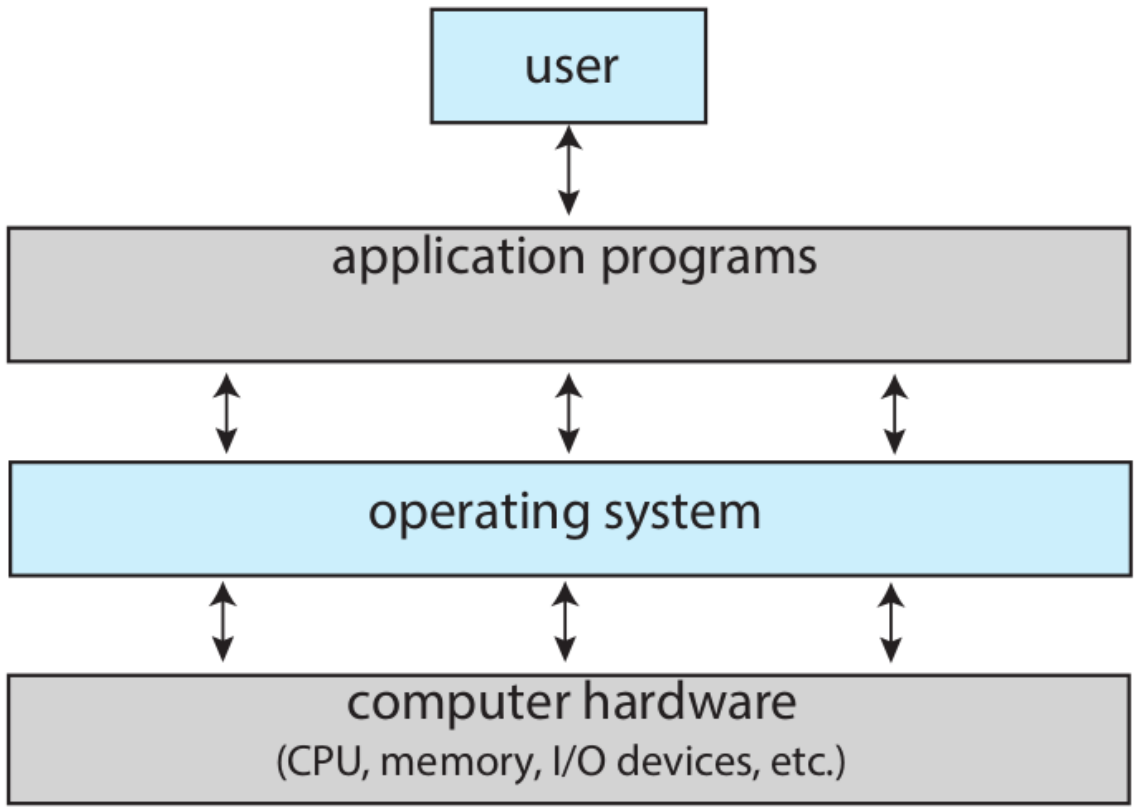
Bridge Course

Operating Systems

Vasudevan T V

Module 1
Overview

- An Operating System is a software that acts as an interface between the user of a computer and the computer hardware
- It provides an environment in which a user can execute programs in a convenient and efficient manner.



Functions of an Operating System

- Memory Management
- Process Management
- Device Management
- File System Management
- Security

Functions of an Operating System

- Memory Management
  - It refers to the management of main memory
  - To execute a program all (or part) of the instructions must be in memory
  - All (or part) of the data that is needed by the program must be in memory
- Memory Management Activities
  - Keeping track of which parts of memory are currently being used and by whom
  - In a multiprogramming environment, the OS decides which process will get memory when and how much
  - Allocating and deallocating memory space as needed

Functions of an Operating System

- Process Management
  - A process is a program in execution
- Process Management Activities
  - In multiprogramming environment, the OS decides which process gets the processor, when and for how much time. This is called process scheduling
  - Keeps track of the status of processor and processes.
  - Allocates the processor to a process
  - De-allocates processor when it is no longer required

Functions of an Operating System

- Device Management
  - It refers to the management of I/O devices
- Device Management Activities
  - Keeps tracks of all devices
  - Decides which process gets the device, when and for how much time
  - Allocates the device to a process
  - De-allocates the device when it is no longer required

Functions of an Operating System

- File System Management
  - Anything that is stored in the secondary storage is a file
  - The structure in which files are organised in a computer is called a file system
  - A directory is a location for storing files
- File System Management Activities
  - Creating and deleting files and directories
  - Gives the permission to the program for operation (read-only, read-write, denied) on file
  - Mapping files onto secondary storage
  - Backup files onto secondary storage

Functions of an Operating System

- Protection and Security
  - Protection is a mechanism for controlling access of processes or users to resources defined by the OS (Setting Passwords, Access Privileges)
  - Security is a mechanism for handling external threats to the system (Attack from hackers, viruses)

