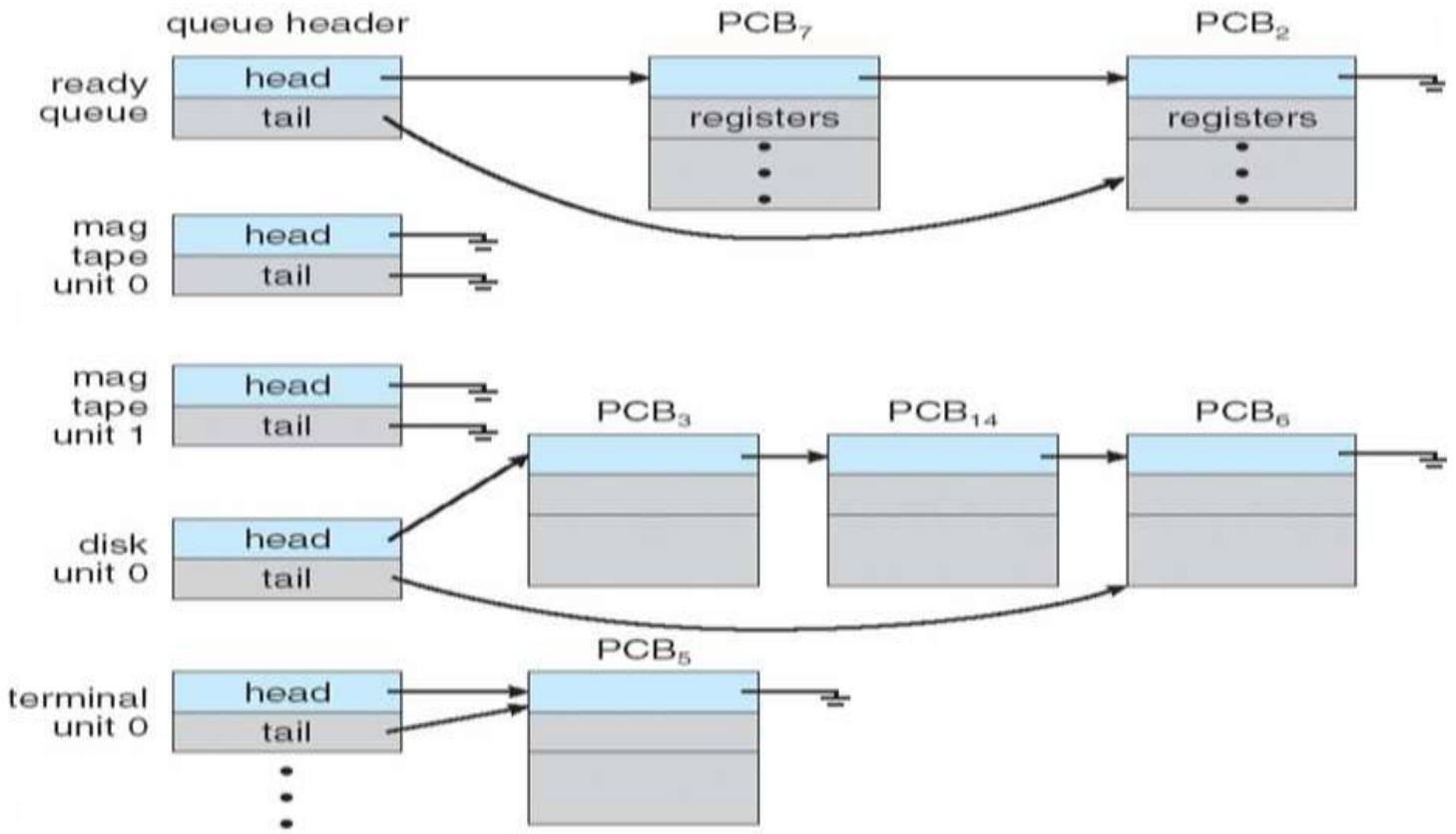
Context Switching



Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- Process scheduler selects among available processes for next execution on CPU
- Maintains scheduling queues of processes
- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device processes migrate among the various queues

Process Scheduling



Process Scheduling

- A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution (or dispatched).
- Once the process is assigned to the CPU and is executing, one of several events could occur:
- The process could issue an I/O request, and then be placed in an I/O queue.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

