# Unit-II Process Management

**Process:** Concept of a Process, Process States, Process Description, Process Control Block, Operations on Processes.

**Threads:** Definition and Types, Concept of Multithreading, Multi core Processors and Threads.

**Scheduling:** Types of Scheduling: Preemptive and Non-preemptive, Scheduling Algorithms and their Performance Evaluation: FCFS, SJF, SRTN, Priority Based, Round Robin, Introduction to Thread Scheduling.

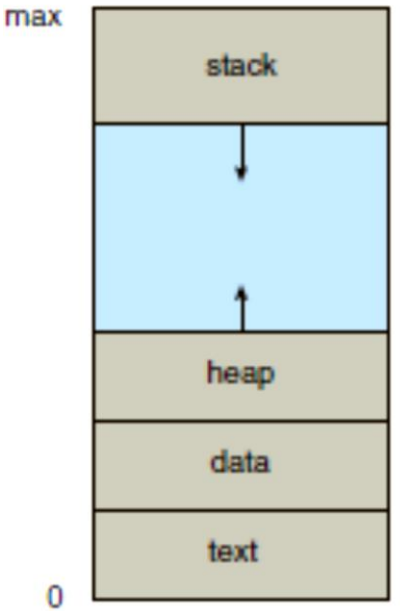
**CO 2:Compare and evaluate process scheduling algorithms and IPC**

## ❖ Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-sharing systems – user programs or tasks
- **Program**: a set of functions or modules
- **Job**: sequence of job steps where each job step is comprised of execution of one program
- **Process**: an execution of a program with allocated resources

**A process includes**:

- **Program Code**: which is known as the **text section**
- **Program Counter**: current activity
- **Stack:** contains temporary data called during execution (function parameters, return address, local variables)
- **Data Section:** contains global variables
- **Heap:** memory that is dynamically allocated during process time



## ❖ OS View of Process/Process State

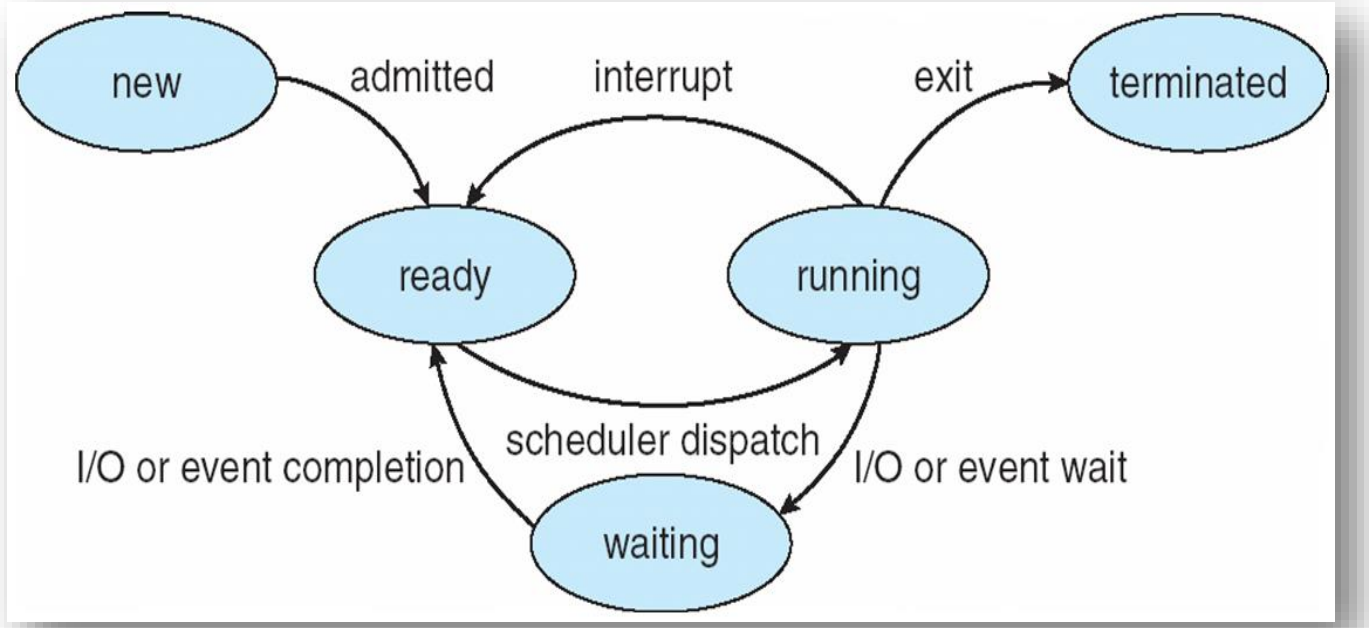
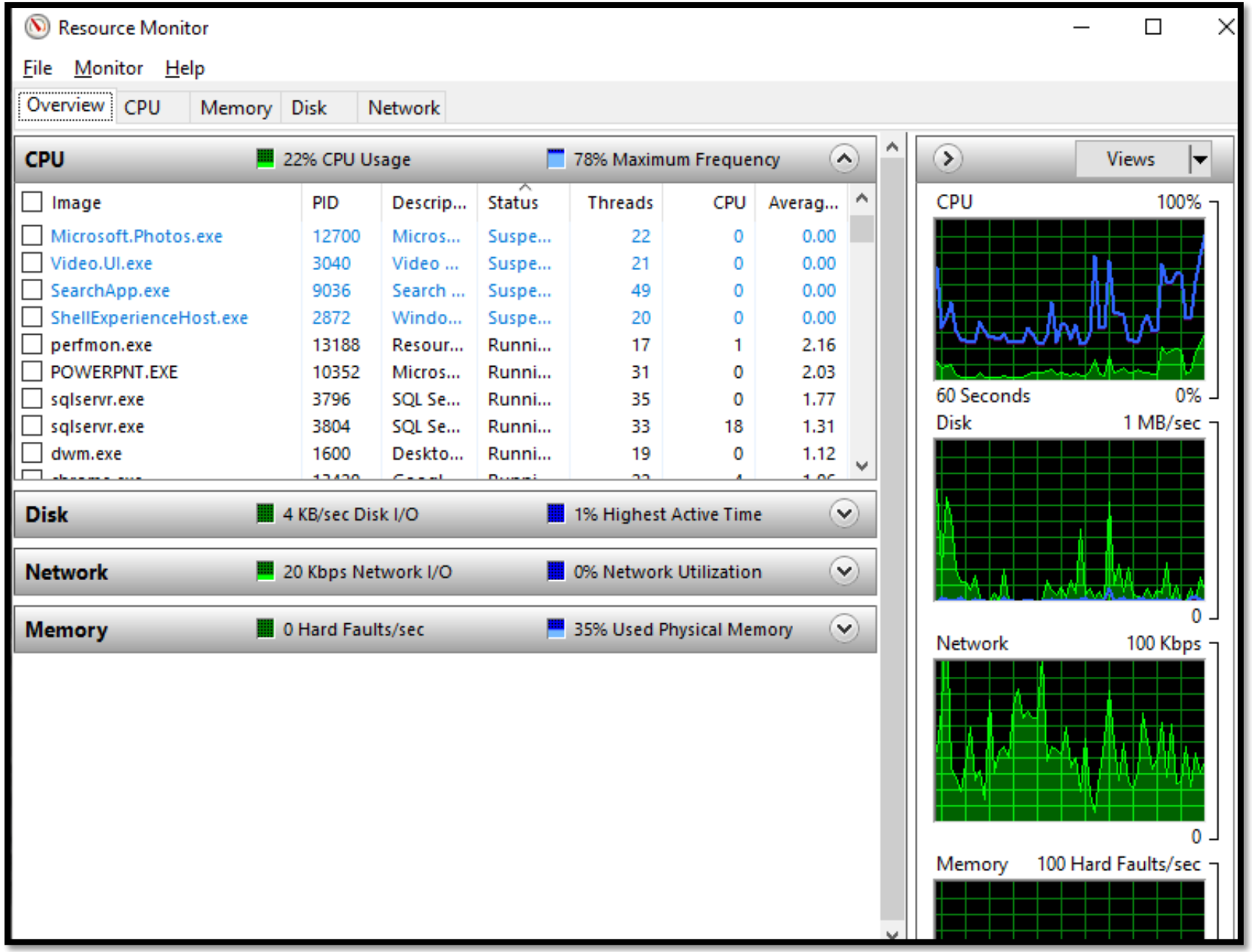