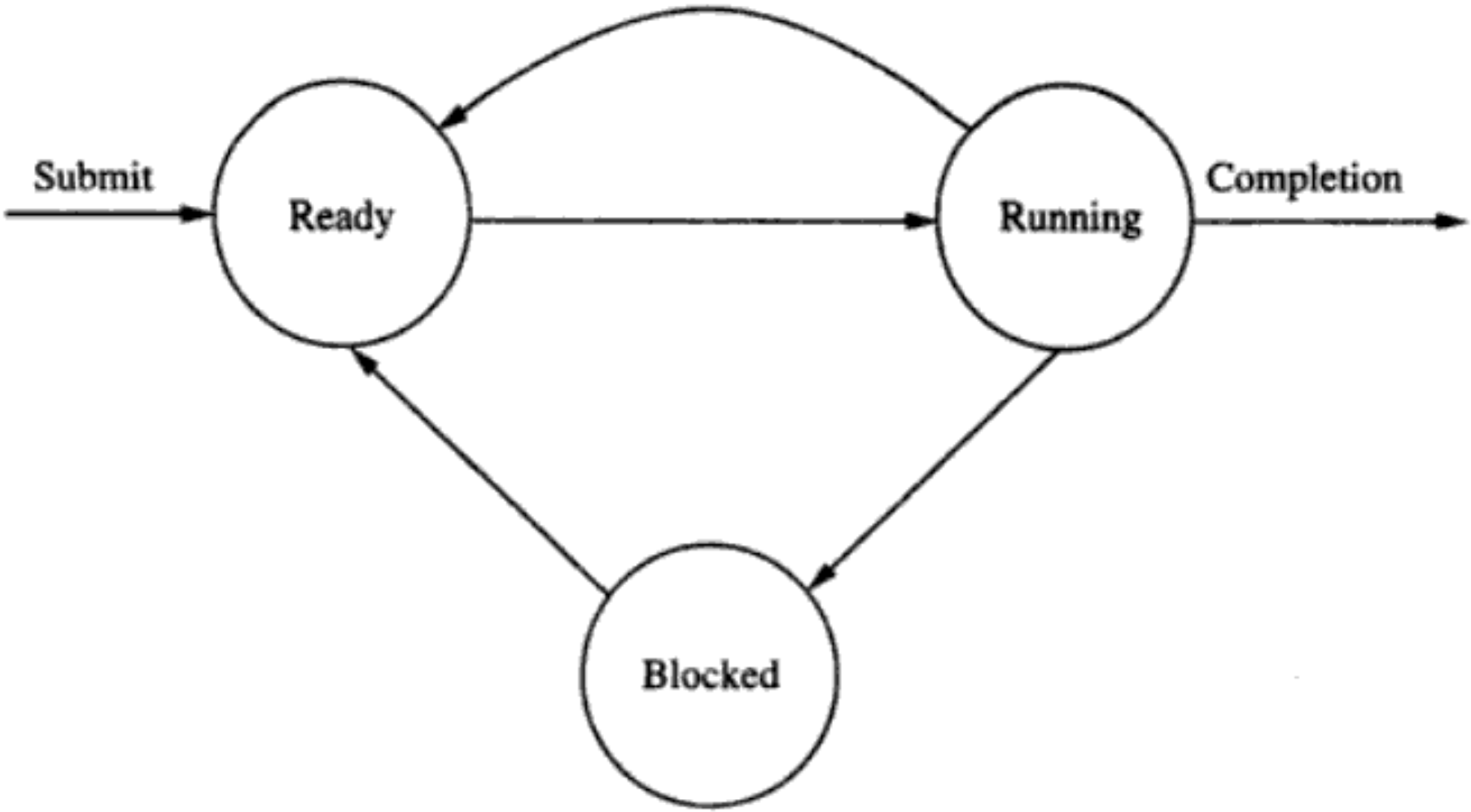
Concept of a Process

- A process is a program in execution
- A process can be in any of the three basic states

1. Running - Instructions are being executed
2. Ready - The process is waiting for the availability of the processor
3. Blocked (Waiting) - The process is waiting for some event to occur such as I/O completion, Memory Availability etc.