Process Scheduling

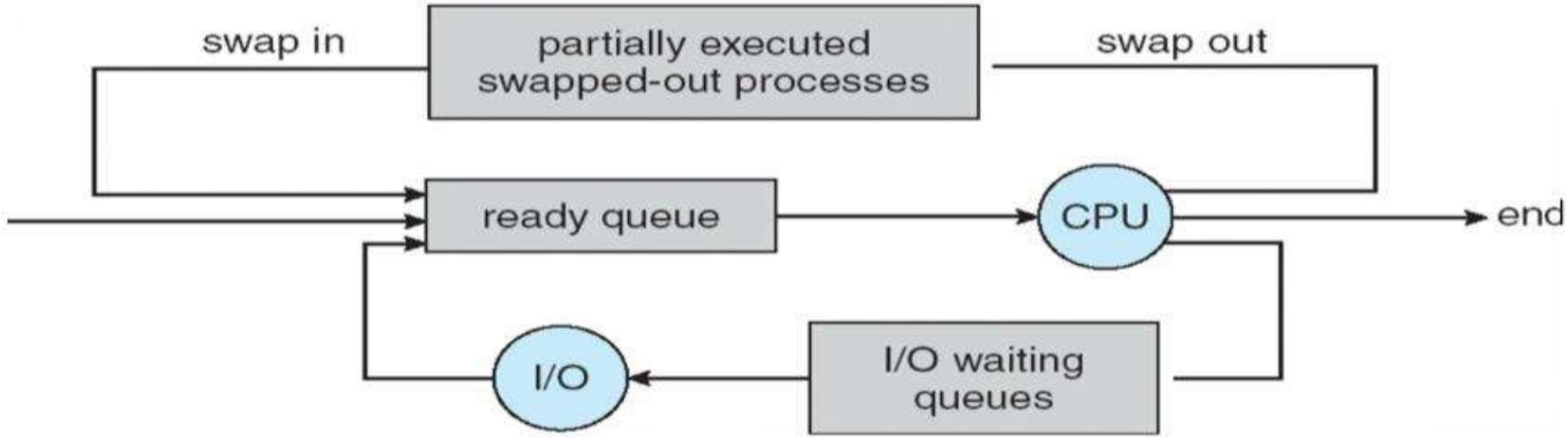
- In first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue.
- A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated

Schedulers

- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the degree of multiprogramming
- Processes can be described as either:
  - I/O-bound process – spends more time doing I/O than computations, many short CPU bursts
  - CPU-bound process – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good *process mix*

Schedulers

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



CPU Scheduling

- CPU scheduling is the basis of multiprogramming operating systems.
- By switching the CPU among processes, the operating system can make the computer more productive.

CPU Scheduler

- Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed.
- The selection process is carried out by the short-term scheduler (or CPU scheduler).
- The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them

Dispatcher

- Dispatcher is a special program which comes into play after the scheduler.
- When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue.

Functions of Dispatcher

- Function involves:
- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

- Imagine you're using your phone and open Spotify. You select a song, adjust volume, scroll playlists — all of this is done in user mode.
- You’re using the Spotify app, and you request to play a song. The kernel handles reading the song file from disk and sending audio to your speakers. You (user mode) can request, but only the kernel mode can execute that hardware-level task.

Scheduling Criteria

- Many criteria have been suggested for comparing CPU-scheduling algorithms. Mainly used for determining the best algorithm.
- 1) **CPU utilization:** We want to keep the CPU as busy as possible. CPU utilization may range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

Scheduling Criteria

- **Throughput**: One measure of work is the number of processes completed per time unit, called throughput. For long processes, this rate may be 1 process per hour; for short transactions, throughput might be 10 processes per second.

Scheduling Criteria

- **Turnaround time:** The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

Scheduling Criteria

- **Waiting time:** Waiting time is the sum of the periods spent waiting in the ready queue.
- **Response time:** Time from the submission of a request until the first response is produced. This measure, called response time, is the amount of time it takes to start responding, but not the time that it takes to output that response.
- **maximize CPU utilization and throughput, and to minimize turnaround time, waiting time, and response time**

Metric	Definition	Formula
Waiting Time (WT)	Total time a process **spends in the ready queue** (not executing, just waiting).	WT = Turnaround Time - Burst Time
Turnaround Time (TAT)	Total time from **submission** of a process to its **completion**.	TAT = Completion Time - Arrival Time

Scheduling Algorithms

- CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.

First-Come, First-Served Scheduling

- With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue.