**Figure:** Process State Transition Diagram

As a process executes, it changes *state:*

1. **New :** The process is being created.
2. **Ready:** The process is waiting to be assigned to a processor.
3. **Running:** Process instructions are being executed
4. **Waiting:** The process is waiting for some event to occur (such as the completion of an I/O operation).
5. **Terminated:** The process has finished execution.

**Continued….**


Resource Monitor
File Monitor Help
Overview CPU Memory Disk Network
CPU 22% CPU Usage 78% Maximum Frequency
Image PID Descrip... Status Threads CPU Averag...
Microsoft.Photos.exe 12700 Micros... Suspe... 22 0 0.00
Video.UI.exe 3040 Video ... Suspe... 21 0 0.00
SearchApp.exe 9036 Search ... Suspe... 49 0 0.00
ShellExperienceHost.exe 2872 Windo... Suspe... 20 0 0.00
perfmon.exe 13188 Resour... Runni... 17 1 2.16
POWERPNT.EXE 10352 Micros... Runni... 31 0 2.03
sqlservr.exe 3796 SQL Se... Runni... 35 0 1.77
sqlservr.exe 3804 SQL Se... Runni... 33 18 1.31
dwm.exe 1600 Deskto... Runni... 19 0 1.12
Disk 4 KB/sec Disk I/O 1% Highest Active Time
Network 20 Kbps Network I/O 0% Network Utilization
Memory 0 Hard Faults/sec 35% Used Physical Memory
Views
CPU 100%
60 Seconds 0%
Disk 1 MB/sec
0
Network 100 Kbps
0
Memory 100 Hard Faults/sec

# ❖ Process Description

- The **OS controls events** within the computer system.
- OS **schedules and dispatches** processes for execution by the processor, **allocates** resources to processes and **responds** to requests by user processes for basic services.
- Fundamentally, **OS manages the use of system resources by processes**.