Concept of a Process

- State Transition Diagram of a Process



Operations on Processes

- There are mainly 2 operations on processes

1. process creation

- A process is created before the execution of its first statement

2. process termination

- A process is terminated after the execution of its final statement

Operations on Processes

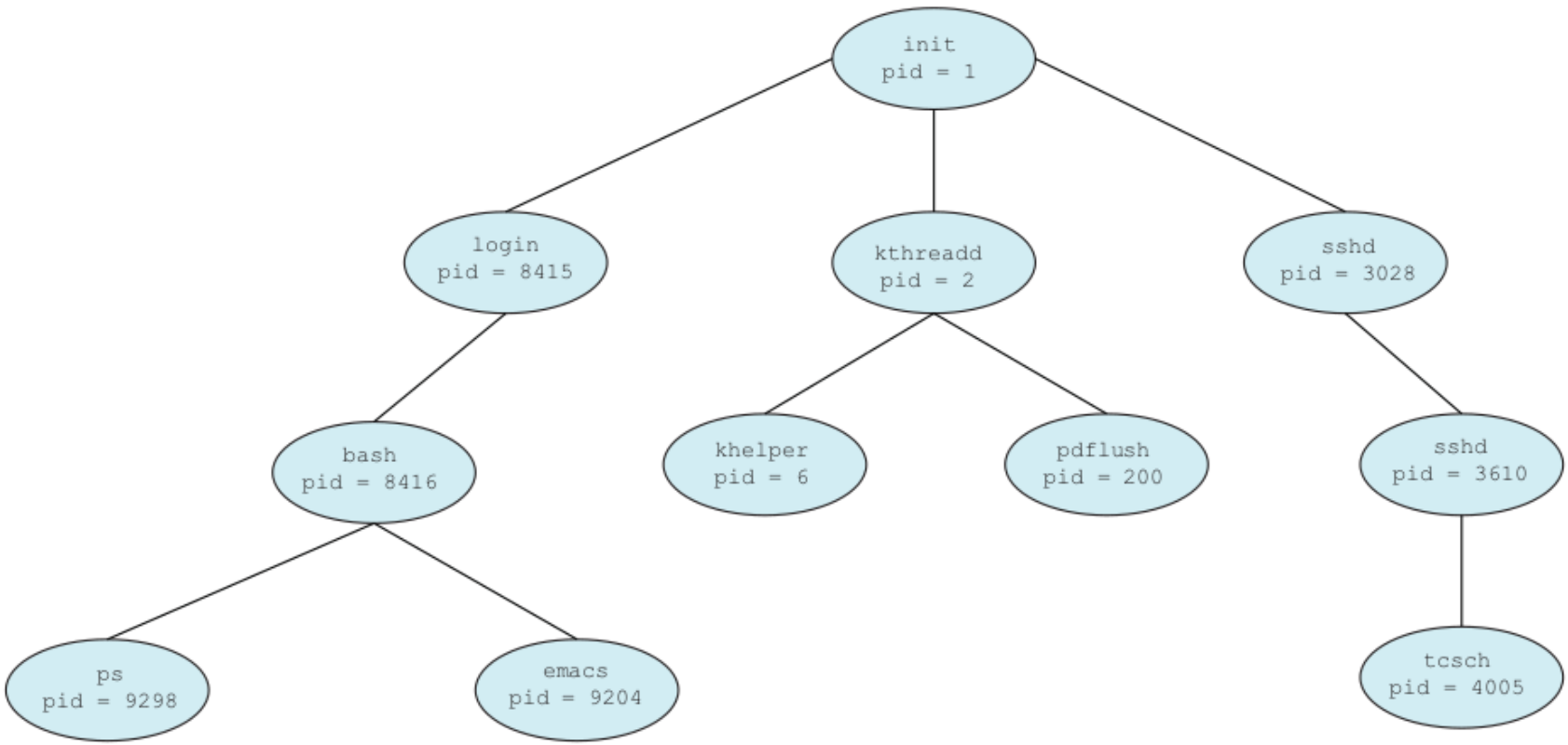
1. process creation

- A process is created before the execution of its first statement
- During the course of execution, a process may create several new processes
- The creating process is called a parent process
- The new processes are called the children of that process
- Each of these new processes may in turn create other processes, forming a tree of processes
- Associated with each process there is a unique process identifier (or pid), which is typically an integer number
- In the linux os, the first process created is the system process init
- The init process has a pid of 1
- The init process serves as the root parent process for all user processes

Operations on Processes

1. process creation

- A tree of processes in the linux system



Operations on Processes

1. process creation

- When a process creates a new process, two possibilities for execution exist

1. The parent continues to execute concurrently with its children
2. The parent waits until some or all of its children have terminated

- When a process creates a child process, that child process will need certain resources (CPU time, memory, files, I/O devices) to accomplish its task

1. A child process can obtain its resources directly from the operating system
2. A child process may be constrained to a subset of the resources of the parent process

- In UNIX OS, a new process is created by the fork() system call
- In Windows OS, a new process is created by CreateProcess() function

Operations on Processes

2. process termination

- A process is terminated after the execution of its final statement
- A process is terminated using the exit() system call
- All the resources of the process—including memory, files, and I/O devices—are deallocated by the operating system

Concurrent Processes

- Two processes can be serial or concurrent
- Two processes are called serial if the execution of one completes before the execution of the other begins
- Two processes are called concurrent if the execution of one overlaps with the other
- Processes executing concurrently in the operating system may be either independent processes or cooperating processes
- A process is independent if it cannot affect or be affected by the other processes executing in the system
- Any process that does not share data with any other process is independent