- To understand the First Come, First Served (FCFS) scheduling algorithm effectively, we'll use two examples -

- One where all processes arrive at the same time,
- Another where processes arrive at different times.
- We'll create Gantt charts for both scenarios and calculate the turnaround time and waiting time for each process.

- Scenario 1: Processes with Same Arrival Time
- Consider the following table of arrival time and burst time for three processes p1, p2 and p3

Process	Arrival Time	Burst Time
p1	0	5
p2	0	3
p3	0	8

- Step-by-Step Execution:
- P1 will start first and run for 5 units of time (from 0 to 5).
- P2 will start next and run for 3 units of time (from 5 to 8).
- P3 will run last, executing for 8 units (from 8 to 16).

01
Step

Process	Arrival Time	Burst Time
P_1	0 ms	5 ms
P_2	0 ms	3 ms
P_3	0 ms	8 ms

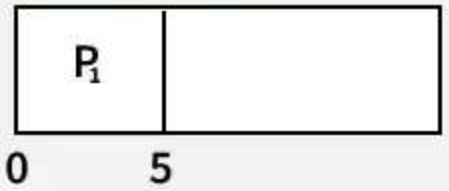
Ready Queue :
at t = 0

P_1	P_2	P_3

First Come First Serve with same arrival time

02
Step

Gantt chart at t = 5

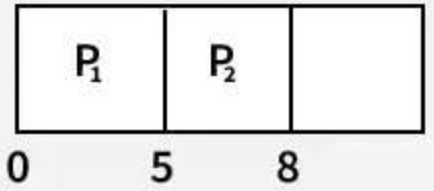


Ready Queue :
at t = 5

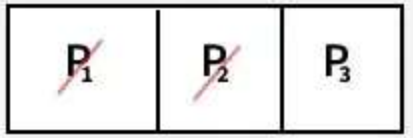
First Come First Serve with same arrival time

03
Step

Gantt chart at t = 8



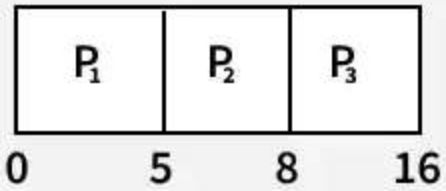
Ready Queue :
at t = 8



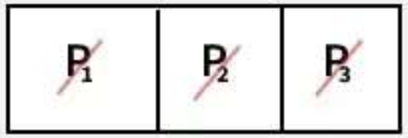
First Come First Serve with same arrival time

04
Step

Gantt chart at t = 16



Ready Queue :
at t = 16



First Come First Serve with same arrival time

- Scenario 2: Processes with Different Arrival Times
- Consider the following table of arrival time and burst time for three processes P1, P2 and P3

Process	Arrival Time (AT)	Burst Time (BT)
P1	2 ms	5 ms
P2	0 ms	3 ms
P3	4 ms	4 ms

- Step-by-Step Execution:
- P2 arrives at time 0 and runs for 3 units, so its completion time is:
- Completion Time of P2=0+3=3
- P1 arrives at time 2 but has to wait for P2 to finish. P1 starts at time 3 and runs for 5 units. Its completion time is:
- Completion Time of P1=3+5=8
- P3 arrives at time 4 but has to wait for P1 to finish. P3 starts at time 8 and runs for 4 units. Its completion time is:
- Completion Time of P3=8+4=12

01 Step

at t = 0

Process	Arrival Time	Burst Time
P_1	2 ms	5 ms
P_2	0 ms	3 ms
P_3	4 ms	4 ms

Ready Queue :

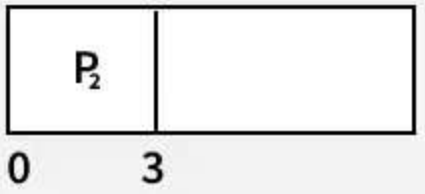
P_2	

First Come First Serve with different Arrival Time

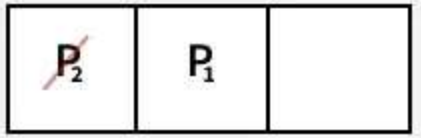
02 Step

at t = 3

Gantt chart :



Ready Queue :