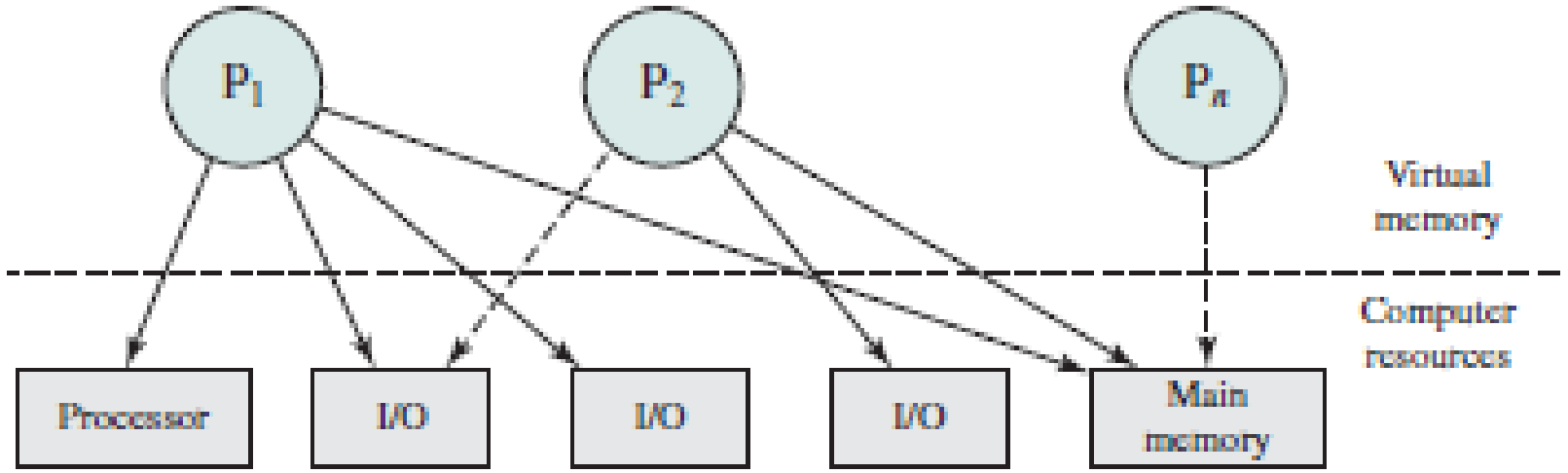



**Figure:** Processes and Resources (resource allocation at one snapshot in time)

**Continued….**

- As shown in above figure, in a multiprogramming environment, there are number of processes ($P_1$ ,…, $P_n$ ) that have been created and exist in virtual memory.
- Each process, during the course of its execution, needs access to certain system resources, including the processor, I/O devices and main memory.
- In the figure, **process $P_1$** is **running**; at least part of the process is in main memory and it has control of two I/O devices.
- **Process $P_2$** is also **in main memory but is blocked** waiting for an I/O device allocated to $P_1$.
- **Process $P_n$** has been **swapped out** and is therefore **suspended**.

## ❖ Operating System Control Structures

- To manage processes and resources, the OS must have **information about the current status of each process and resource.**
- For this, the **OS constructs and maintains tables of information** about each entity that it is managing.
- **Four different types of tables** maintained by the OS:
  - memory
  - I/O
  - file
  - process