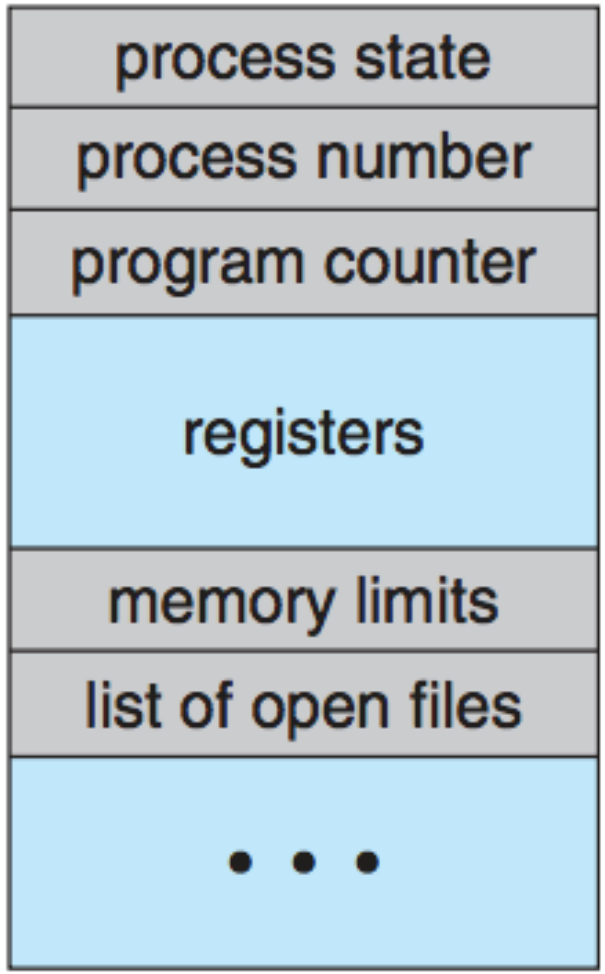
Concurrent Processes

- A process is cooperating if it can affect or be affected by the other processes executing in the system
- Clearly, any process that shares data with other processes is a cooperating process
- Concurrent processes interact with the other by using shared variables and message passing
- If concurrent processes do not interact with the other, then their execution is same as their serial execution

Process Control Block

- Each process is represented in the operating system by a process control block (PCB) (task control block)
- It contains several pieces of information associated with a process, including these
- process state
- process number
- program counter - This counter indicates the address of the next instruction to be executed for this process
- CPU Registers - high speed memory in the CPU for storing data
- memory limits - maximum memory allocated for the process
- list of open files

Process Control Block



Process Context

- When the CPU switches from one process to another, the current state of a process need to be saved, so that it can be retrieved later
- The current state of a process is called a process context
- This information is available in the PCB of a process
- It includes the process state, the value of the CPU registers, memory limits etc.
- The act of CPU switching from one process to another is called a context switch

Processor Scheduling

- In a multiprogramming environment, several processes are to be executed in the CPU
- As processes enter the system, they are put into a job queue, which consists of all processes in the system
- The processes that are ready and waiting to execute are kept on a list called the ready queue
- The processes waiting for a particular I/O device are put in a device queue
- The CPU scheduler selects from among the processes that are ready to execute and allocates the CPU to one of them

Processor Scheduling

- For CPU scheduling, various algorithms are used
- The choice of a particular algorithm may favour one class of processes over another
- The criteria used for judging the best algorithm to be used in a particular situation is given below

1. CPU Utilisation

- It is the fraction of time the CPU remains busy
- Conceptually, CPU utilisation can range from 0 to 100 percent
- In a real system, it ranges from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system)

2. Throughput

- The number of processes that are completed per time unit is called throughput
- For long processes, this rate may be one process per hour
- For short transactions, it may be ten processes per second

Processor Scheduling

3. Turnaround Time

- The interval from the time of submission of a process to the time of completion is the turnaround time

4. Waiting Time

- It is the sum of the periods spent waiting by a process in the ready queue

5. Response Time

- It is the time from the submission of a request until the first response is produced by a process

Scheduling Algorithms

- There are 2 kinds of scheduling algorithms - preemptive and non-preemptive
- A preemptive scheduling algorithm picks a process and lets it run for a maximum of some fixed time
- If it is still running at the end of the time interval, it is suspended and the scheduler picks another process to run (if one is available)
- A non preemptive scheduling algorithm picks a process to run and then just lets it run until it releases the CPU either by terminating or by requesting I/O

Scheduling Algorithms

1. First Come First Served (FCFS) Scheduling Algorithm

- It is the simplest CPU scheduling algorithm
- Here, the process that requests the CPU first, is allocated the CPU first
- It is a non preemptive scheduling algorithm
- Consider the following set of processes that arrive at time 0, with the length of the CPU burst time given in milliseconds
- Burst time is the total time taken by the process for its execution on the CPU.