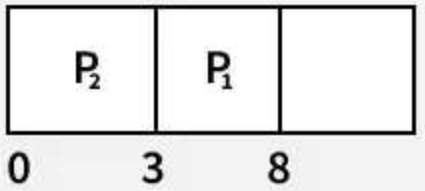


First Come First Serve with different Arrival Time

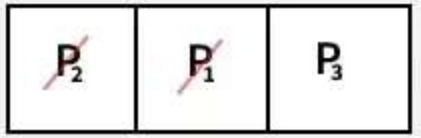
03 Step

at t = 8

Gantt chart :



Ready Queue :

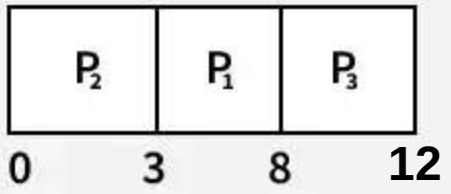


First Come First Serve with different Arrival Time

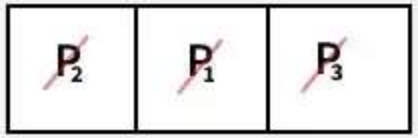
04 Step

at t = 12

Gantt chart :



Ready Queue :



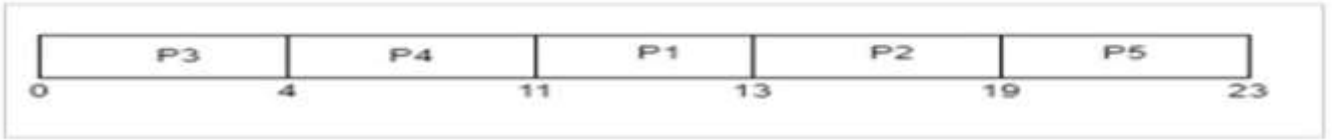
First Come First Serve with different Arrival Time

Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	2	2
P2	5	6
P3	0	4
P4	0	7
P5	7	4

Solution

Gantt chart



For this problem CT, TAT, WT is shown in the given table –

Process ID	Arrival time	Burst time	CT	TAT=CT-AT	WT=TAT – BT
P1	2	2	13	13-2= 11	11-2= 9
P2	5	6	19	19-5= 14	14-6= 8
P3	0	4	4	4-0= 4	4-4= 0
P4	0	7	11	11-0= 11	11-7= 4
P5	7	4	23	23-7= 16	16-4=12

Average Waiting time = (9+8+0+4+12)/5 = 33/5 = 6.6 time unit (time unit can be considered as milliseconds)

Average Turn-around time = (11+14+4+11+16)/5 = 56/5 = 11.2 time unit (time unit can be considered as milliseconds)

Consider the following set of processes that arrive at time 0, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

First-Come, First-Served Scheduling

Pre-emptive and Non pre-emptive scheduling

- CPU scheduling decisions may take place under the following four circumstances:

1.When a process switches from the running state to the waiting state (for example, I/O request)

2.When a process switches from the running state to the ready state (for example, when an interrupt occurs)

3. When a process switches from the waiting state to the ready state (for example, completion of I/O)

4. When a process terminates

Preemptive and Non preemptive scheduling

- When scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is non pre-emptive; otherwise, the scheduling scheme is pre-emptive.
- Under non pre-emptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Shortest job first scheduling- Non pre-emptive

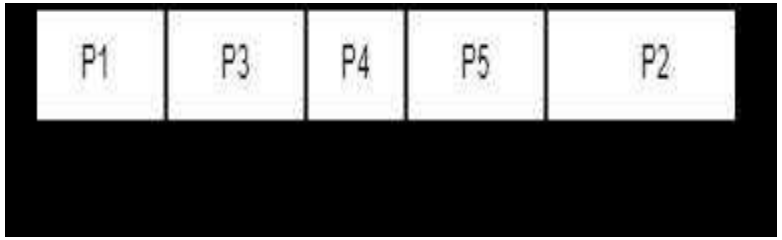
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie. Note that a more appropriate term would be the shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of a process, rather than its total length

Shortest job first scheduling- Non preemptive

Given five processes with process id/ no. as P1, P2, P3, P4, P5 along with their respective arrival time (A.T.) and burst time (B.T.). We need to draw the Gantt chart and find the completion time for each process. The time is denoted in milliseconds(ms).