**Continued….**

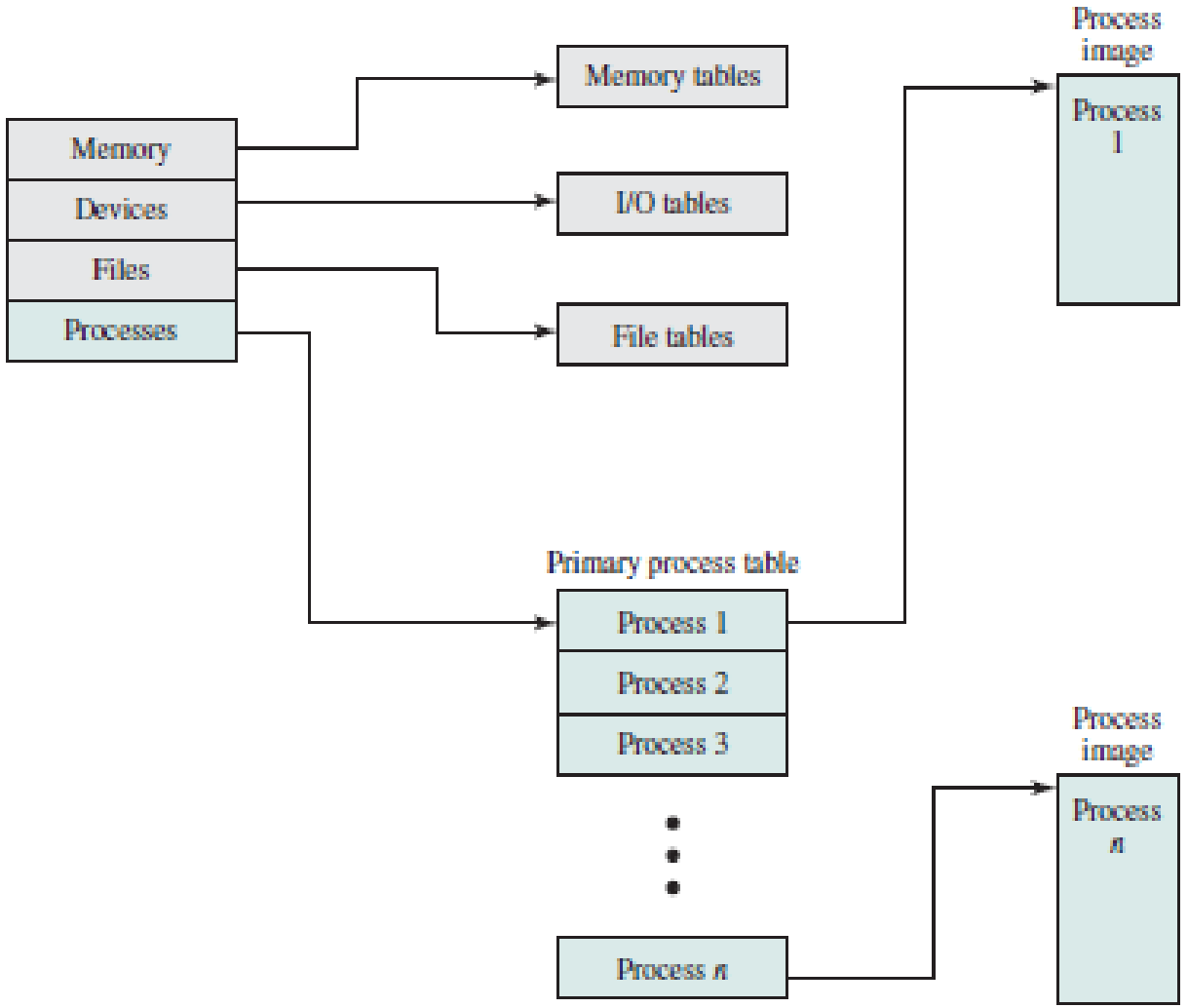



**Figure:** General Structure of Operating System Control Tables

**Continued….**

1. **Memory tables** are used to keep track of both main (real) and secondary (virtual) memory.
2. **I/O tables** are used by the OS to manage the I/O devices and channels of the computer system.
3. **File tables** provide information about the existence of files, their location on secondary memory, their current status and other attributes.
4. The OS must maintain **process tables** to manage processes:
   - There must be some reference to Memory, I/O and files directly or indirectly in the process tables as these resources are managed on behalf of processes.

## ❖ Process Control Structures

**To manage and control a process**, the OS must know:

1. **where** the process is located;

2. the **attributes of the process** that are necessary for its management.

**Continued….**

## 1. Process Location:

Each process has **collection of program, data, stack and attributes** associated with it which is referred as the **process image**

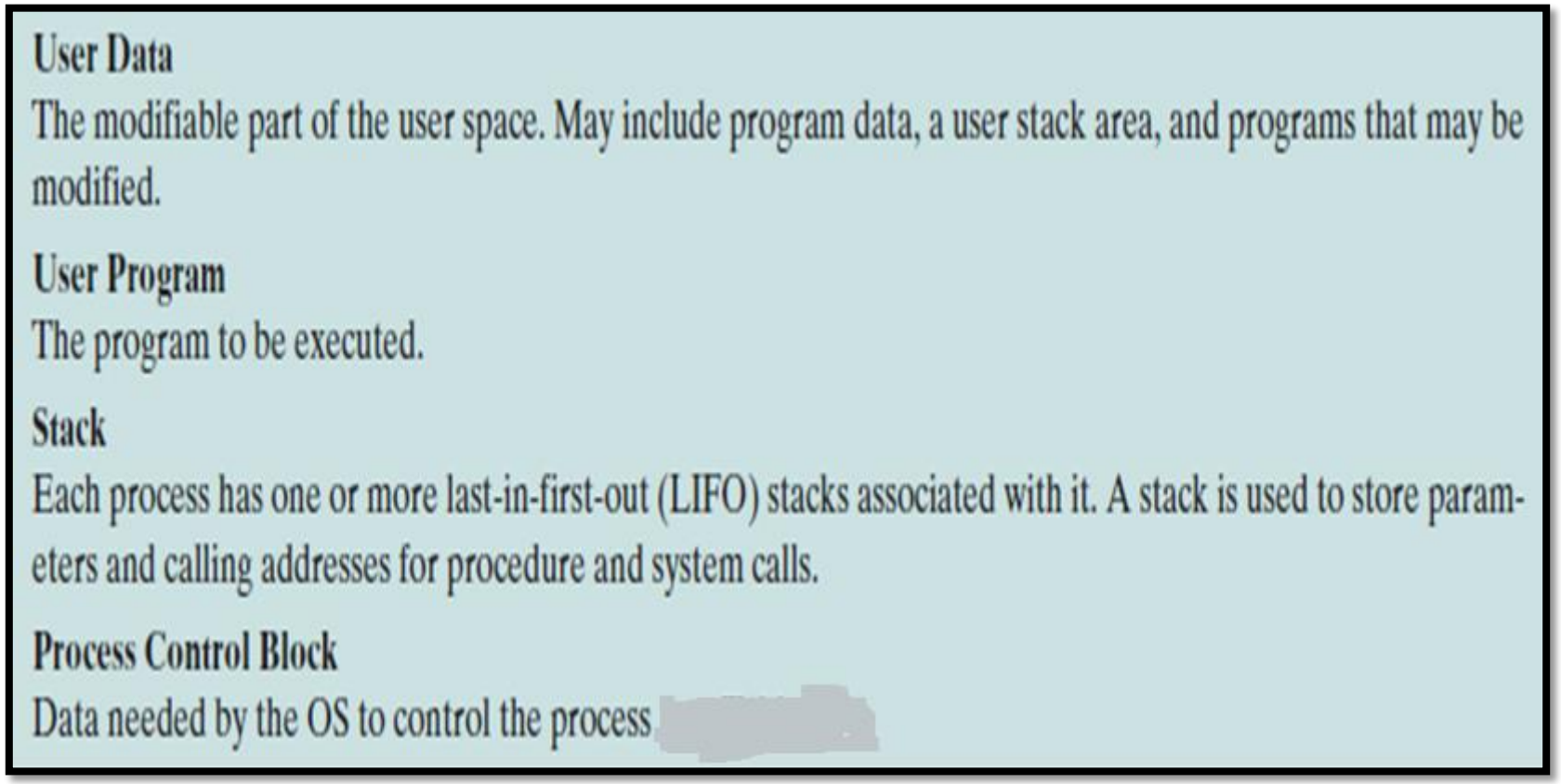
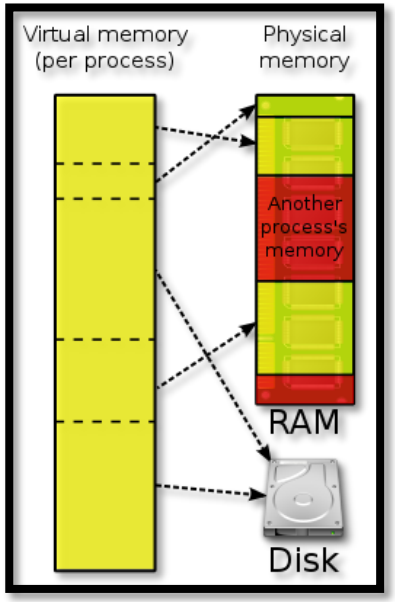