Process	Burst Time
P1	24
P2	3
P3	3

Scheduling Algorithms

1. First Come First Served (FCFS) Scheduling Algorithm

- Let these processes arrive in the order P_1 , P_2 , P_3
- The execution of these processes in FCFS order is given in the Gantt chart below

- Waiting Time for $P_1 = 0$
- Waiting Time for $P_2 = 24$
- Waiting Time for $P_3 = 27$
- Average Waiting Time $= \frac{0+24+27}{3} = 17$

Scheduling Algorithms

1. First Come First Served (FCFS) Scheduling Algorithm

- Let these processes arrive in the order P_2 , P_3 , P_1
- The execution of these processes in FCFS order is given in the Gantt chart below

- Waiting Time for $P_1 = 6$
- Waiting Time for $P_2 = 0$
- Waiting Time for $P_3 = 3$
- Average Waiting Time $= \frac{6+0+3}{3} = 3$
- Thus we can see that the average waiting time varies substantially, if the CPU burst times vary greatly

Scheduling Algorithms

2. Shortest Job First (SJF) Scheduling Algorithm

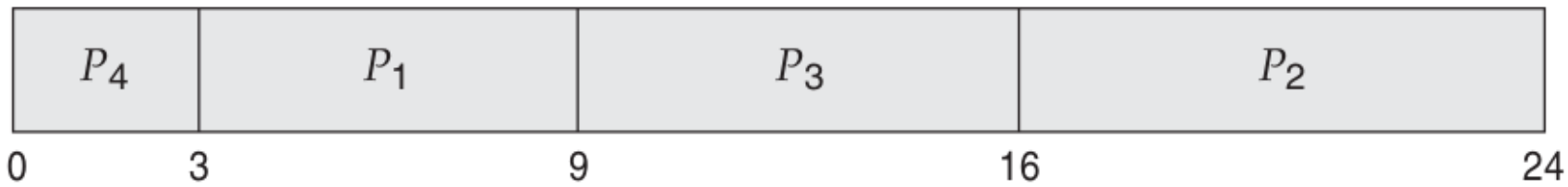
- Here, the process that has the shortest burst time is allocated the CPU first
- The CPU is assigned next to the process having the smallest burst time among the remaining ones
- This is continued till all processes are completed
- If 2 processes are having the same burst time, then FCFS scheduling algorithm is used to break the tie

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

Scheduling Algorithms

2. Shortest Job First (SJF) Scheduling Algorithm

- The execution of these processes using this algorithm is given in the Gantt chart below



- Waiting Time for $P_1 = 3$
- Waiting Time for $P_2 = 16$
- Waiting Time for $P_3 = 9$
- Waiting Time for $P_4 = 0$
- Average Waiting Time $= \frac{3+16+9+0}{4} = 7$ ms
- This is less than the average waiting time in FCFS scheduling which is 10.25 ms

Scheduling Algorithms

2. Shortest Job First (SJF) Scheduling Algorithm

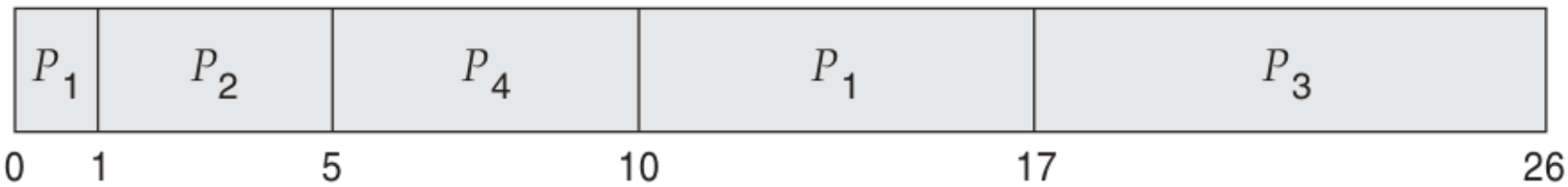
- There is a preemptive version of the SJF scheduling algorithm
- Here the currently executing process is preempted if its remaining time is greater than the burst time of a newly arriving process
- This is called Shortest Remaining Time First algorithm

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Scheduling Algorithms

2. Shortest Job First (SJF) Scheduling Algorithm

- The execution of these processes using this algorithm is given in the Gantt chart below



- Waiting Time for $P_1 = 10 - 1 = 9$
- Waiting Time for $P_2 = 1 - 1 = 0$
- Waiting Time for $P_3 = 17 - 2 = 15$
- Waiting Time for $P_4 = 5 - 3 = 2$
- Average Waiting Time $= \frac{9+0+15+2}{4} = 6.5$ ms
- This is less than the average waiting time in non preemptive SJF scheduling which is 7.75 ms

Scheduling Algorithms

3. Priority Scheduling Algorithm

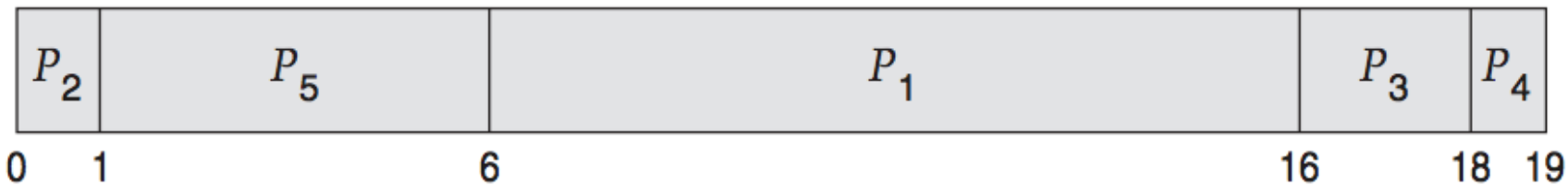
- Here, each process is assigned some priority
- The CPU is assigned first to the process having the highest priority
- The CPU is assigned next to the process having the next higher priority
- This is continued till all processes are completed
- If 2 processes are having the same priority, then FCFS scheduling algorithm is used to break the tie
- Here lower numbers represent higher priority

