Unit 2
Process and Thread

Marwadi University

Computer Engineering Diploma

Unit no 2
Process and Threads
Operating System
09CE2405

Prof.Foram Chovatiya

What is Program?

- Set of Instructions
- Stored on disk
- Also called **software/Application**

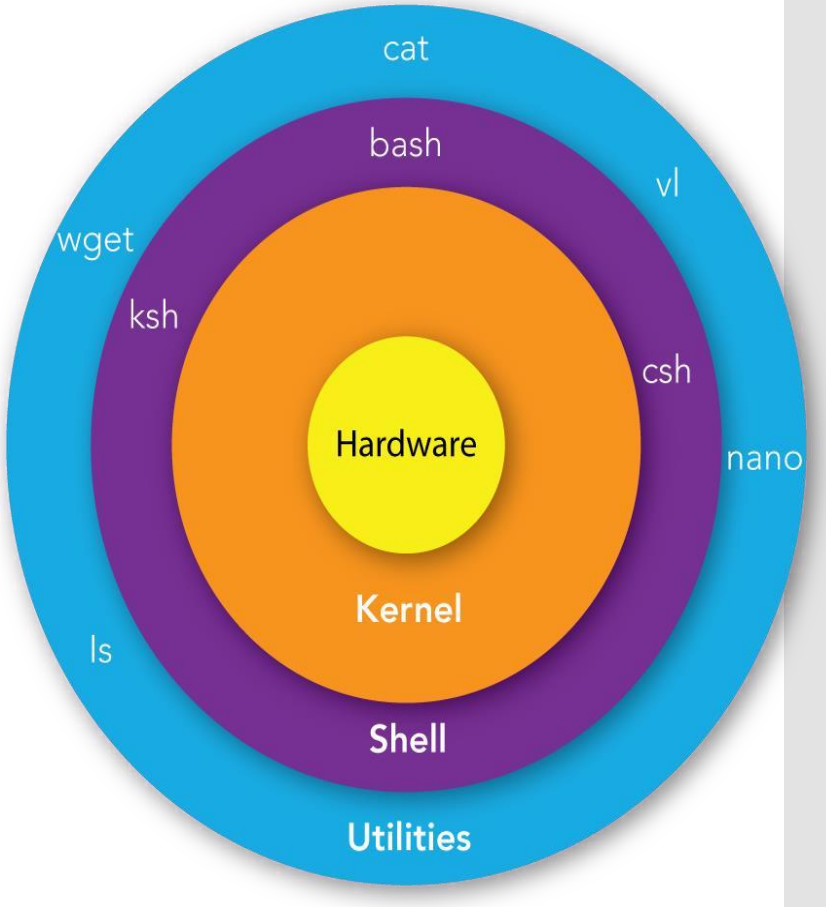
Process

"A program in execution"

- **Examples of Processes**
- User Processes
  - Shells
  - Text editors, databases
- System Processes
  - Memory Management, Process Scheduling, System Background Processes.

Shell

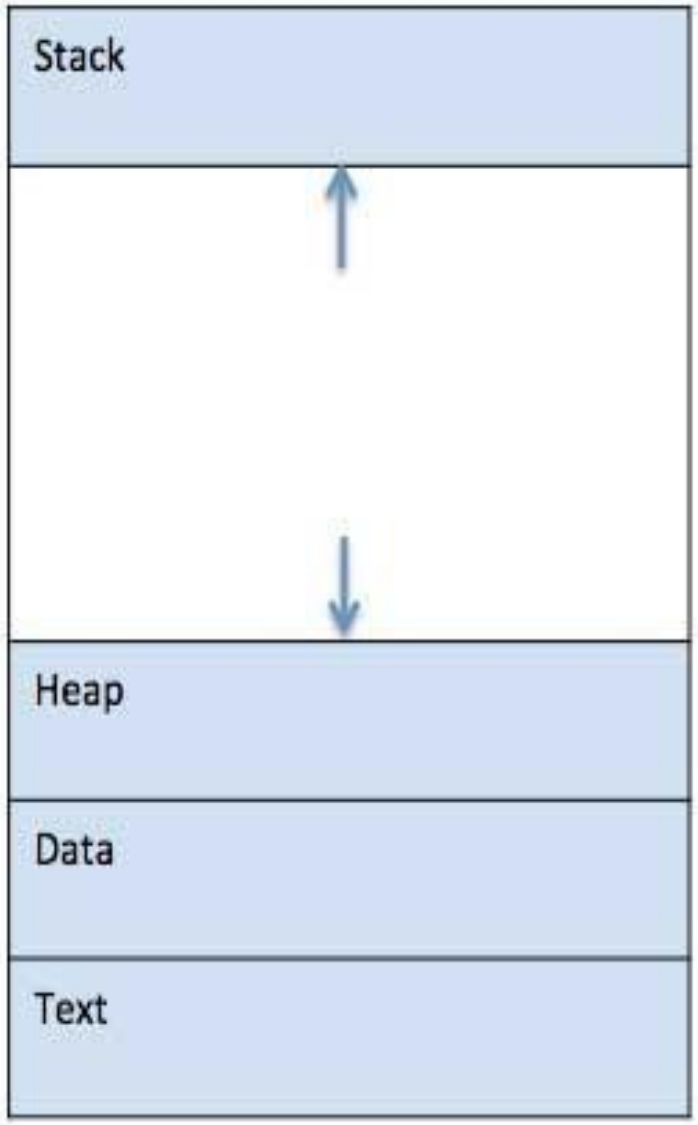
- A shell is special user program which provide an interface to user to use operating system services.
- Shell accept human readable commands from user and convert them into something which kernel can understand



Process

- The execution of a process must progress in a sequential fashion.
- When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, data and text.
- This is also known User Level Context

Process in Memory



Process in Memory

(1) Stack

- The process Stack contains the temporary data such as method/function parameters, return address and local variables.

(2) Heap

- This is dynamically allocated memory to a process during its runtime.

(3) Data

- This section contains the global and static variables.

(4) Text

- This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

Process Life Cycle

When a process executes, it passes through different State.

(1) Start

(2) Ready

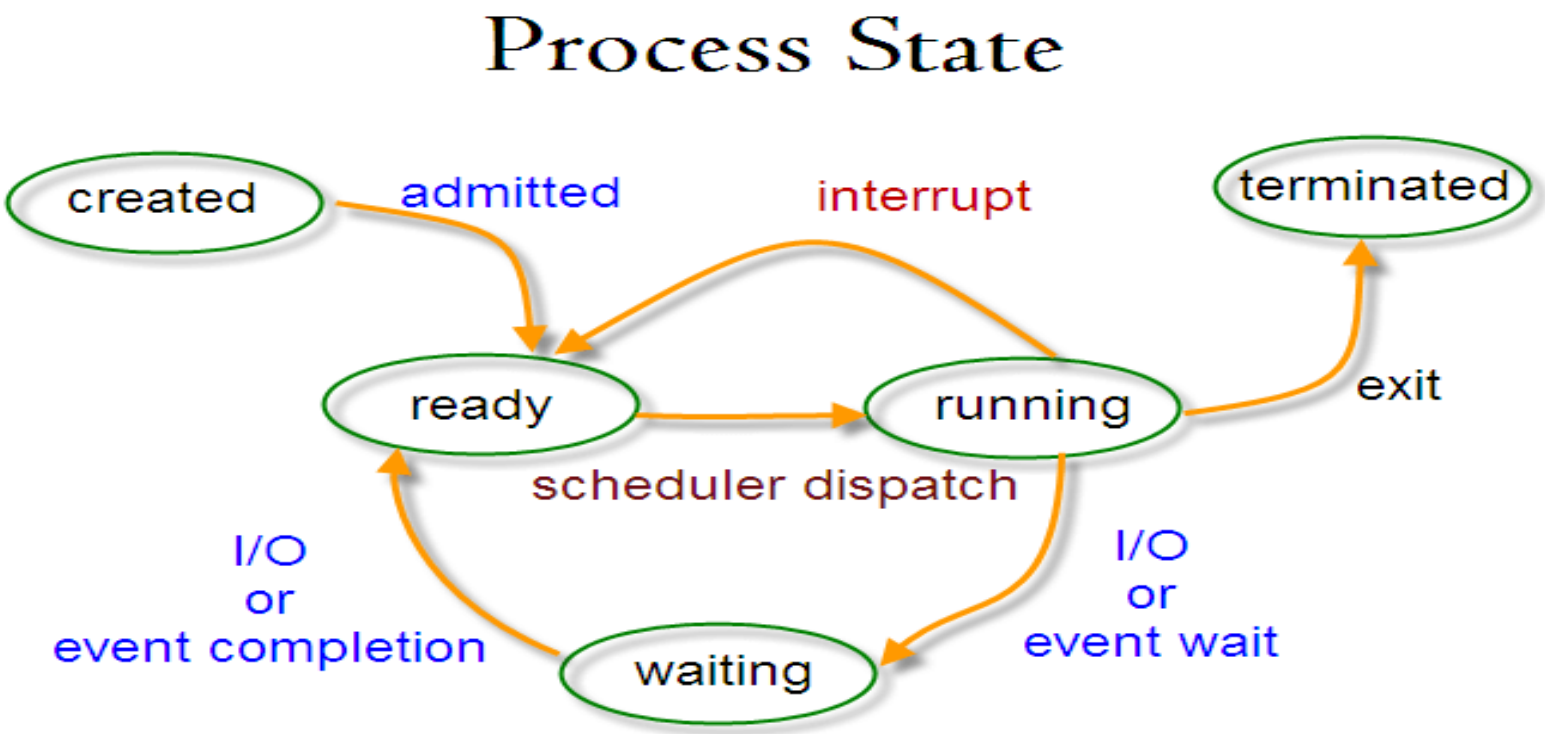
(3) Running

(4) Waiting

(5) Terminate or End

Process State

A process changes state as it executes.



Process States

As a process executes, it changes state. The state of a process is defined in part.

- By the current activity of that process, A process may be in one of the following states:

- **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 States

(1) Start

- This is the initial state when a process is first started/created.

(2) Ready

- Ready processes are waiting to have the processor allocated to them by the operating system so that they can run.
- Process may come into this state after **Start** state or **while running** – case in which it interrupted by the scheduler to assign CPU to some other process.

(3) Running

- Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

Process States

(4) Waiting

- Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

(5) Terminated or Exit

- Once the process finishes its execution, or it is terminated by the operating system.
- it is moved to the terminated state where it removed from main memory.

Process Transition

State Transition	Description
Ready → Running	CPU start or resumes its execution
Running → Ready	Interrupt Occurs
Running →Waiting	Process request for I/O Request for memory or some other resources. Waits for message from other process.
Waiting → Ready	Request made by process is satisfied or event for which process was waiting occurs.