Process id/ no.	Arrival Time(A.T.)	Burst Time(B.T.)
P1	0 ms	2 ms
P2	1 ms	5 ms
P3	2 ms	3 ms
P4	3 ms	1 ms
P5	5 ms	4 ms

Shortest job first scheduling- Non premptive



Process id/ no.	Arrival Time(A.T.)	Burst Time(B.T.)
P1	0 ms	2 ms
P2	1 ms	5 ms
P3	2 ms	3 ms
P4	3 ms	1 ms
P5	5 ms	4 ms

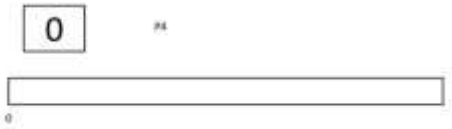
B.T.)	Completion time(C.T)	Turn around time(T.A.T) (C.T. - A.T.)	Waiting time(W.T.) (T.A.T. - B.T.)
	2ms	2-0= 2ms	2-2=0ms
	15ms	15-1= 14ms	14-5= 9ms
	5ms	5-2= 3ms	3-3 = 0ms
	6ms	6-3= 3ms	3-1= 2ms
	10ms	10-5= 5ms	5-4= 1ms

Shortest job first scheduling-preemptive

Consider the following five process:

Process Queue	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

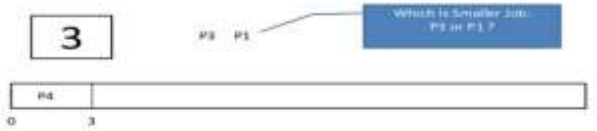
Shortest job first scheduling- preemptive



Step 1) At time= 1, Process P3 arrives. But, P4 has a shorter burst time. It will continue execution.

Step 2) At time = 2, process P1 arrives with burst time = 6. The burst time is more than that of P4. Hence, P4 will continue execution.

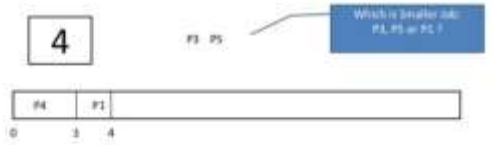
Step 3) At time = 3, process P4 will finish its execution. The burst time of P3 and P1 is compared. Process P1 is executed because its burst time is lower.



Step 4) At time = 4, process P5 will arrive. The burst time of P3, P5, and P1 is compared. Process P5 is executed because its burst time is lowest. Process P1 is preempted.

Process Queue	Burst time	Arrival time
P1	5 out of 6 is remaining	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

Shortest job first scheduling- premptive



Step 5) At time = 5, process P2 will arrive. The burst time of P1, P2, P3, and P5 is compared. Process P2 is executed because its burst time is least. Process P5 is preempted.

Process Queue	Burst time	Arrival time
P1	5 out of 6 is remaining	2
P2	2	5
P3	8	1
P4	3	0
P5	3 out of 4 is remaining	4

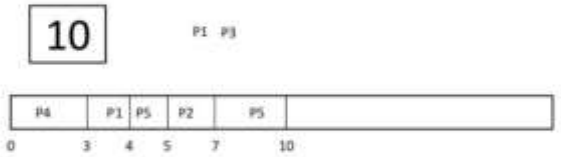
Step 6) At time =6, P2 is executing.

Step 7) At time =7, P2 finishes its execution. The burst time of P1, P3, and P5 is compared. Process P5 is executed because its burst time is lesser.

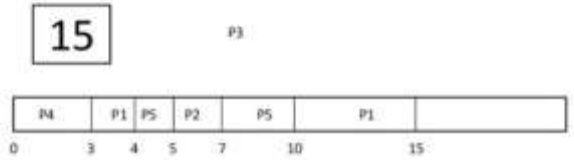
Process Queue	Burst time	Arrival time
P1	5 out of 6 is remaining	2
P2	2	5
P3	8	1
P4	3	0
P5	3 out of 4 is remaining	4

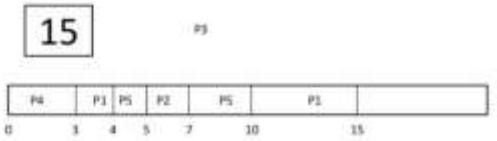
Shortest job first scheduling- pre-emptive

Step 8) At time =10, P5 will finish its execution. The burst time of P1 and P3 is compared. Process P1 is executed because its burst time is less.



Step 9) At time =15, P1 finishes its execution. P3 is the only process left. It will start execution.





Step 10) At time =23, P3 finishes its execution.