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Scheduling Algorithms

3. Priority Scheduling Algorithm

- The execution of these processes using this algorithm is given in the Gantt chart below



- Waiting Time for $P_1 = 6$
- Waiting Time for $P_2 = 0$
- Waiting Time for $P_3 = 16$
- Waiting Time for $P_4 = 18$
- Waiting Time for $P_5 = 1$
- Average Waiting Time $= \frac{6+0+16+18+1}{5} = 8.2$ ms

Scheduling Algorithms

3. Priority Scheduling Algorithm

- A priority scheduling algorithm can leave some low-priority processes waiting indefinitely
- This problem is called indefinite blocking or starvation
- A solution to this problem is to gradually increase the priority of processes that wait in the system for a long time
- This is called aging
- A priority scheduling algorithm can be preemptive also
- A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process

Scheduling Algorithms

4. Round Robin(RR) Scheduling Algorithm

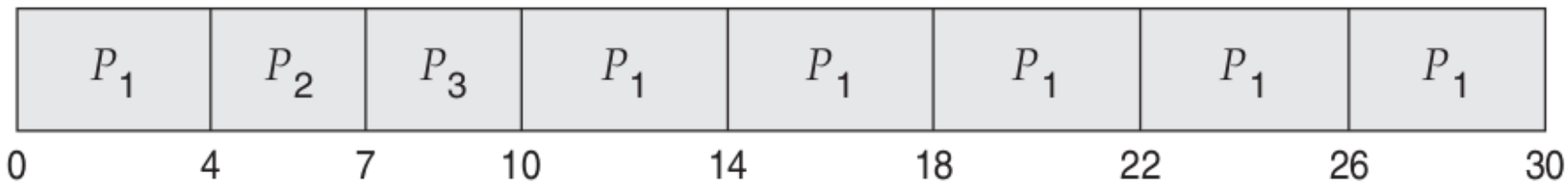
- Here, CPU is allocated to each process for a time interval called a time quantum or time slice
- After that time quantum, the currently running process is preempted and the next process is executed
- This is continued till all the processes are completed
- The process which has arrived first begins its execution first
- If the execution time of a process is less than the time quantum, the CPU is released voluntarily by the process

Process	Burst Time
P1	24
P2	3
P3	3

Scheduling Algorithms

4. Round Robin(RR) Scheduling Algorithm

- Let the time quantum be 4ms
- The execution of these processes using this algorithm is given in the Gantt chart below



- Waiting Time for P_1 = 10 - 4 = 6
- Waiting Time for P_2 = 4
- Waiting Time for P_3 = 7
- Average Waiting Time = $\frac{6+4+7}{3}$ = 5.66 ms

Scheduling Algorithms

4. Round Robin(RR) Scheduling Algorithm

- The performance of the RR algorithm depends heavily on the size of the time quantum
- If the time quantum is extremely large, the RR policy is same as the FCFS policy
- If the time quantum is extremely small (say, 1 millisecond), the RR approach can result in a large number of context switches
- This will slow down the execution of processes

Concepts of Threads

- A thread is a basic unit of execution within a process
- A thread performs a single task associated with a process
- A thread is a lightweight process
- A process can have many threads within it
- A web browser might have one thread display images or text while another thread retrieves data from the network
- A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

Problems of Concurrent Processes : The Critical Section Problem

- Concurrent Processes(or Threads) access shared variables at the same time
- A Critical Section is a code segment in a process in which a shared variable is accessed
- In such a situation, the integrity of variables may be violated
- To solve this problem (Critical Section Problem) we have to ensure that processes need to synchronise in such a way that only one process access the shared variables at one time
- Process Synchronisation is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources
- A resource is any hardware or software associated with a computer.
- We have to ensure that no two processes are executing in their critical sections at the same time

The Critical Section Problem

- Each process must request permission to enter its critical section
- The section of code implementing this request is the entry section
- The critical section may be followed by an exit section
- The remaining code is the remainder section

```
while (true) {

    entry section

        critical section

    exit section

        remainder section

}
```

The Critical Section Problem

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

1. Mutual Exclusion
   Only one process can execute in its critical section in a given period of time.
2. Progress
   When no process is executing in its critical section, then any process which wants to enter its critical section must be granted permission without delay.
   If there are two or more processes which want to enter their critical sections, then the selection cannot be postponed indefinitely.
3. Bounded Waiting
   No process can prevent any other from entering its critical section indefinitely. In other words, there exists a bound on the waiting time of a process before entering its critical section.