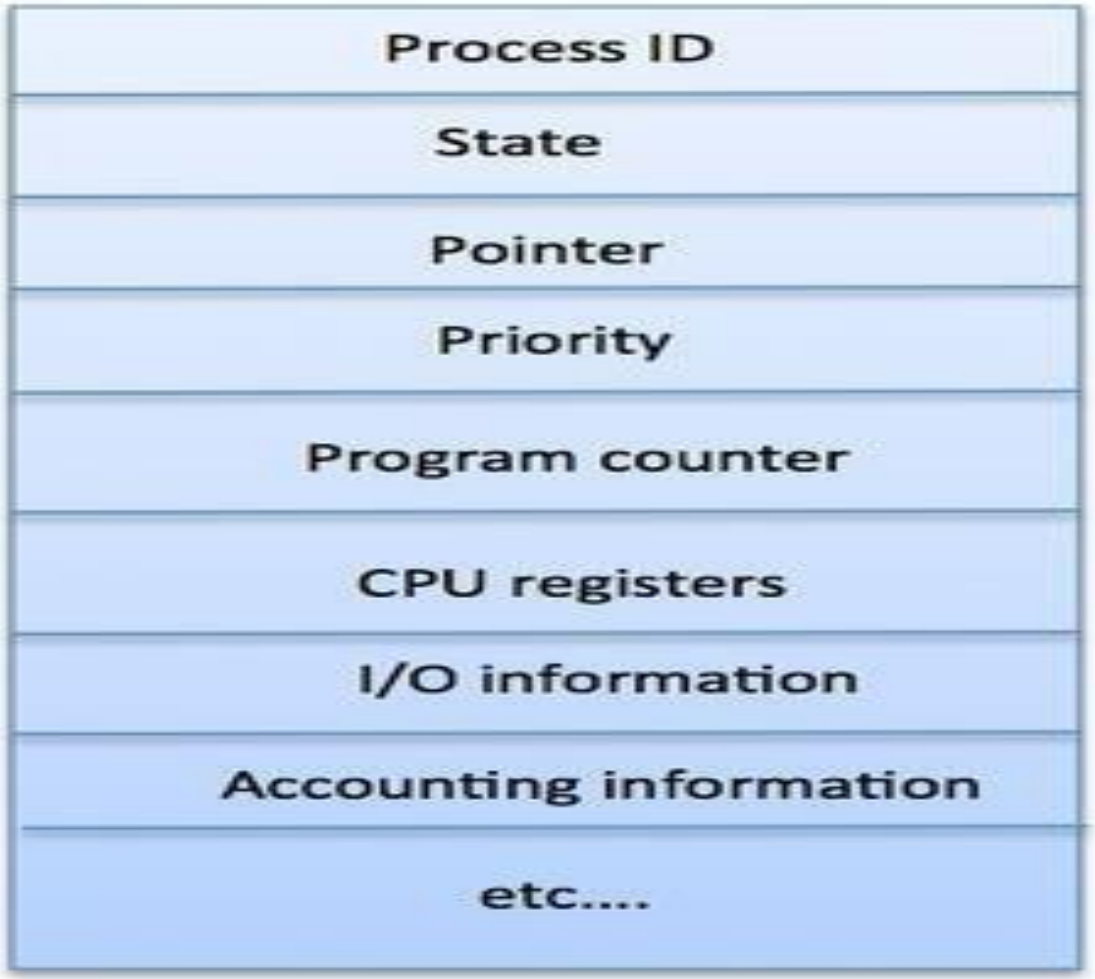
Process Table

- Process table is used to maintain process information (one for each process)
- Process table is array of structure
- Each entry in a process table can be considered as a **Process Control Block.**
- PCB must be saved when a process is switched from ***running*** **to** ***ready*** or ***blocked*** **states**

Process Control Block

- A Process Control Block is a data structure maintained by the Operating System for every process.
- The PCB is identified by an integer process ID (PID).
- A PCB keeps all the information needed to keep track of a process.

Process Control Block



Process Control Block

(1) Process Number - Every process is assigned with a unique id known as Process ID.

(2) Process State - This specifies the process state i.e. new, ready, running, waiting or terminated.

(3) Program Counter - This contains the address of the next instruction that needs to be executed in the process.

(4) CPU registers - Indicates various register set of CPU where process need to be stored for execution.

Process Control Block

(5) CPU Scheduling Information - process priority

(6) Memory Management Information - This field contains the information about memory management system used by operating system. e.g. base registers, limit registers etc.

(7) I/O Status Information - This information includes the list of I/O devices used by the process, the list of files etc.

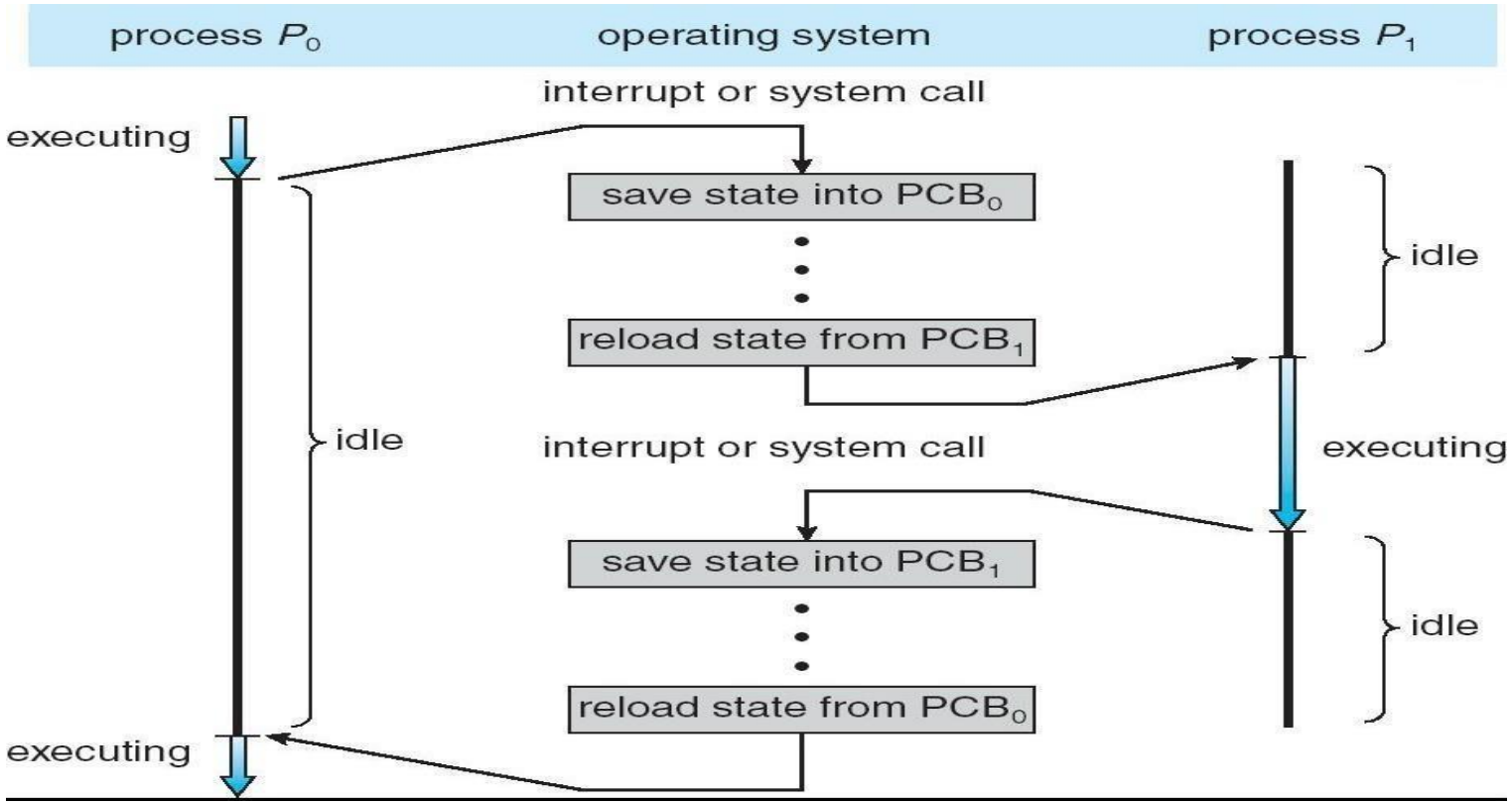
(8) Accounting Information - This information includes CPU utilization and execution time of a process.

Context Switching

- Task that switches CPU from one process to another process.
- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.

Context Switch PCB is a key to Multiprogram ming

- Using this technique, a context switcher enables multiple processes to share a single CPU.
- Time for context switch is dependent on hardware support (1-1000 microseconds).



Context Switching

- There are two situations that a context switch is necessary.

- **Multitasking (Concurrent Programming)** – When the CPU needs to switch processes in and out of memory, so that more than one process can be running.

- **Kernel/User Switch** – When switching between user mode to kernel mode, it may be used.

What is Thread?

"Thread is the smallest unit of processing that can be scheduled by an operating system".

- In traditional operating systems, each process has an address space and a single thread of execution.
- In many situations, it is desirable to have multiple threads of control in the same address space running in parallel, as though they were separate processes.

Thread

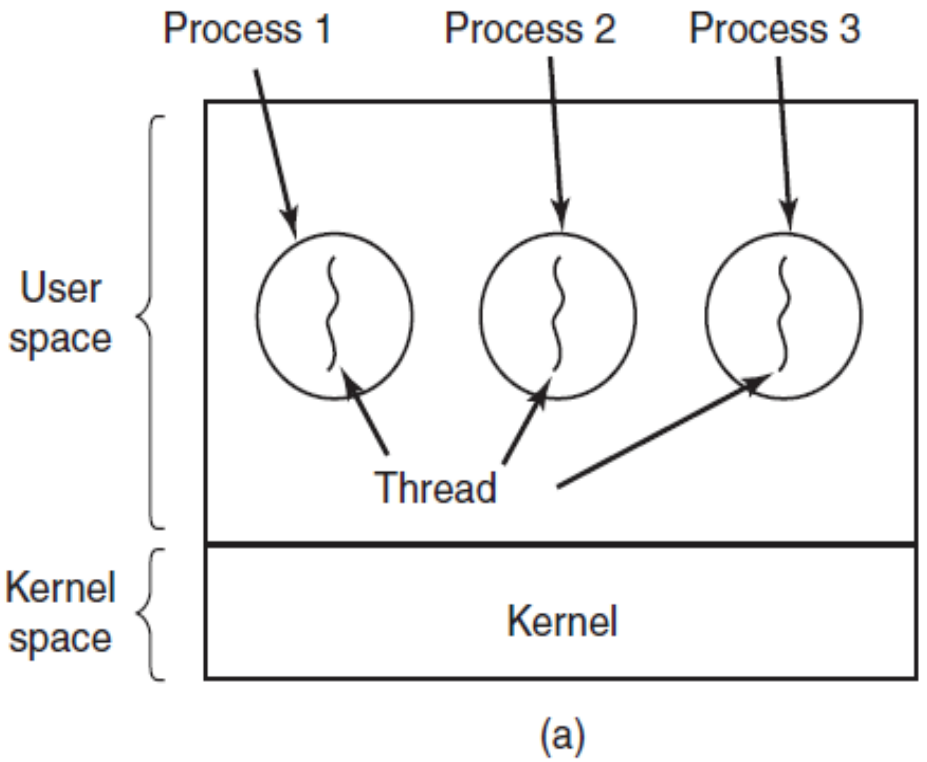
- Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others.
- *A thread is a path of execution within a process.*
- *Also, a process can contain multiple threads.*
- *Thread is also known as lightweight process.*

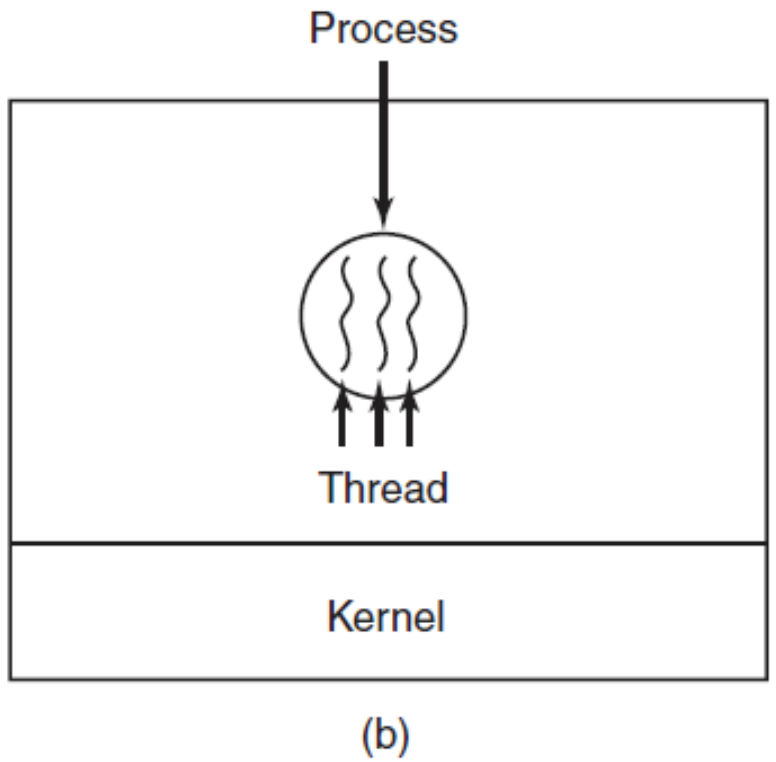
Thread

For example in a word processor,

- Background thread may check spelling and grammar
- Foreground thread processes user input (keystrokes)
- Third thread loads images from the hard drive
- Fourth does periodic automatic backups of the file being edited.