**Figure:** Typical Elements of a Process Image

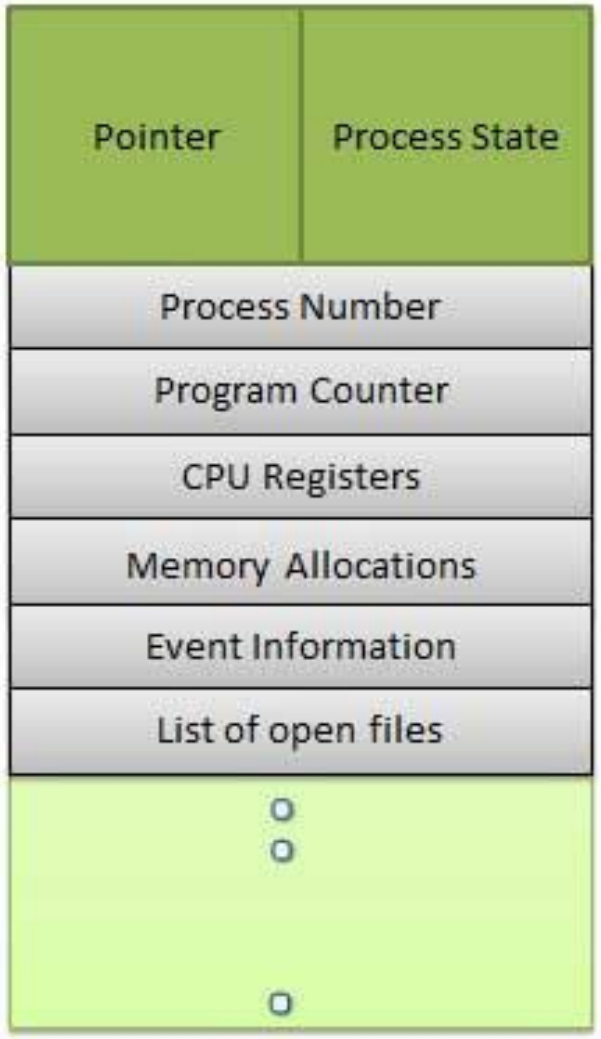
**Continued….**

- The process image is maintained as a **contiguous block of memory**.
- This block is maintained **in secondary memory**, usually disk.
- Thus the OS can manage the process, at least **a small portion of its image must be maintained in main memory.**
- To execute the process, **the entire process image must be loaded into main memory** or **at least virtual memory**.
- Thus, the **OS needs to know the location of each process on disk** and the location of that **process in main memory**.
- The **process tables** maintained by the OS must show the location of each page of each process image.




**Continued….**

- **Process Control Block (PCB)**
  - A process is represented in OS by a PCB.
  - It is a **data structure** to control the process progress.




**Continued….**

- **Information associated with each process:**

- **Process state:** the state may be new, ready, running, waiting, halted.
- **Program counter:** indicates the **address of the next instruction** to be executed for the process.
- **CPU registers:** vary in number and type, depending on the **computer architecture**.
- **CPU scheduling information:** a process priority, pointers to scheduling queues and any other **scheduling parameters**.
- **Memory-management information:** include such items as the **value of the base and limit registers** and the **page tables** or the **segment tables**.
- **Accounting information:** the amount of CPU and real time used, time limits, job or process numbers etc.
- **I/O status information:** list of I/O devices allocated to the process, a list of open files and so on.