Step 11) Let's calculate the average waiting time for above example.

```
Wait time
P4= 0-0=0
P1=  (3-2) + 6 =7
P2= 5-5 = 0
P5= 4-4+2 =2
P3= 15-1 = 14
```

```
Average Waiting Time =
0+7+0+2+14/5 = 23/5 =4.6
```

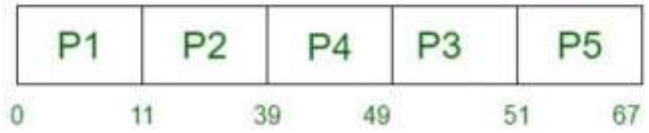
Priority scheduling

- In Priority scheduling, there is a priority number assigned to each process.
- In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority.
- The Process with the higher priority among the available processes is given the CPU

Priority scheduling-Nonpreemptive

Process	Arrival Time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

Gantt Chart –



Priority scheduling-Nonpreemptive

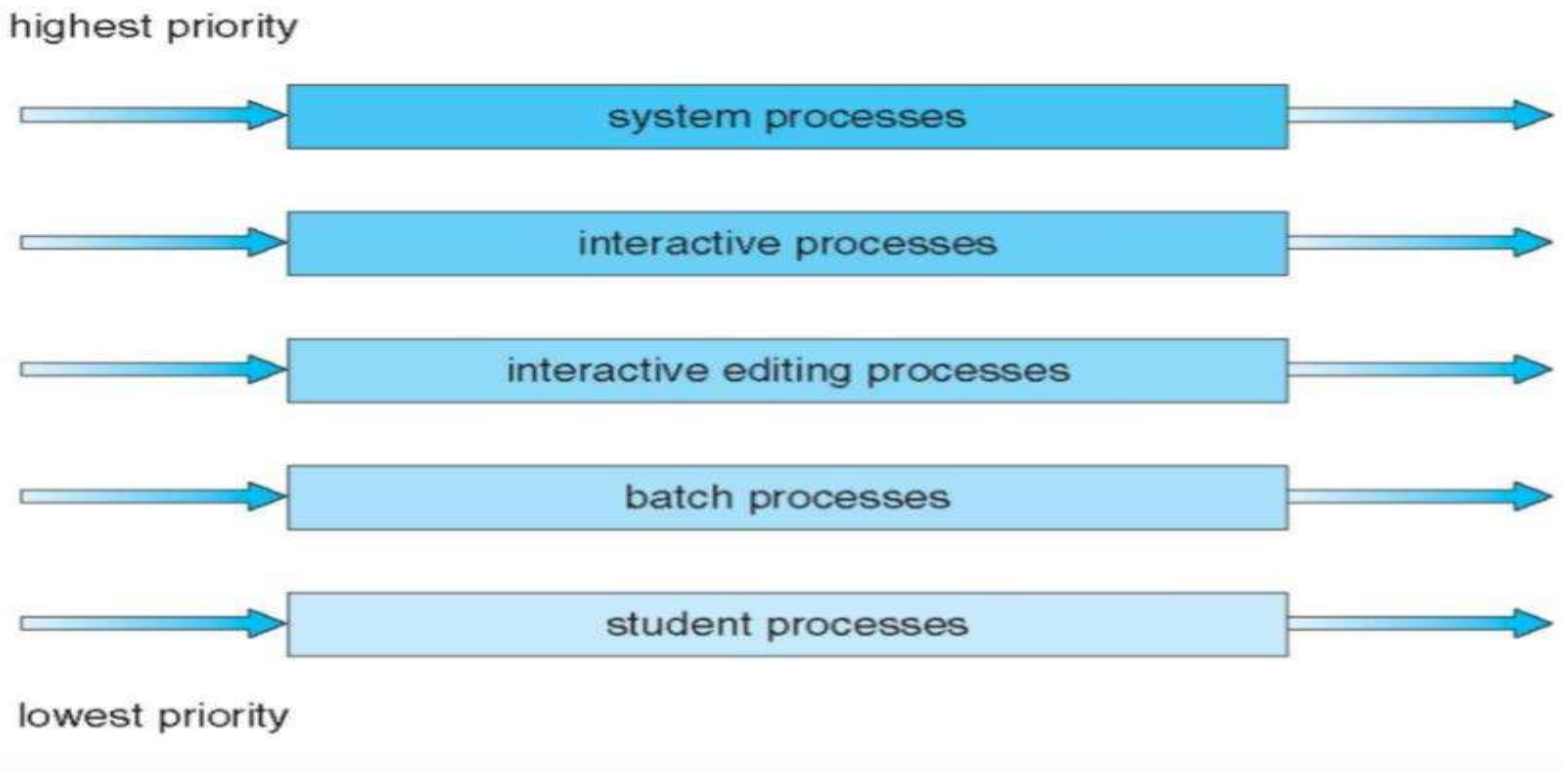
```
Input :
process no-> 1 2 3 4 5
arrival time-> 0 1 3 2 4
burst time-> 3 6 1 2 4
priority-> 3 4 9 7 8
Output :
Process_no    arrival_time    Burst_time    Com
1              0               3
2              1               6
3              3               1
4              2               2
5              4               4
Average Waiting Time is : 5.6
Average Turn Around time is : 8.8
```