Thread





(a) Three processes each with one thread.

(b) One process with three threads.

Thread Continue….

- In Fig. (a) we see three traditional processes. Each process has its **own address** space and a single thread of control. Each of them operates in a different address space,
- In Fig. (b) we see a single process with three threads of control. all three of them **share the same address space.**
- Like a traditional process (i.e. a process with only one thread), a thread can be in any one of several states: **new, ready, running, blocked, or terminated.**

Thread Structure

- **Program Counter** → that keeps track of which instruction to execute next.
- **System Registers** → hold its current working variables.
- **Stack** → , which contains the execution history, with one frame for each procedure called but not yet returned from.

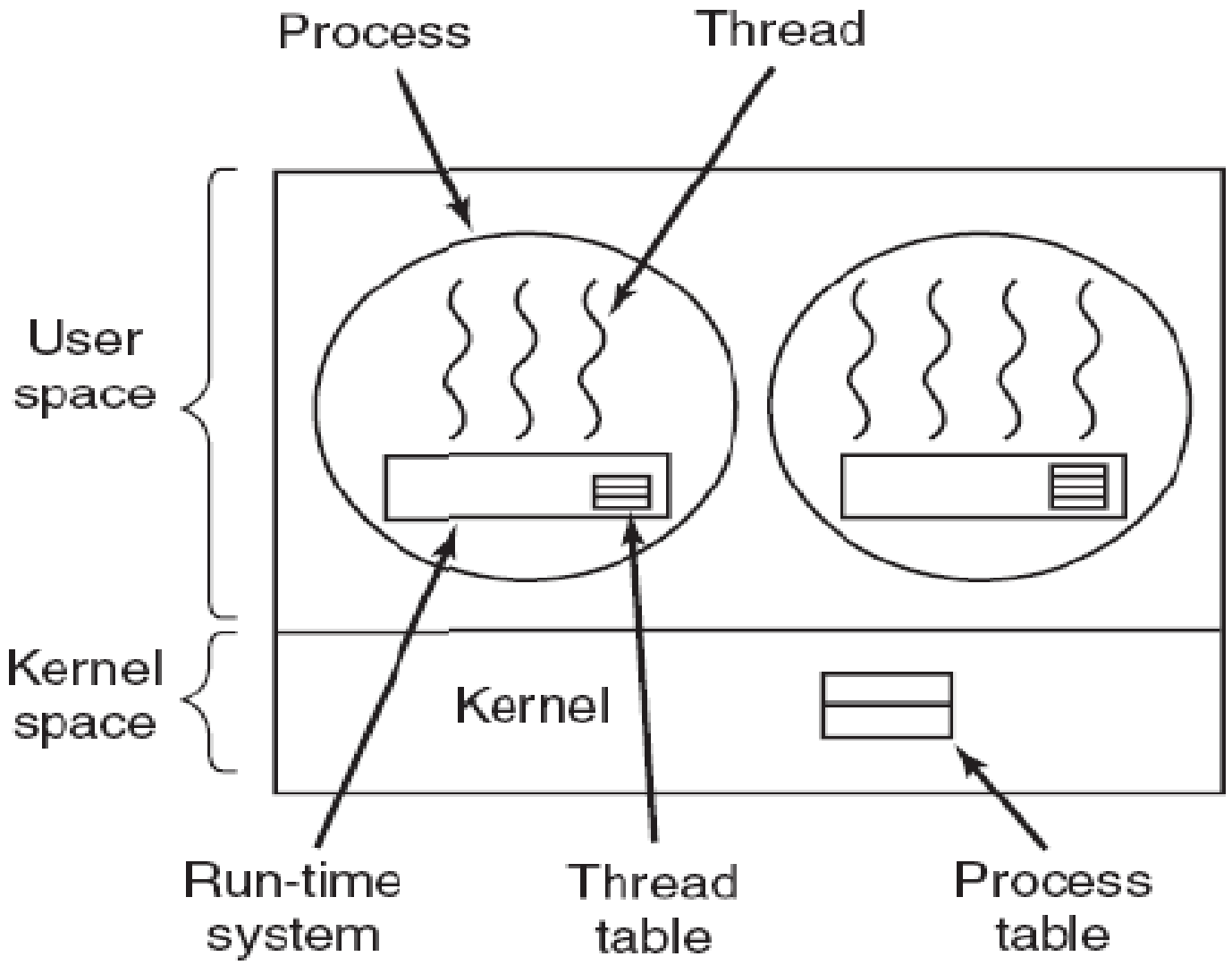
Types of Thread

- There are two types of thread.

(1) User Level Thread

(2) Kernel Level Thread

User Level Thread



User Level Thread

- User level threads are implemented in **user level libraries.**
- User-Level threads are managed entirely by the **run-time system** (user-level library).
- The kernel knows nothing about user-level threads and manages them as if they were single-threaded processes
- Creating a new thread, switching between threads, and synchronizing threads are done via procedure call.

  i.e no kernel involvement.
- User-Level threads are hundred times faster than Kernel-Level threads.
- Example : Java Threads

User Level Thread

- When threads are managed in user space, each process needs its **own** **private thread table** to keep track of the threads in that process.
- This table keeps track only of the thread properties, such as each thread's program counter, stack pointer, registers, state, and so forth.
- The thread table is managed by the **run-time system.**

Advantages of User level Threads

1. User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed.
2. User-level threads can be run on any operating system.
3. There are no kernel mode privileges required for thread switching in user-level threads.

Disadvantages of User level Threads

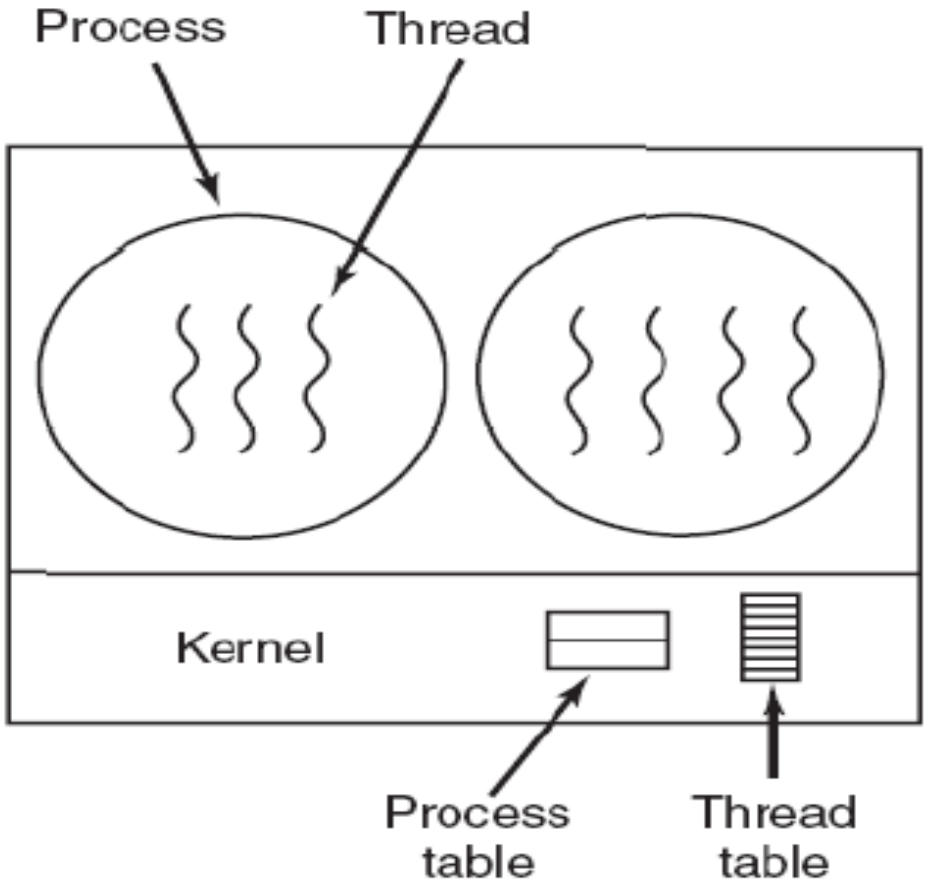
1. Multithreaded applications in user-level threads cannot use multiprocessing to their advantage.
2. The entire process is blocked if one user-level thread performs blocking operation.

Kernel Level Thread

- Here, kernel **knows** about threads and manages the threads.
- Instead of thread table in each process, the kernel has a **thread table** that keeps track of **all threads** in the system. (In addition to the traditional process table to keep track of processes).
- Kernel provides **system call** to create and manage threads.

- Example: Windows

Kernel Level Thread



Advantages of Kernel Level Thread

1. Because kernel has **full knowledge** of all threads, scheduler may decide to give more time to a process having large number of threads than process having small number of threads.
2. Blocking of one thread in a process will not affect the other threads in the same process as Kernel knows about multiple threads present so it will schedule other runnable thread.
3. So, Kernel threads do not require any new, non-blocking system calls.

Disadvantages of Kernel Level Threads

1. The kernel level threads are **slow and inefficient**. As threads are managed by system calls, at considerably greater cost.
2. Since kernel must manage and schedule threads as well as processes. It requires a full **Thread Control Block** (TCB) for each thread to maintain information about threads.
3. As a result there is significant **overhead** and increased in kernel complexity.