## 2. Process Attributes

- A sophisticated multiprogramming system **requires** a great deal of **information about each process**.
- This information can be considered to **reside in a process control block**.
- We can group the **process control block information** into **three general categories:**
  - Process identification - Process id
  - Processor state information - different registers & stack pointers
  - Process control information – all information of a process (scheduling, PCB, memory management etc)

**Continued….**

## Process Identification

**Identifiers**

Numeric identifiers that may be stored with the process control block include

- Identifier of this process
- Identifier of the process that created this process (parent process)
- User identifier

## Processor State Information

**User-Visible Registers**

A user-visible register is one that may be referenced by means of the machine language that the processor executes while in user mode. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

**Control and Status Registers**

These are a variety of processor registers that are employed to control the operation of the processor. These include

- **Program counter:** Contains the address of the next instruction to be fetched
- **Condition codes:** Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
- **Status information:** Includes interrupt enabled/disabled flags, execution mode

**Stack Pointers**

Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.

**Continued….**

**Process Control Information**

**Scheduling and State Information**
This is information that is needed by the operating system to perform its scheduling function. Typical items of information:

- **Process state:** Defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted).
- **Priority:** One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable).
- **Scheduling-related information:** This will depend on the scheduling algorithm used. Examples are the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running.
- **Event:** Identity of event the process is awaiting before it can be resumed.

**Data Structuring**
A process may be linked to other process in a queue, ring, or some other structure. For example, all processes in a waiting state for a particular priority level may be linked in a queue. A process may exhibit a parent–child (creator–created) relationship with another process. The process control block may contain pointers to other processes to support these structures.

**Interprocess Communication**
Various flags, signals, and messages may be associated with communication between two independent processes. Some or all of this information may be maintained in the process control block.

**Process Privileges**
Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.

**Memory Management**
This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

**Resource Ownership and Utilization**
Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.

# ❖ Context Switching

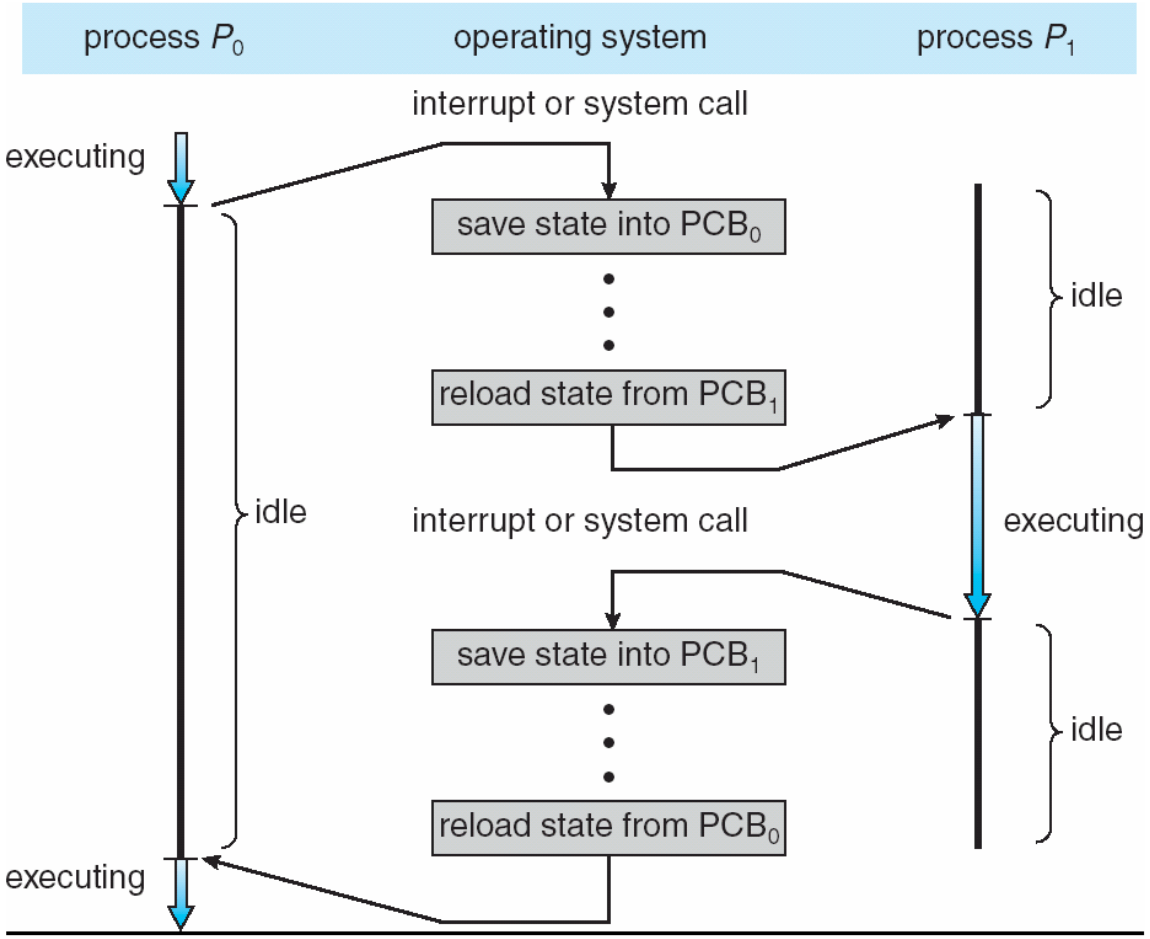



**Figure:** Context Switching

## ❖ Operations on Process

- **Process Creation:**

- **Parent** process creates **children** processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via **a process identifier** (**pid**)
- **Resource sharing possibilities**:
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- **Execution**:
  - Parent and children execute concurrently
  - Parent waits until children terminate