Complete_time	Turn_Around_Time	Wait
3	3	0
9	8	2
16	13	12
11	9	7
15	11	7

Multilevel Queue Scheduling

- multilevel queue-scheduling algorithm partitions the ready queue into several separate queues .
- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

Multilevel Queue Scheduling



Multilevel Queue Scheduling

- there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling.
- Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty

Multilevel Feedback Queue Scheduling

- Multilevel feedback queue scheduling, however, allows a process to move between queues.
- The idea is to separate processes with different CPU-burst characteristics.

Multilevel Feedback Queue Scheduling

Multilevel Feedback Queue Scheduling

- a multilevel feedback queue scheduler with three queues, numbered from 0 to 2 .
- The scheduler first executes all processes in queue 0. Only when queue 0 is empty will it execute processes in queue 1. Similarly, processes in queue 2 will be executed only if queues 0 and 1 are empty

Multilevel Feedback Queue Scheduling

- A process entering the ready queue is put in queue 0. A process in queue 0 is given a time quantum of 8milliseconds. If it does not finish within this time, it is moved to the tail of queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2.
- Processes in queue 2 are run on an FCFS basis, only when queues 0 and 1 are empty.

Multilevel Feedback Queue Scheduling

- A multilevel feedback queue scheduler is defined by the following parameters:
- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher priority queue
- The method used to determine when to demote a process to a lower-priority queue
- The method used to determine which queue a process will enter when that process needs service

Interprocess communication(IPC)

- Process can be of two types:
- Independent process.
- Co-operating process
- independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes.