Difference between user level thread and kernel level thread

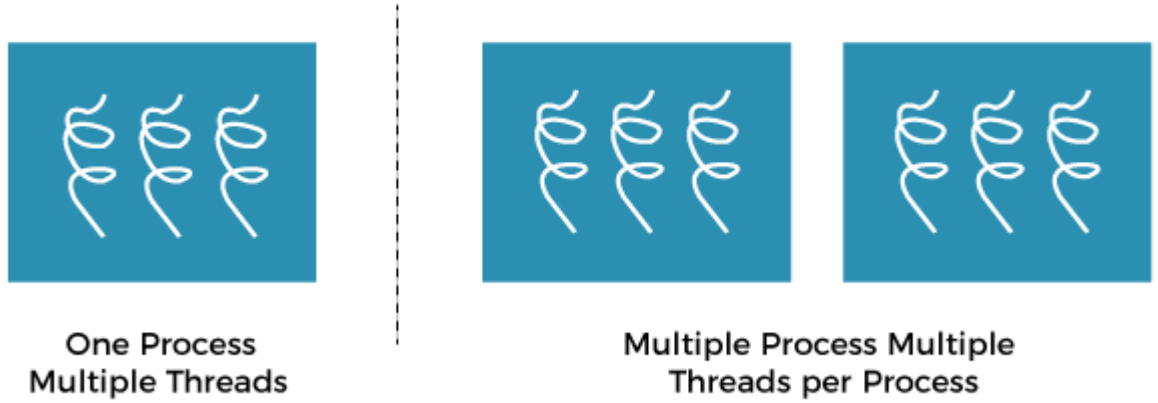
User Level Threads	Kernel Level Threads
1) The user level threads are **faster** and **easier** to create and manage.	1) The kernel-level threads are **slower** and **complex** to create and manage.
2) Supported above the kernel and implemented by thread library at user level.	2) Directly supported from the OS.
3) Can run on **any** OS.	3) **Specific** to the OS.
4) A **library** provides support for thread creation , scheduling and management with no support from kernel.	4) Thread creation , scheduling and management is done in **kernel space** with the help of OS directly.
5) **Kernel is unaware** of User level threads.	5) **Kernel has full knowledge** of these threads.
6) Process as a whole gets **"one time slice"** irrespective of whether process has one thread or 1000 threads within.	6) Scheduler may decide to give **more time** to a process having large number of threads than process having small number of threads.

Difference between process and thread

Process	Thread
1) Process is **heavy weight** and uses more resources.	1) Thread is **light weight** and uses less resources than a process.
2) Process switching needs interaction with operating system	2) Thread switching does not need to interact with operating system.
3) All the processes have **separate address space.**	3) All the threads of a process share **common address space**.
4) Process switching is **more expensive** .	4) Thread switching is **less expensive**.
5) Processes are **complex** to create because it requires separate address space.	5) Threads are **easier** to create because they don't require separate address space.
6) If one process terminates, its resources are deallocated and all its threads are terminated.	6) If a thread in a process terminates, another thread of the same process can run.
7) Processes can **live on its own**.	7) Threads **can't live on its on**. It must lie within a process.

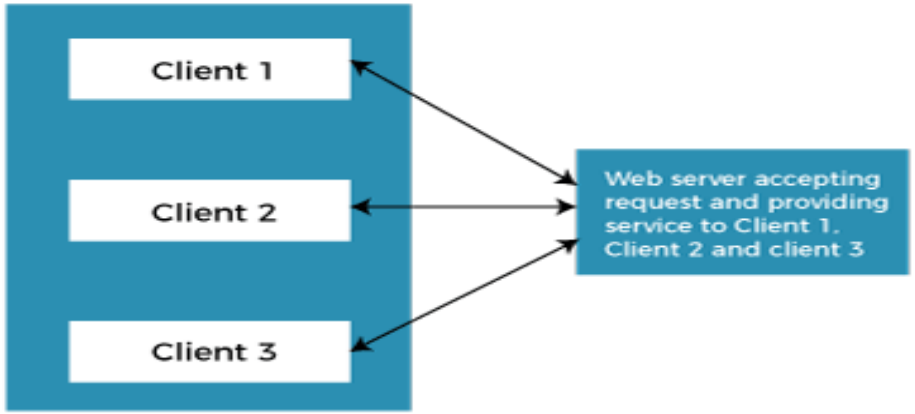
Multithreading Model

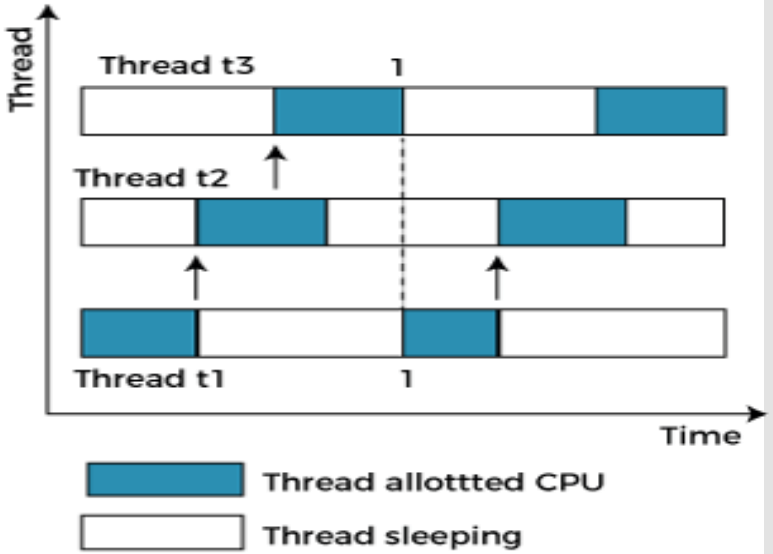
- Multithreading allows the application to divide its task into individual threads.
- In multi-threads, the same process or task can be done by the number of threads, or we can say that there is more than one thread to perform the task in multithreading.
- With the use of multithreading, multitasking can be achieved.



Multithreading Model

- The main drawback of single threading systems is that only one task can be performed at a time, so to overcome the drawback of this single threading, there is multithreading that allows multiple tasks to be performed.
- **For example:**





In the above example, client1, client2, and client3 are accessing the web server without any waiting. In multithreading, several tasks can run at the same time.

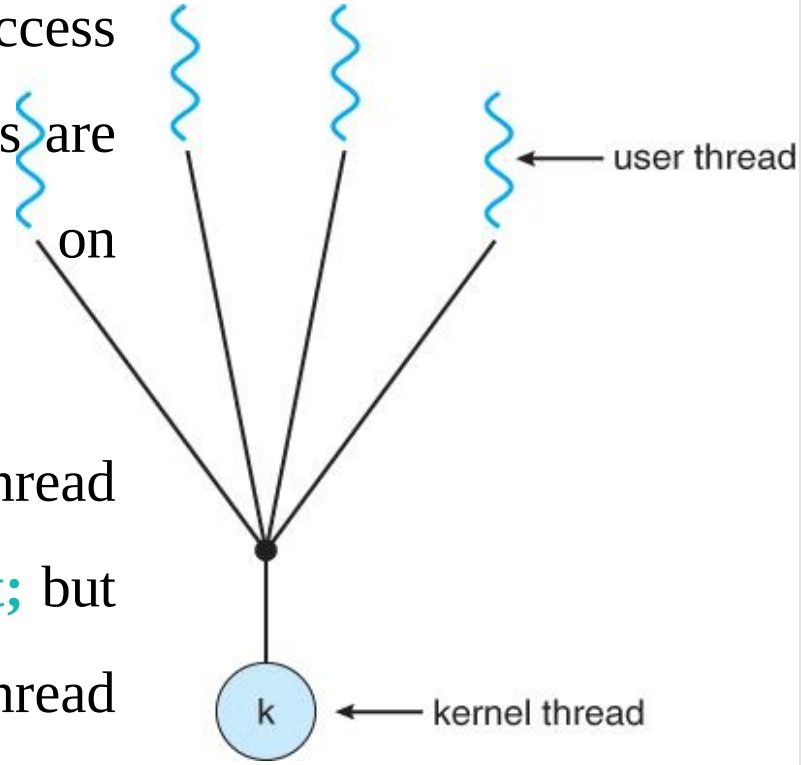
Multithreading Model

There exists three established multithreading models classifying these relationships are:

- Many to one multithreading model
- One to one multithreading model
- Many to Many multithreading models

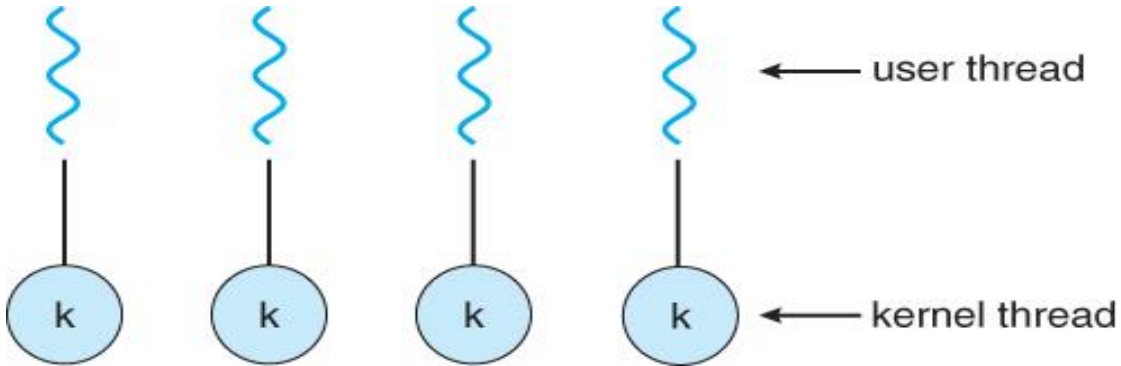
Many-to-One Model

- The many-to-one model maps many user-level threads to one kernel thread.
- Also, because **only one thread** can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.
- 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.



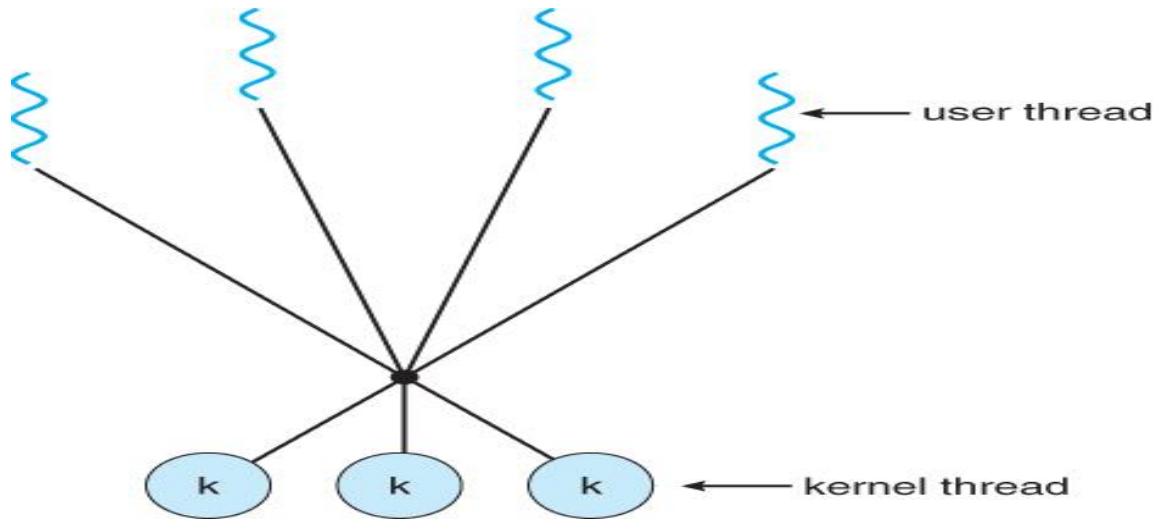
One-to-One Model

- The one-to-one model maps each user thread to a kernel thread.
- It provides **more concurrency** than the many-to-one model.
- It allows multiple threads to run in **parallel** on multiprocessors.
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.
- So, there is an **overhead** of creating kernel threads.



Many-to-Many Model

- The many-to-many model multiplexes many user-level threads to a **smaller or equal** number of kernel threads.
- The number of kernel threads may be specific to either a particular **application** or a particular machine.
- Developers can create as many user threads as necessary, and the corresponding kernel threads can run in **parallel** on a multiprocessor.