**Continued….**

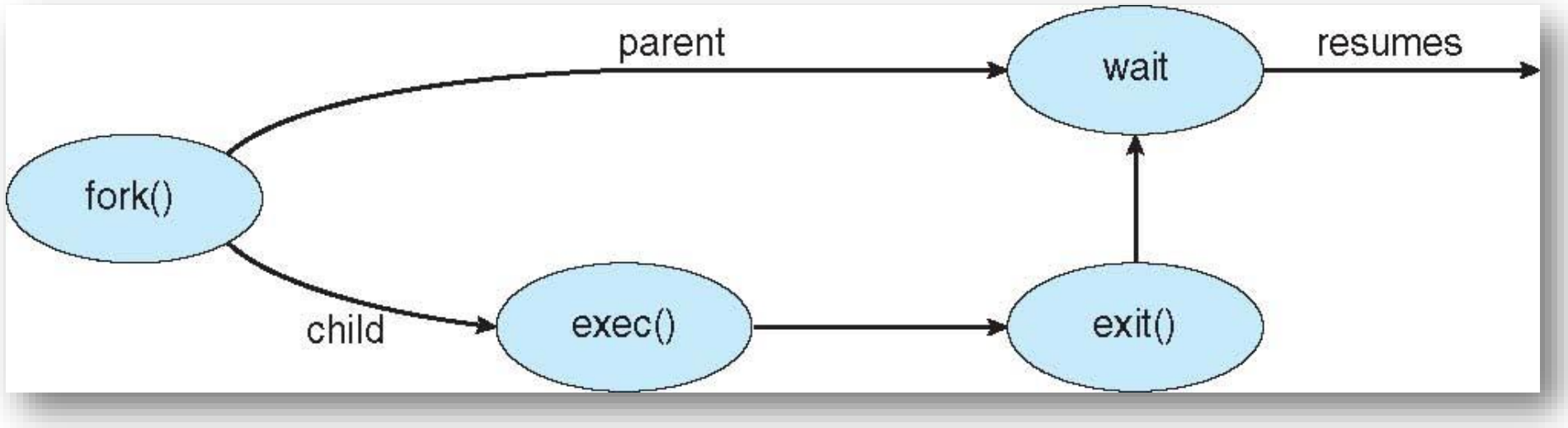



**Figure:** Process Creation

- **Process Termination** → **Normal**, **Abnormal**

- Process executes last statement and asks the OS to delete it (**exit**):
  - Output data from child to parent (via **wait**)
  - Process's resources are de-allocated by OS
- Parent may terminate execution of children processes (**abort**):
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting:
    - Some OS does not allow child to continue if its parent terminates
      - All children terminated - **cascading termination**

## ❖ Threads

- Context switch between the processes is an expensive operation; It leads to high overhead.
- A **thread** is alternative model for execution of a program that has **smaller overhead** while switching between threads of the same application.
    - Process - a single **thread** of execution (only one task at a time. eg: a word-processor program )

**Continued….**

- **What is Thread?**
  - It is a **basic unit of CPU utilization** which consists of a program counter, a register set & a stack space.
  - It **shares** with other threads belonging to the same process its **code section, data section and other operating-system resources** such as open files and signals.
  - Threads, also called as **lightweight processes** (LWPs) are **independently scheduled** parts of a single program.
  - Improves application performance through **parallelism**.
  - Each thread **belongs to exactly one process**.
  - Represents a **separate flow of control**.