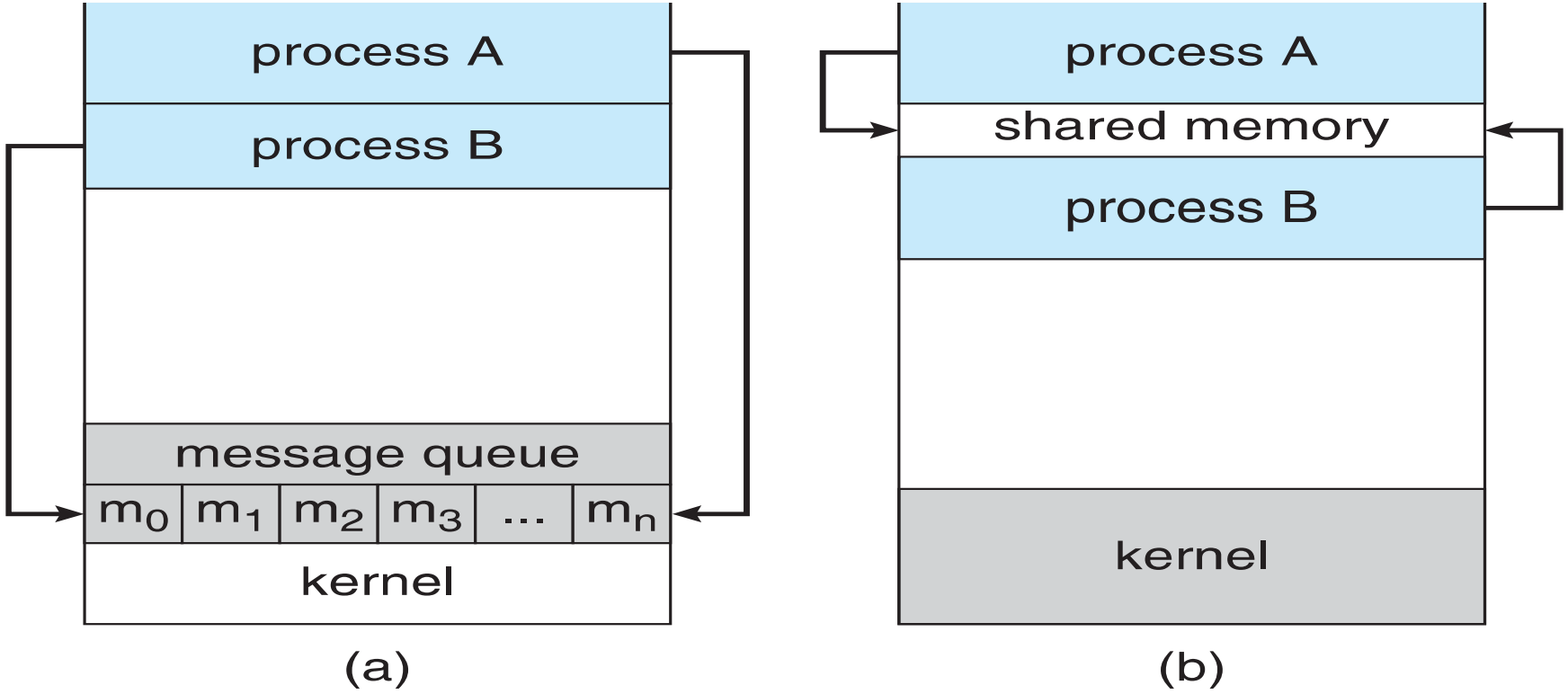
Interprocess communication

- Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.
- The communication between these processes can be seen as a method of co-operation between them.

Interprocess communication

- IPC is particularly useful in a distributed environment where the communicating processes may reside on different computers connected with a network.
- An example is a chat program used on the World Wide Web.
- 2 models for interprocess communication
- Shared Memory
- Message passing

Interprocess communication



Message-Passing System

- Communication among the user processes is accomplished through the passing of messages.
- An IPC facility provides at least the two operations:
  1)**send(message)**
  2) **receive(message)**
- processes P and Q want to communicate, they must send messages to and receive messages from each other; a communication link must exist between them

Message-Passing System

- Processes that want to communicate must have a way to refer to each other. They can use either direct or indirect communication.
- **Direct Communication**
- With direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication.
- In this scheme, the send and receive primitives are defined as:

Message-Passing System

- **send(P, message)** –send a message to process P
- **receive(Q, message)** –receive a message from process Q
- Communication link in this scheme has the following properties:
- A link is established automatically between every pair of processes that want to communicate.

Message-Passing System

- **Indirect Communication**
- With indirect communication, the messages are sent to and received from mailboxes, or ports which messages can be placed by processes and from which messages can be removed.
- Each mailbox has a unique identification. In this scheme, a process can communicate with some other process via a number of different mailboxes.

Message-Passing System

- Two processes can communicate only if they share a mailbox. The send and receive primitives are defined as follows:
- **send (A, message)** -Send a message to mailbox A.
- **receive (A, message)** -Receive a message from mailbox A
- In this scheme, a **communication link** has the following properties:
- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.

Synchronization

- Message passing may be either blocking or nonblocking-also known as synchronous and asynchronous.
- **Blocking send:** The sending process is blocked until the message is received by the receiving process or by the mailbox.
- **Nonblocking send:** The sending process sends the message and resumes operation.
- **Blocking receive:** The receiver blocks until a message is available.
- **Nonblocking receive:** The receiver retrieves either a valid message or a null.

Buffering

- Whether the communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue. Basically, such a queue can be implemented in three ways:
- **Zero capacity:** The queue has maximum length 0; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.

Buffering

- **Bounded capacity:** The queue has finite length n; thus, at most n messages can reside in it.
- **Unbounded capacity:** The queue has potentially infinite length; thus, any number of messages can wait in it. The sender never blocks
- The zero-capacity case is sometimes referred to as a message system with **no buffering**; the other cases are referred to as **automatic buffering.**

Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.

Producer-Consumer Problem

Bounded-Buffer – Producer

```
item next_produced;
while (true) {
        /* produce an item in next produced */
        while (((in + 1) % BUFFER_SIZE) == out)
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
}
```

Bounded Buffer – Consumer

```
item next_consumed;

while (true) {
     while (in == out)
          ; /* do nothing */
     next_consumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;

     /* consume the item in next consumed */
}
```