Advantages of Thread

- Responsiveness
- Faster context switch
- Effective Utilization of Multiprocessor system
- Resource sharing
- Communication
- Enhanced Throughput of the system

Process Management (Process Scheduling)

- Allocating the CPU to a different process to reduce idle time
- Each process change requires a *context switch*
- A context switch is *pure overhead*

Preemptive

- In preemptive scheduling, the CPU is allocated to the processes for a limited time.

Non-Preemtive

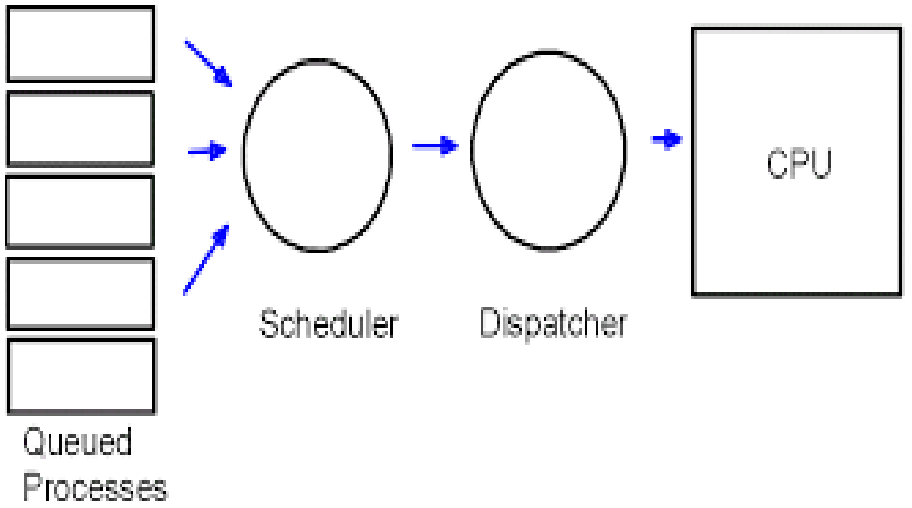
- Under non preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

CPU Scheduler

- Selects among the processes in memory that are ready to execute and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
    1. Switches from running to waiting state
    2. Switches from running to ready state
    3. Switches from waiting to ready
    4. Terminates
- Scheduling under 1 and 4 is nonpreemptive (context switch is caused by the running program)
- All other scheduling is preemptive (context switch caused by external reasons)

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by scheduler.
- This involves:
  - switching context



- Dispatch latency –
  - Time taken by the dispatcher to stop one process and to start another process

Types of Scheduler

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

Long Term Scheduler

- Long term scheduler is also known as job scheduler.
- It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.
- Long Term scheduler mainly controls the degree of Multiprogramming.
- The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.

Long Term Scheduler

- If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time.
- This will reduce the degree of Multiprogramming.
- Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

Medium Term Scheduler

- Medium term scheduler takes care of the swapped out processes.
- If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting.
- Medium term scheduler is used for this purpose.
- It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping.

Medium Term Scheduler

- The medium term scheduler is responsible for suspending and resuming the processes.
- It reduces the degree of multiprogramming.
- The swapping is necessary to have a perfect mix of processes in the ready queue.

Short Term Scheduler

- Short term scheduler is also known as CPU scheduler.
- It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.
- A scheduling algorithm is used to select which job is going to be dispatched for the execution.
- The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.
- This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

Scheduling Criteria

- **Burst time** : Burst time is the total time taken by the process for its execution on the CPU.
- **Arrival time:** Arrival time is the time when a process enters into the ready state and is ready for its execution.
- **Exit time** : Exit time is the time when a process completes its execution and exit from the system.
- **Response time :** Response time is the time spent when the process is in the ready state and gets the CPU for the first time.

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

Scheduling Criteria

- **Turnaround time:** Turnaround time is the total amount of time spent by the process from coming in the ready state for the first time to its completion.
- **Turnaround time = Burst time + Waiting time**
- **or**
- **Turnaround time = Exit time (Completion time)- Arrival time**

Scheduling Criteria

- **Waiting time :** Waiting time is the total time spent by the process in the ready state waiting for CPU.

Waiting time = Turnaround time - Burst time

- There is a difference between waiting time and response time.
- Response time is the time spent between the ready state and getting the CPU for the first time.
- But the waiting time is the total time taken by the process in the ready state.

Scheduling Criteria

- **Throughput**
- Throughput is a way to find the efficiency of a CPU. It can be defined as the number of processes executed by the CPU in a given amount of time.

EXAMPLE:

Process	Arrival time	Burst time
P1	0 ms	8 ms
P2	1 ms	7 ms
P3	2 ms	10 ms

- Here in the above example, **the arrival time** of all the 3 processes are 0 ms, 1 ms, and 2 ms respectively.

Here, the **response time** of all the 3 processes are:

P1: 0 ms

P2: 7 ms because the process P2 have to wait for 8 ms during the execution of P1 and then after it will get the CPU for the first time. Also, the arrival time of P2 is 1 ms. So, the response time will be 8-1 = 7 ms.

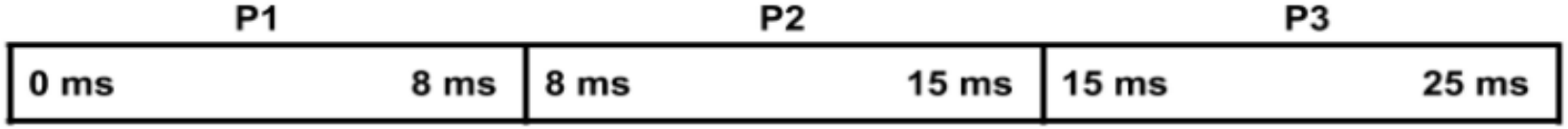
P3: 13 ms because the process P3 have to wait for the execution of P1 and P2 i.e. after 8+7 = 15 ms, the CPU will be allocated to the process P3 for the first time. Also, the arrival of P3 is 2 ms. So, the response time for P3 will be 15-2 = 13 ms.

EXAMPLE:

Process	Arrival time	Burst time
P1	0 ms	8 ms
P2	0 ms	7 ms
P3	2 ms	10 ms

For example, consider the arrival time of all the below 3 processes to be 0 ms, 0 ms, and 2 ms.

Gantt Chart



Then the **waiting time** for all the 3 processes will be:

- **P1:** 0 ms
- **P2:** 8 ms because P2 have to wait for the complete execution of P1 and arrival time of P2 is 0 ms.
- **P3:** 13 ms becuase P3 will be executed after P1 and P2 i.e. after 8+7 = 15 ms and the arrival time of P3 is 2 ms. So, the waiting time of P3 will be: 15-2 = 13 ms.

Waiting time = Turnaround time - Burst time

EXAMPLE:

- **Turnaround time = Burst time + Waiting time**
- For example, if we take the First Come First Serve scheduling algorithm, and the order of arrival of processes is P1, P2, P3 and each process is taking 2, 5, 10 seconds.
- Then the turnaround time of P1 is 2 seconds because when it comes at 0th second, then the CPU is allocated to it and so the waiting time of P1 is 0 sec and the turnaround time will be the Burst time only i.e. 2 seconds.
- The turnaround time of P2 is 7 seconds because the process P2 have to wait for 2 seconds for the execution of P1 and hence the waiting time of P2 will be 2 seconds. After 2 seconds, the CPU will be given to P2 and P2 will execute its task. So, the turnaround time will be 2+5 = 7 seconds.
- Similarly, the turnaround time for P3 will be 17 seconds because the waiting time of P3 is 2+5 = 7 seconds and the burst time of P3 is 10 seconds. So, turnaround time of P3 is 7+10 = 17 seconds.

EXAMPLE:

- For example, let's say, the process P1 takes 3 seconds for execution, P2 takes 5 seconds, and P3 takes 10 seconds. So, **throughput**, in this case, the throughput will be (3+5+10)/3 = 18/3 = 6 seconds.

Scheduling Algorithm Objectives

- Maximum CPU utilization
- Maximum throughput
- Minimum turnaround time
- Minimum waiting time
- Minimum response time

First Come First Serve (FCFS)

- First job that requests the CPU gets the CPU
- Non preemptive
- Process continues till the burst cycle ends

FCFS Example

Process	Burst Time
P1	24
P2	3
P3	3

The Gantt Chart for the schedule is:

FCFS Continue Previous example

Process	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT) or Exit Time	Turn Around Time (TAT) TAT=CT-AT	Waiting Time (WT) WT= TAT-BT
P1	0	24	24	24	0
P2	0	3	27	27	24
P3	0	3	30	30	27

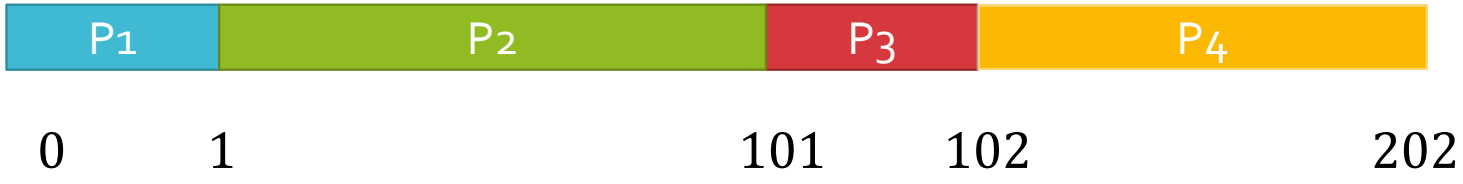
Average waiting time: (0 + 24 + 27)/3 = 17

Average Turn Around time: (24 + 27 + 30)/3 = 27

FCFS Example 2

Process	Arrival time (AT)	Burst Time (BT)
P1	0	1
P2	1	100
P3	2	1
P4	3	100

The Gantt Chart for the schedule is:



FCFS Example2

Process	Arrival time(AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT=CT-AT	Waiting Time (WT) WT = TAT –BT
P1	0	1	1	1	0
P2	1	100	101	100	0
P3	2	1	102	100	99
P4	3	100	202	199	99

Average waiting time: (0 + 0 + 99 + 99)/4 = 49.5

Average Turn Around time: (1 + 100 + 100 + 199)/4 = 100

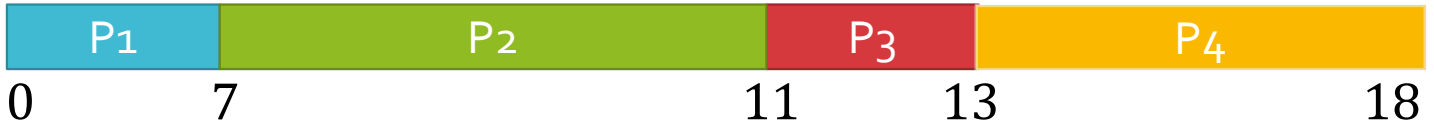
FCFS Example

Process	Arrival Time	Burst Time
P1	0	7
P2	0	4
P3	0	2
P4	0	5

FCFS Example

Process	Arrival time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT=CT-AT	Waiting Time (WT) WT = TAT –BT
P1	0	7	7	7	0
P2	0	4	11	11	7
P3	0	2	13	13	11
P4	0	5	18	18	13

The Gantt Chart for the schedule is:



Average waiting time = (0+7+13+18)/4 = 7.75

Average Response time= (0+7+11+13)/4 = 7.75

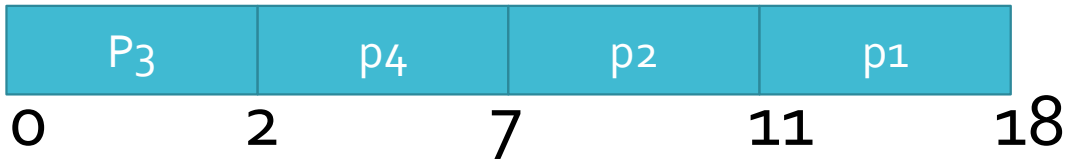
FCFS EXAMPLE

Process	Arrival Time	Burst Time
P1	0	7
P2	0	4
P3	0	2
P4	0	5

FCFS Example

- Oder of scheduling matters:

Process	Arrival time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT=CT-AT	Waiting Time (WT) WT = TAT –BT
P1	0	7	18	18	11
P2	0	4	11	11	7
P3	0	2	2	2	0
P4	0	5	7	7	2



Find average waiting time?