Semaphores

- A solution to this problem is the usage of semaphores
- A semaphore is an integer variable that, apart from initialisation, is accessed only through two standard atomic operations: wait() and signal() .
- Semaphores were introduced by the Dutch computer scientist E.W.Dijkstra

```
wait(S)
{
     while (S <= 0);  // If S is not positive no
                      // operation is performed
     S = S - 1;  // If S is positive, it is decremented
                 // by 1
}
signal(S)
{
     S = S + 1; // S is incremented by 1
}
```

Semaphores

- There are two types of semaphores

1. counting semaphore
   They have an unrestricted value domain. The semaphore count is the number of available resources. If the resources are added, semaphore count is incremented and if the resources are removed, the count is decremented.
2. binary semaphore
   Their values are restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation works only when semaphore is 0.

- Binary semaphores are easier to implement than counting semaphores

Semaphore

- Semaphore Usage
- Consider two concurrently running processes: P1 with a statement S1 and P2 with a statement S2.
- Suppose we require that S2 be executed only after S1 has completed
- We can implement this scheme by letting P1 and P2 share a common binary semaphore S, initialised to 0

```
Process P1          Process P2
----------          ----------

..........          ..........
S1 ;                wait(S);
signal(S);          S2;
..........          ..........
```

- The object used for achieving mutual exclusion is called mutex
- Here binary semaphore S is the mutex

Problems of Concurrent Processes: The Producer Consumer Problem

- A Producer process produces information that is consumed by the Consumer process
- Example
- A web server produces (that is, provides) HTML files and images, which are consumed (that is, read) by the client web browser requesting the resource
- One solution to the producer consumer problem uses shared memory
- These processes share a common buffer pool where items are deposited by producers and removed by consumers

The Producer Consumer Problem

- Two types of buffers can be used
- The unbounded buffer places no practical limit on the size of the buffer
- The consumer may have to wait for new items, but the producer can always produce new items
- The bounded buffer assumes a fixed buffer size
- In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full

The Producer Consumer Problem

- Another solution to the producer consumer problem uses message passing
- Here a producer process communicates with a consumer process by sending messages
- The message passing model uses two communication primitives - SEND and RECEIVE
- The SEND primitive has two parameters - a message and its destination
- The RECEIVE primitive has two parameters - the source of the message and a buffer for storing the message

Deadlocks

- In a multi programming environment, several processes may compete for a finite number of resources
- A process requests resources
- If the resources are not available at that time, the process enters a waiting state
- Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes
- This situation is called a deadlock
- Example
- P_1 is holding a resource R1 and waiting for resource R2
- P_2 is holding a resource R2 and waiting for resource R1

Deadlocks

- A deadlock situation can arise if the following four conditions hold simultaneously in a system

1. mutual exclusion

- At least one process must be held in non shareable mode; that is only one process can use this resource at a time

2. Hold and Wait

- A process holds at least one resource and waits to acquire other resources that are currently being held by other processes

3. No Preemption

- Resources cannot be preempted; They are released voluntarily by processes after completing their tasks

4. Circular Wait

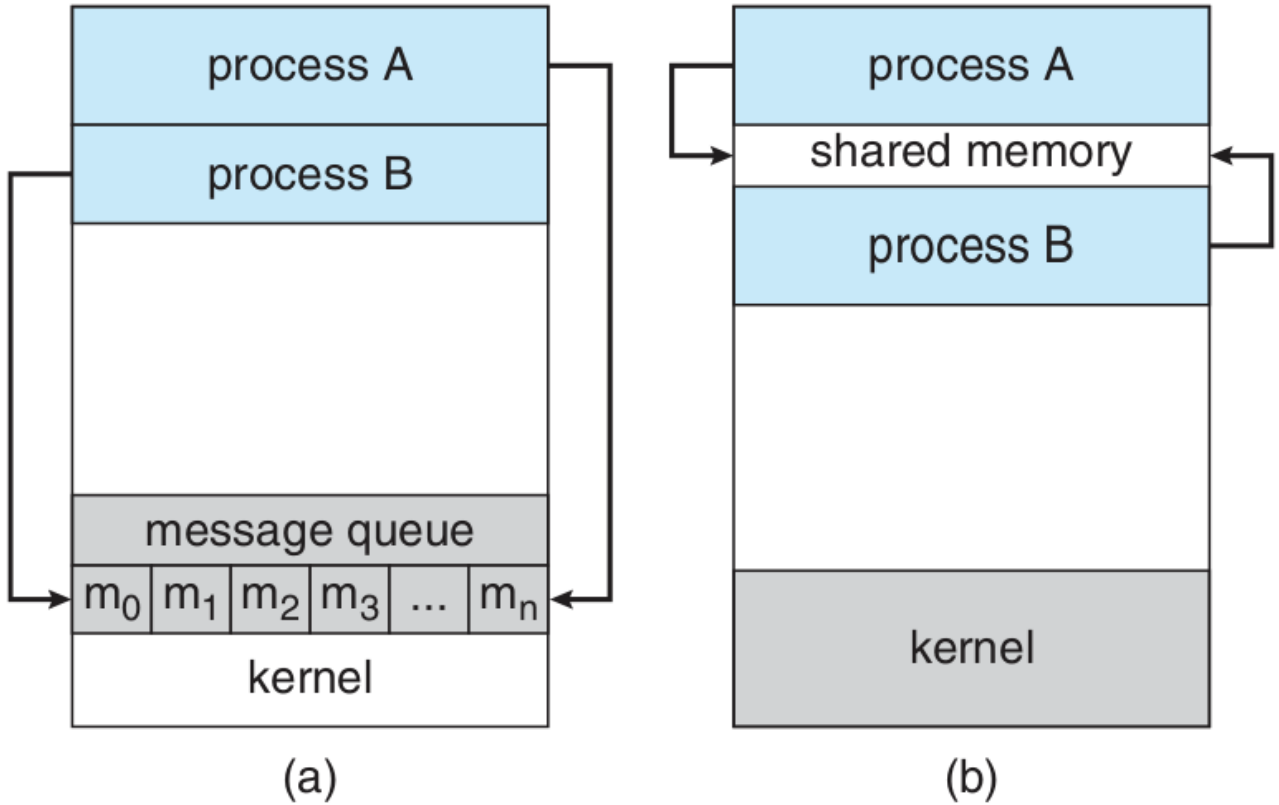
- A set $\{P_0 , P_1 , ..., P_n \}$ of waiting processes must exist such that P_0 is waiting for a resource held by P_1 , P_1 is waiting for a resource held by P_2 , ..., P_{n-1} is waiting for a resource held by P_n , and P_n is waiting for a resource held by P_0