Average response time?

Average TAT?

SJF

- Shortest Job First (SJF) Scheduling Algorithm
- Non-Preemptive

Non Preemptive SJF

- This algorithm associates with each process the length of the process's next CPU burst time.
- When the CPU is available it is assign to the next process that has the smallest next CPU burst time.
- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

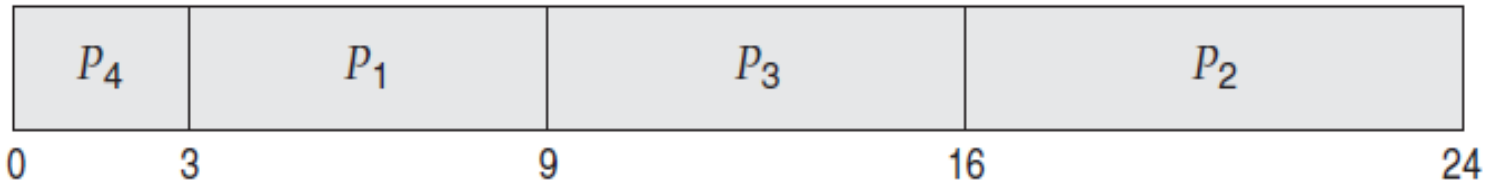
Non Preemptive SJF

- Non-Preemptive SJF Example

Non Preemptive SJF example

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:



Non Preemptive SJF Example-1

Process	Arrival time (AT)	Burst Time (BT)	Completion Time (CT) Or Exit Time	Turn Around Time (TAT) TAT=CT-AT	Waiting Time (WT) WT = TAT –BT
P1	0	6	9	9	3
P2	0	8	24	24	16
P3	0	7	16	16	9
P4	0	3	3	3	0

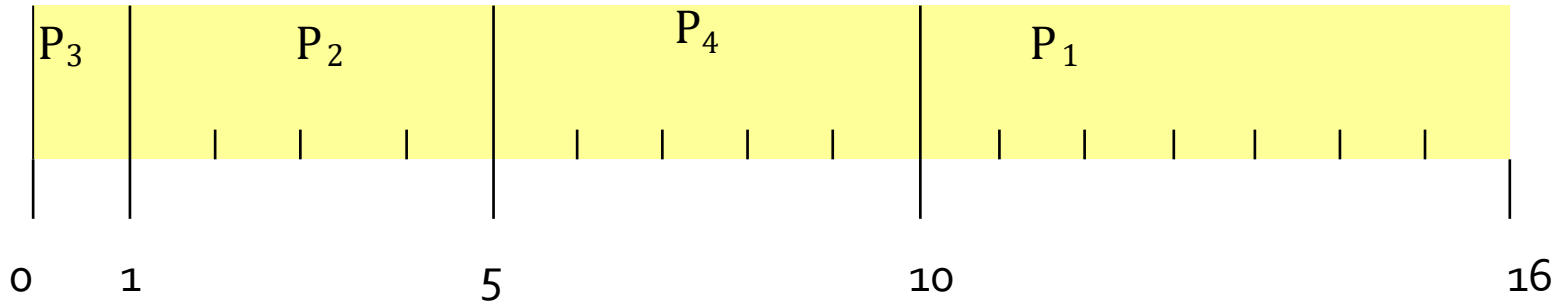
What is average waiting time? 7

What is average response time(burst time)? 7

What is average TAT? 13

SJF EXAMPLE-2

Process	Arrival Time	Burst Time
P1	0.0	6
P2	0.0	4
P3	0.0	1
P4	0.0	5



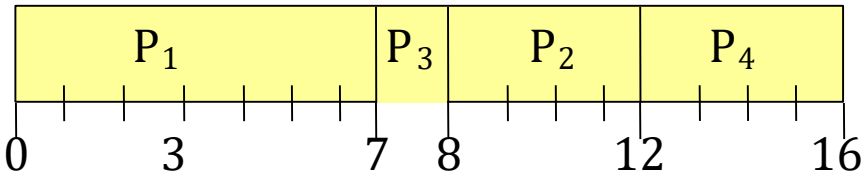
Find Avg Waiting time? Avg TAT? Avg Resp Time?

Average waiting time = (10 + 1 + 0 + 5)/4 = 4

Average turn-around time = (16+ 5 + 1 + 10)/4 = 8

Non Preemptive SJF EXAMPLE-3

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4



Find Avg Waiting time? Avg TAT? Avg Resp Time?

Average turn-around time: (CT-AT)

$= ((7 - 0) + (12 - 2) + (8 - 4) + (16 - 5))/4 = (7 + 10 + 4 + 11)/4 = 8$

Average waiting time: (TAT-BT)

$= ((7 - 7) + (10 - 4) + (4 - 1) + (11 - 4))/4 = (0 + 6 + 3 + 7)/4 = 4$

Preemptive SJF

- Shortest Remaining Time Next (SRTN) or
- Preemptive Shortest Job First (SJF) Scheduling Algorithm
- Preemptive SJF scheduling is sometimes called shortest – remaining – time – first scheduling.

Preemptive SJF

- This Algorithm is the preemptive version of SJF scheduling.
- In SRTN, the execution of the process can be stopped after certain amount of time.
- At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.
- Once all the processes are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF scheduling**.

SRTN or Preemptive SJF

Process	Arrival Time	Burst Time
P0	0	10
P1	1	6
P2	3	2
P3	5	4

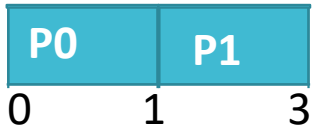
Gantt Chart

Process	Remaining Time
Po	9
P1	6

SRTN or Preemptive SJF

Process	Arrival Time	Burst Time
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Gantt Chart



Process	Remaining Time
P0	9
P1	4
P2	2

SRTN or Preemptive SJF

Process	Arrival Time	Burst Time
P0	0	10
P1	1	6
P2	3	2
P3	5	4

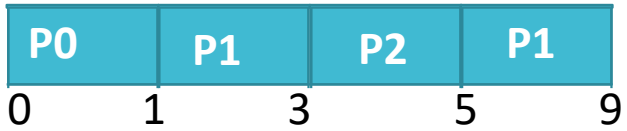
Gantt Chart

Process	Remaining Time
P0	9
P1	4
P3	4

SRTN or Preemptive SJF

Process	Arrival Time	Burst Time
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Gantt Chart



Process	Remaining Time
Po	9
P3	4

SRTN or Preemptive SJF

Process	Arrival Time	Burst Time
P0	0	10
P1	1	6
P2	3	2
P3	5	4

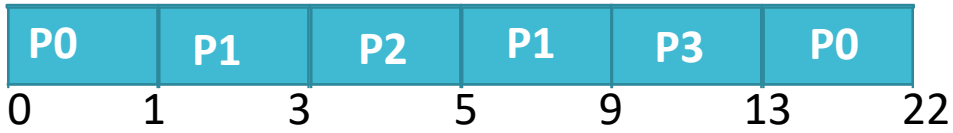
Gantt Chart

Process	Remaining Time
Po	9

SRTN 0r Preemptive SJF

Process	Arrival Time	Burst Time
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Gantt Chart



Preemptive SJF

Process	Arrival time(AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT=CT-AT	Waiting Time(WT) WT = TAT-BT
P0	0	10	22	22	12
P1	1	6	9	8	2
P2	3	2	5	2	0
P3	5	4	13	8	4

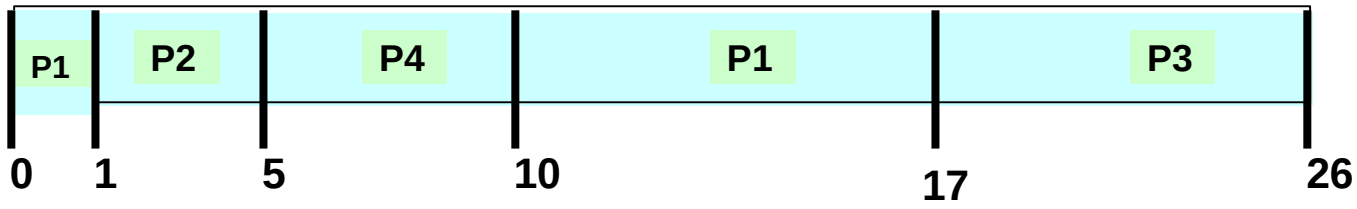
Average Turnaround Time = (22+8+2+8)/4 = 10 ms

Average Waiting Time =(12+2+0+4)/4 =4.5 ms

Preemptive SJF Example or SRTN Example

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

Gantt Chart



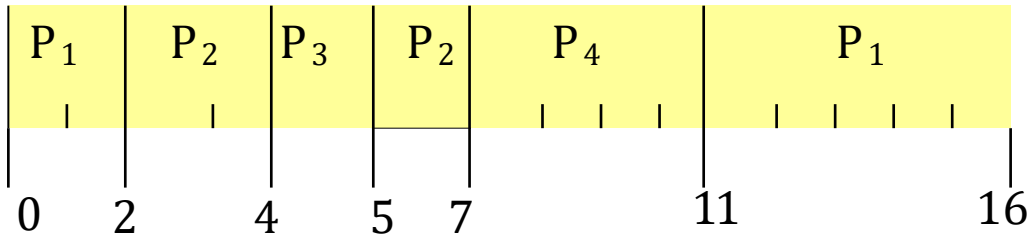
AVG TAT =(17-0)+(5-1)+(26-2)+(10-3)=52/4**=13**

Average wait = ((17-8) + (4-4) + (24-9) + (7-5))/4 = 26/4 **=6.5**

Preemptive SJF Example SRTF or SRTN Example

Process	Arrival	Time/BT
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart



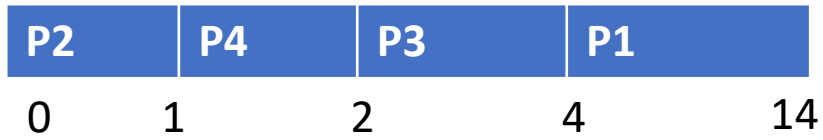
Average turn-around time = ((16-0)+(7-2)+(5-4)+ (11-5))/4 = 7

Average waiting time= ([(16– 7) + (5 - 4)] + [(1 – 1) + (6 –4)/4 = 3

Preemptive SJF Example SRTF or SRTN Example

Process	Priority	Time/BT
P1	3	10
P2	1	1
P3	4	2
P4	5	1

Gantt Chart



Average turn-around time = ?

Average waiting time= ?