Inter Process Communication

- Processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information
- There are two fundamental models of interprocess communication: shared memory model and message passing model
- In the shared memory model, a region of memory is shared by the cooperating processes
- Processes can then exchange information by reading and writing data to the shared region
- In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes

Inter Process Communication

- (a) Message Passing Model (b) Shared Memory Model



- Message passing model is useful for exchanging smaller amounts of data
- Shared memory model can be faster than message passing model

Inter Process Communication

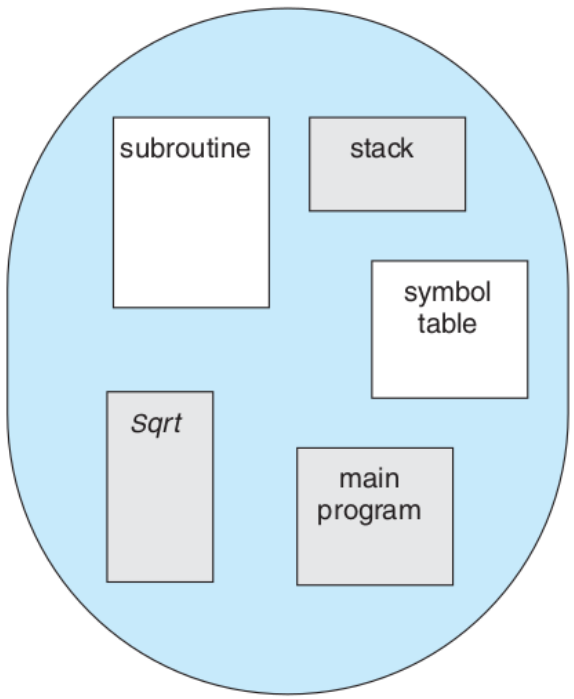
- The message passing model uses two communication primitives - SEND and RECEIVE
- These primitives can have four modes - blocking,non blocking, synchronous and asynchronous
- In blocking mode, SEND primitive does not return control to the sender process until message has been sent, or an acknowledgement has been received
- In non blocking mode, SEND primitive returns control to the sender process as soon as the message is copied to kernel buffer, an intermediate buffer used for storing message
- In synchronous mode, SEND primitive is blocked until the execution of the corresponding RECEIVE message (Another SEND cannot be performed before the completion of RECEIVE)
- In asynchronous mode, SEND primitive is not blocked until the execution of the corresponding RECEIVE message (Another SEND can be performed before the completion of RECEIVE)

Memory Organisation

- In operating systems, physical and logical addresses are used to access memory
- A physical address is the actual address in main memory where data is stored
- An address generated by the CPU during program execution is called a logical address (virtual address)
- The set of all logical addresses generated by a program is a logical address space
- The set of all physical addresses corresponding to these logical addresses is a physical address space
- The run-time mapping from logical to physical addresses is done by a hardware device called the memory-management unit (MMU)

Memory Management

- A programmer's view of memory is different from the actual physical memory
- When writing a program, a programmer thinks of it as a main program with a set of methods, procedures, or functions
- It may also include various data structures: objects, arrays, stacks, variables, and so on



References

1. A. Silberchatz et.al., “Operating System Concepts”, 9th Edition Wiley (2015)
2. https://www.coursera.org/learn/os-power-user