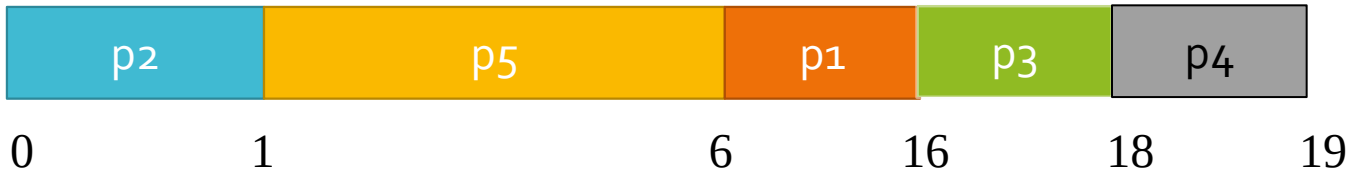
Priority Scheduling Algorithm

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order to break a tie.
- discuss scheduling in terms of ***high*** priority and ***low*** priority.
- Priorities are generally indicated by some fixed range of numbers, such as 0 to 7 or 0 to 4,095.
- However, there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority; others use low numbers for high priority.

Priority Scheduling Example Non Preemptive

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

The Gantt Chart for the schedule is:



Priority Scheduling Example Non Preemptive

Process	Priority	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT=CT-AT	Waiting Time(WT) WT = TAT- BT
P1	3	10	16	16	6
P2	1	1	1	1	0
P3	4	2	18	18	16
P4	5	1	19	19	18
P5	2	5	6	6	1

Average turn around time= 16+1+18+19+6=60/5=12

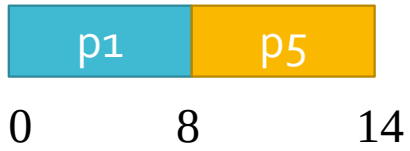
Average waiting time =(16-10)+(1-1)+(18-2)+(19-1)+(6-5) / 5=8.2

Priority Scheduling Example Non Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1

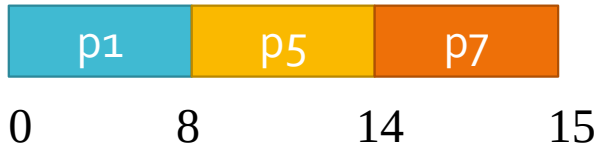
Priority Scheduling Example Non Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1



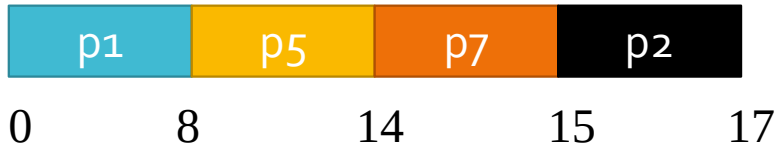
Priority Scheduling Example Non Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1



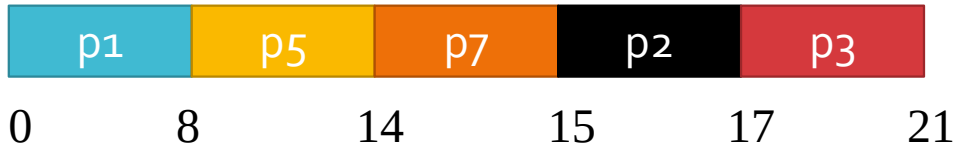
Priority Scheduling Example Non Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1



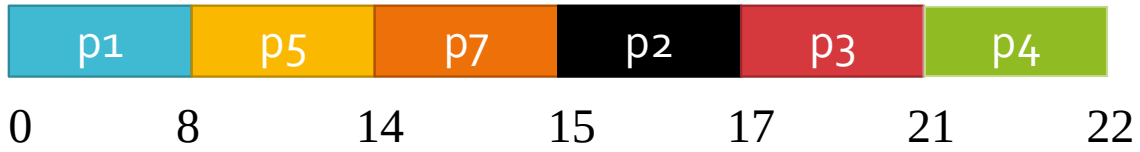
Priority Scheduling Example Non Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1



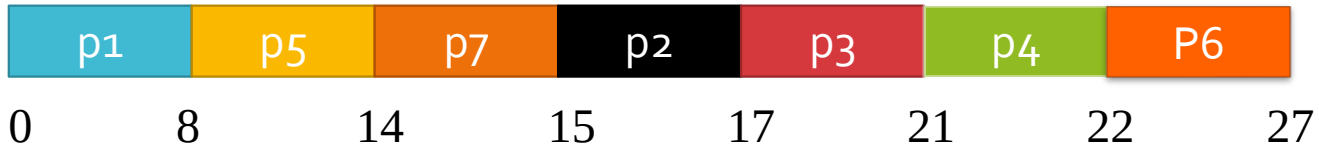
Priority Scheduling Example Non Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1



Priority Scheduling Example Non Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1



What is Average TAT?
Average Waiting Time?

Priority Scheduling Example Non Preemptive

Process	Priority	Arrival Time	Burst Time	CT	TAT	WT
P1	3	0	8	8	8	0
P2	4	1	2	17	16	14
P3	4	3	4	21	18	14
P4	5	4	1	22	18	17
P5	2	5	6	14	9	3
P6	6	6	5	27	21	16
p7	1	10	1	15	5	4

What is Average TAT? 13.5
Average Waiting Time? 9.7

Priority Scheduling Example Preemptive

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

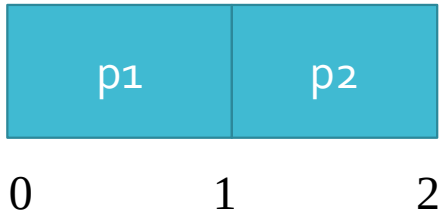
Priority Scheduling Example Preemptive

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

Process	Burst Time	Priority
P1	7	3
P2	4	1

Priority Scheduling Example Preemptive

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



Process	Burst Time	Priority
P1	7	3
P2	3	1
P3	9	4

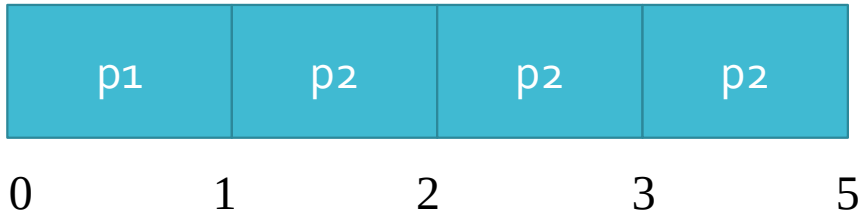
Priority Scheduling Example Preemptive

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

Process	Burst Time	Priority
P1	7	3
P2	2	1
P3	9	4
P4	5	2

Priority Scheduling Example Preemptive

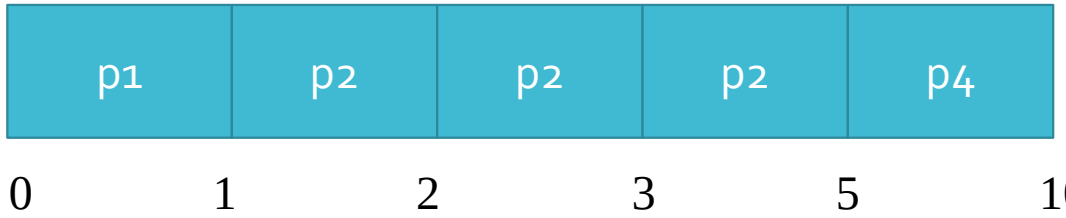
Process	Arrival Time	Burst Time	Priority
P1	0	8	3
P2	1	4	1
P3	2	9	4
P4	3	5	2



Process	Burst Time	Priority
P1	7	3
P2	2	1
P3	9	4
P4	5	2

Priority Scheduling Example Preemptive

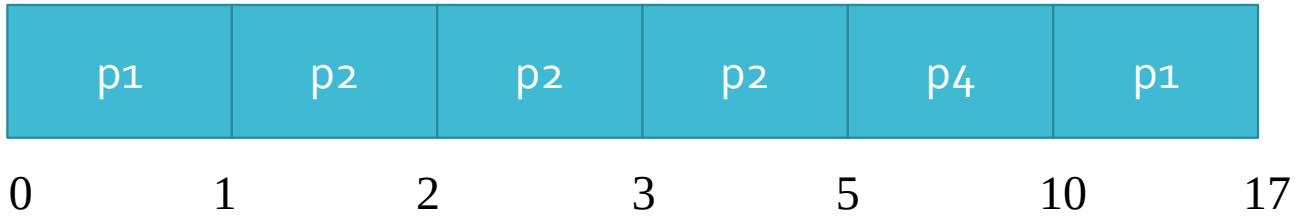
Process	Arrival Time	Burst Time	Priority
P1	0	8	3
P2	1	4	1
P3	2	9	4
P4	3	5	2



Process	Burst Time	Priority
P1	7	3
P2	2	1
P3	9	4
P4	5	2

Priority Scheduling Example Preemptive

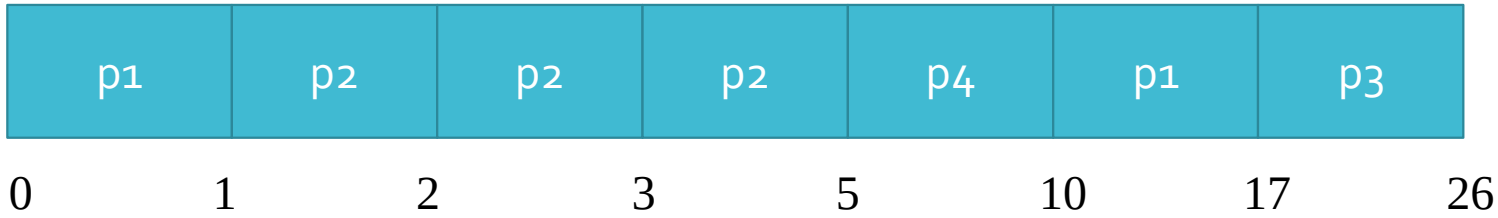
Process	Arrival Time	Burst Time	Priority
P1	0	8	3
P2	1	4	1
P3	2	9	4
P4	3	5	2



Process	Burst Time	Priority
P1	7	3
P2	2	1
P3	9	4
P4	5	2

Priority Scheduling Example Preemptive

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



Process	Burst Time	Priority
P1	7	3
P2	2	1
P3	9	4
P4	5	2

Priority Scheduling Example Preemptive

Process	Arrival Time	Priority	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT=CT-AT	Waiting Time(WT) WT= TAT-BT
P1	0	3	8	17	17	9
P2	1	1	4	5	4	0
P3	2	4	9	26	24	15
P4	3	2	5	10	7	2

Average Turnaround time =(17-0)+(5-1)+(26-2)+(10-3)=13 ms

Average waiting time = (17-8)+(4-4)+(24-9)+(7-5)=6.5 ms

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

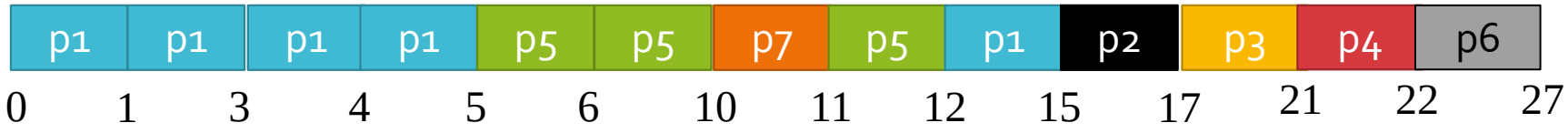
Average Response Time?

Priority Scheduling Example Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1

Priority Scheduling Example Preemptive

Process	Priority	Arrival Time	Burst Time
P1	3	0	8
P2	4	1	2
P3	4	3	4
P4	5	4	1
P5	2	5	6
P6	6	6	5
p7	1	10	1



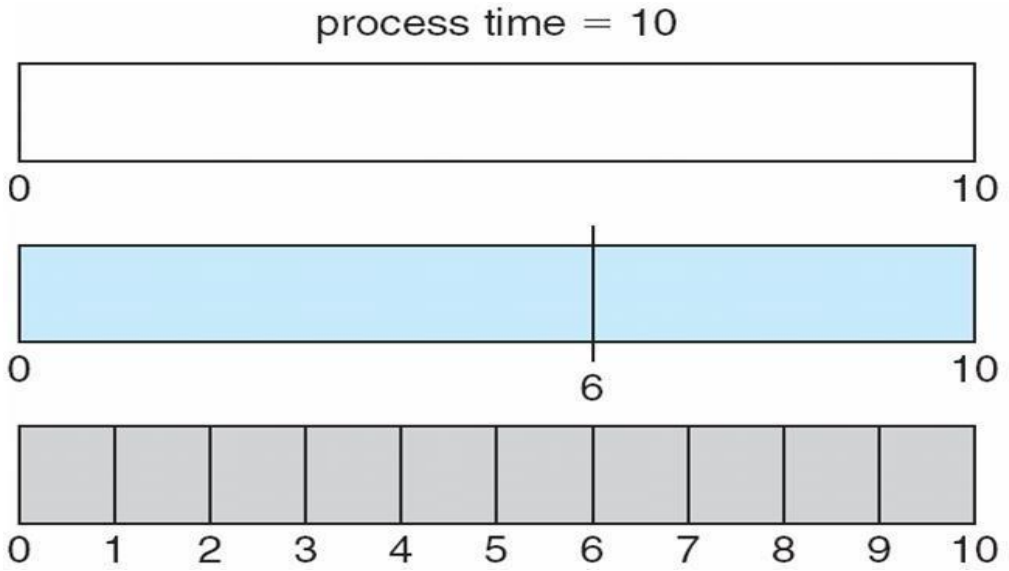
(1) Average Waiting Time? 9.8

(2) Average Turn Around Time? 13.7

(3) Average Response Time? 8.7

Round Robin Scheduling Algorithm

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.

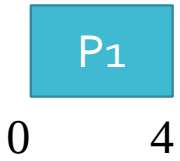


quantum	context switches
12	0
6	1
1	9

RR Example

Process	Burst Time
P1	24
P2	3
P3	3

Time Quantum is 4



Queue

P2	**3**
P3	3
P1	20

RR Example

Process	Burst Time
P1	24
P2	3
P3	3

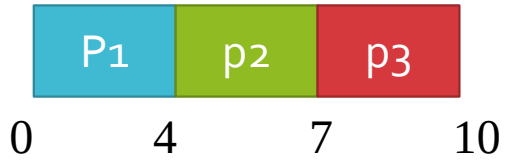
Time Quantum is 4

P2	0
P3	0
P1	20

RR Example

Process	Burst Time
P1	24
P2	3
P3	3

Time Quantum is 4



P2	**0**
P3	0
P1	20

RR Example

Process	Burst Time
P1	24
P2	3
P3	3

Time Quantum is 4

P2	**0**
P3	0
P1	16

RR Example

Process	Burst Time
P1	24
P2	3
P3	3

Time Quantum is 4

P2	**0**
P3	0
P1	12

RR Example

Process	Burst Time
P1	24
P2	3
P3	3

Time Quantum is 4

P2	**0**
P3	0
P1	8

RR Example

Process	Burst Time
P1	24
P2	3
P3	3

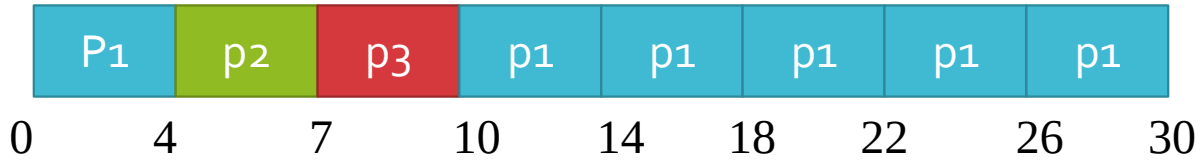
Time Quantum is 4

P2	**0**
P3	0
P1	4

RR Example

Process	Burst Time
P1	24
P2	3
P3	3

Time Quantum is 4

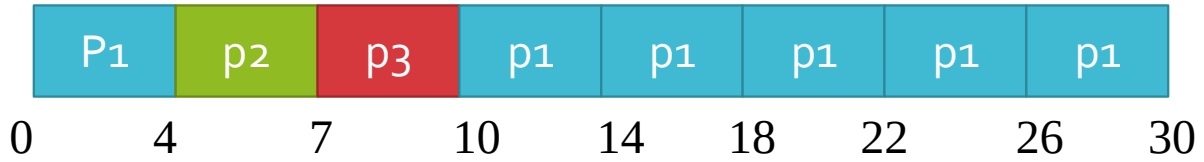


P2	**0**
P3	0
P1	0

Average Waiting Time?
Average Turn Around Time?
Average Response Time?

RR Example

Process	Arrival Time	BT	CT	TAT= CT-AT	WT=TAT-BT
P1	0	24	30	30	6
P2	0	3	7	7	4
P3	0	3	10	10	7



Average Waiting Time? 5.66
Average Turn Around Time? 15.66
Average Response Time? 3.66

RR Example

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

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

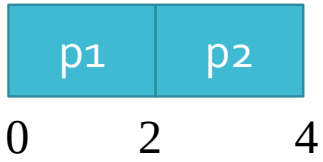
If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time.

RR Example

Here p1 runs for 2 seconds @ that time two process comes in a ready queue p2 & p3 and p1 added after that

P2	**3**
P3	1
P1	3

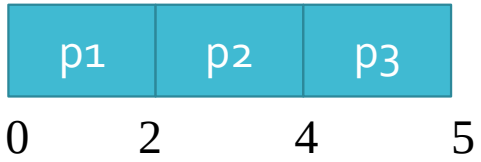
RR Example



Here after 4 seconds time two process comes in a ready queue p4 & p5 in sequence and p2 added after that

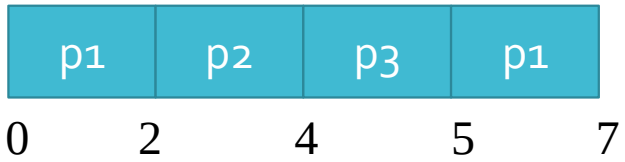
P3	**1**
P1	3
P4	2
P5	3
p2	1

RR Example



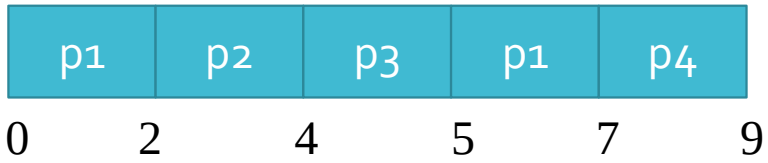
P1	**3**
P4	2
P5	3
p2	1

RR Example



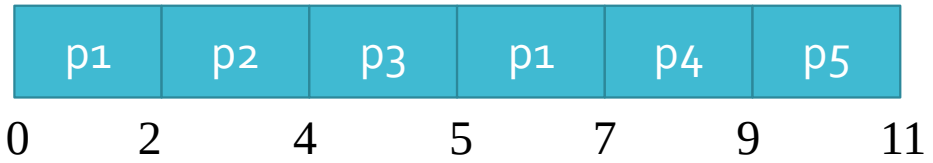
P4	**2**
P5	3
p2	1
P1	1

RR Example



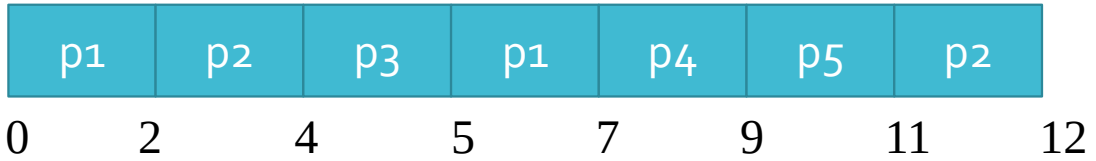
P5	**3**
p2	1
P1	1

RR Example



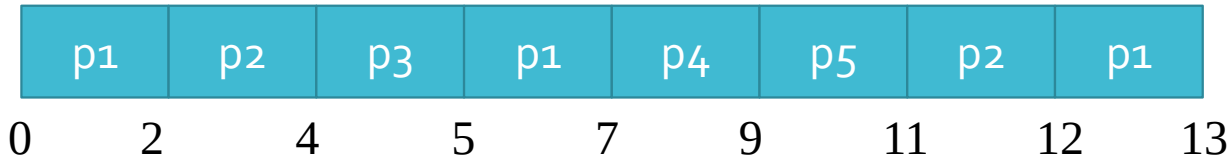
p2	**1**
P1	1
P5	1

RR Example



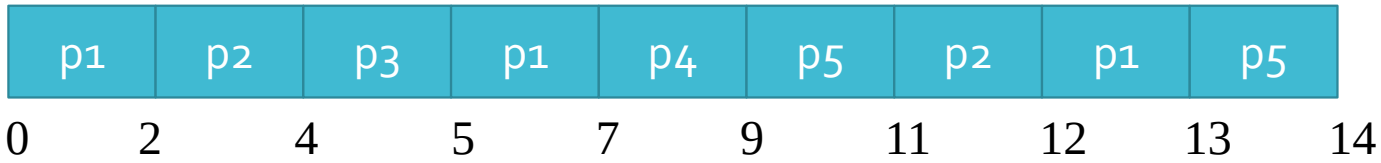
P1	**1**
P5	1

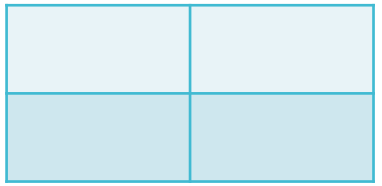
RR Example



P_5	1

RR Example





Empty

RR Example

Process Id	CT	Turn Around time	Waiting time
P1	13	13 – 0 = 13	13 – 5 = 8
P2	12	12 – 1 = 11	11 – 3 = 8
P3	5	5 – 2 = 3	3 – 1 = 2
P4	9	9 – 3 = 6	6 – 2 = 4
P5	14	14 – 4 = 10	10 – 3 = 7

- Average Turn Around time = (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6 unit
- Average waiting time = (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8 unit

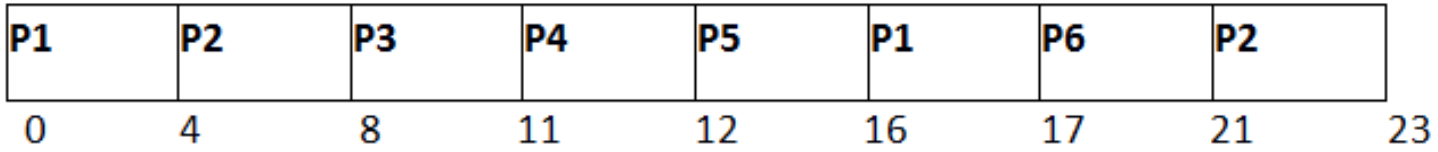
RR Example

Process ID	Arrival Time	Burst Time
1	0	5
2	1	6
3	2	3
4	3	1
5	4	5
6	6	4

Time Quantum is 4

- Average Turn Around time??
- Average waiting time??

RR Example



Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	5	17	17	12
2	1	6	23	22	16
3	2	3	11	9	6
4	3	1	12	9	8
5	4	5	24	20	15
6	6	4	21	15	11

Avg Waiting Time = (12+16+6+8+15+11)/6 = 76/6 units

Thank you....