**Continued….**

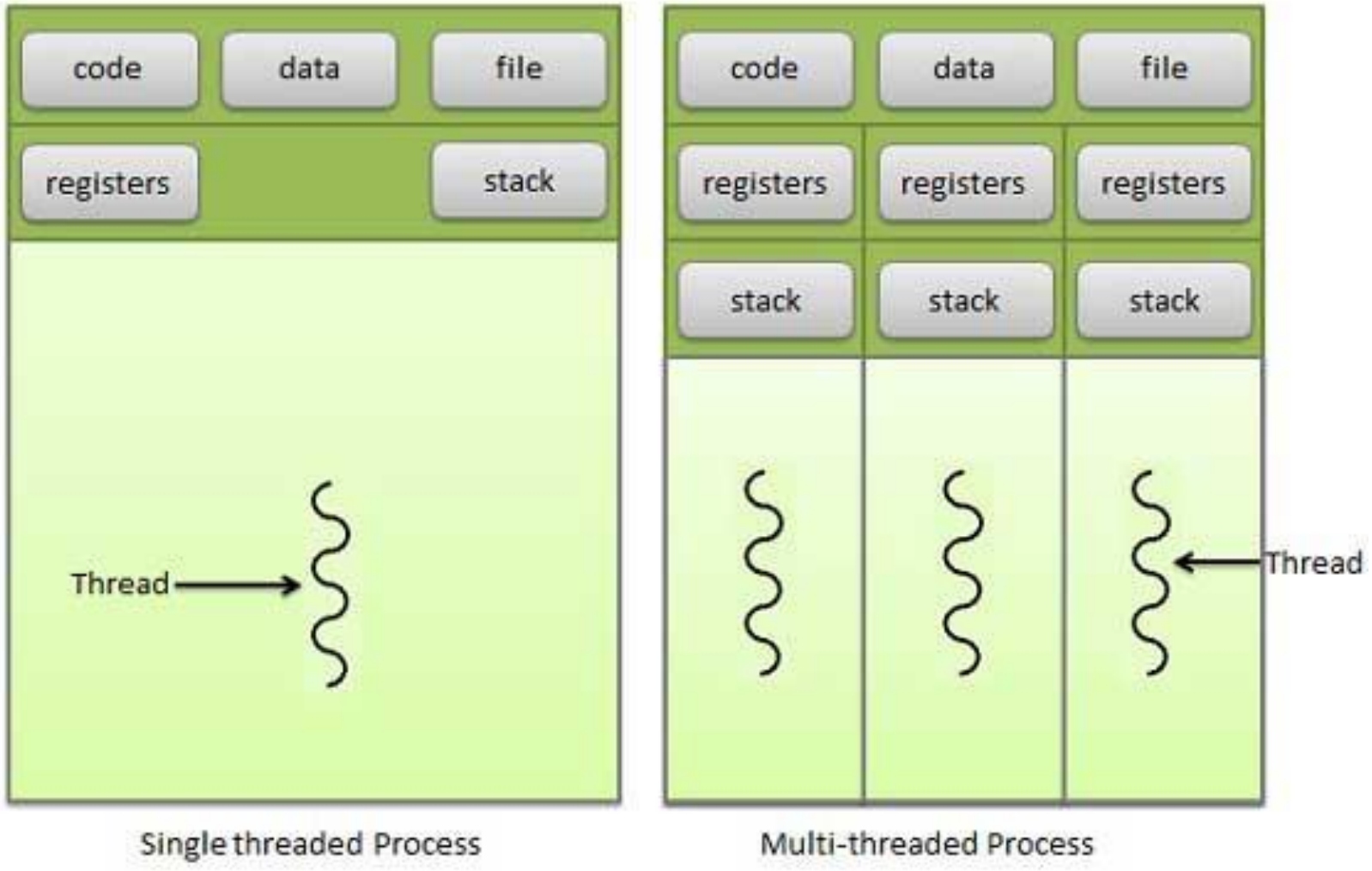



**Figure:** Single-threaded and multithreaded processes

- **Difference between Process & Thread**

| Process | Thread |
|---|---|
| Process means any **program is in execution** which **takes more time** to create & terminate. | Thread means **a segment of a process** which **takes less time** to create & terminate. |
| Process is **heavy weight** or resource intensive. | Thread is **light weight** taking lesser resources than a process. |
| **Process switching** needs interaction with operating system. | **Thread switching** does not need to interact with operating system. |
| In multiple processing environments, each process has its **own memory and file resources**. | All threads can **share** same set of open files, child processes. |
| If **one process is blocked** then it **will not affect** the execution of other processes | If a **user-level thread is blocked**, then **all other user-level threads** are blocked. |
| In multiple processes, each process **operates independently of the others.** | One thread **can read, write or change** another thread's data. |

- **Multithreaded Programming**

- Multithreading refers to the **ability of an OS to support** multiple, concurrent paths of execution within a single process.
- We say that a task is ***multithreaded*** if it is composed of several independent sub-processes which do work on **common data, share same address space, share same resources and they have identical context.**
- Most modern operating systems have extended the process concept to allow a process to have **multiple threads of execution** and thus to perform **more than one task at a time**.

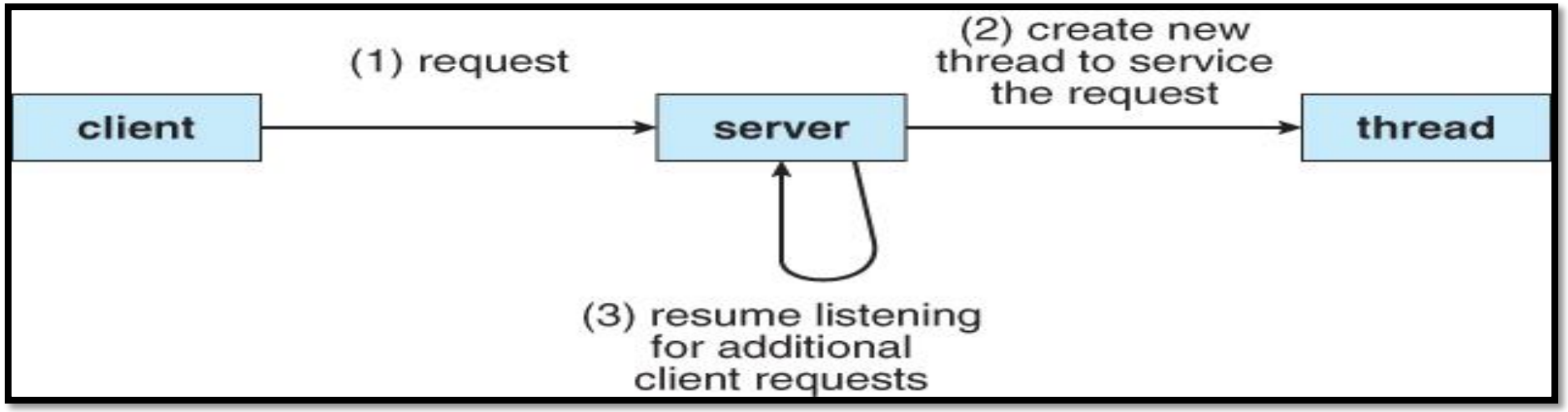
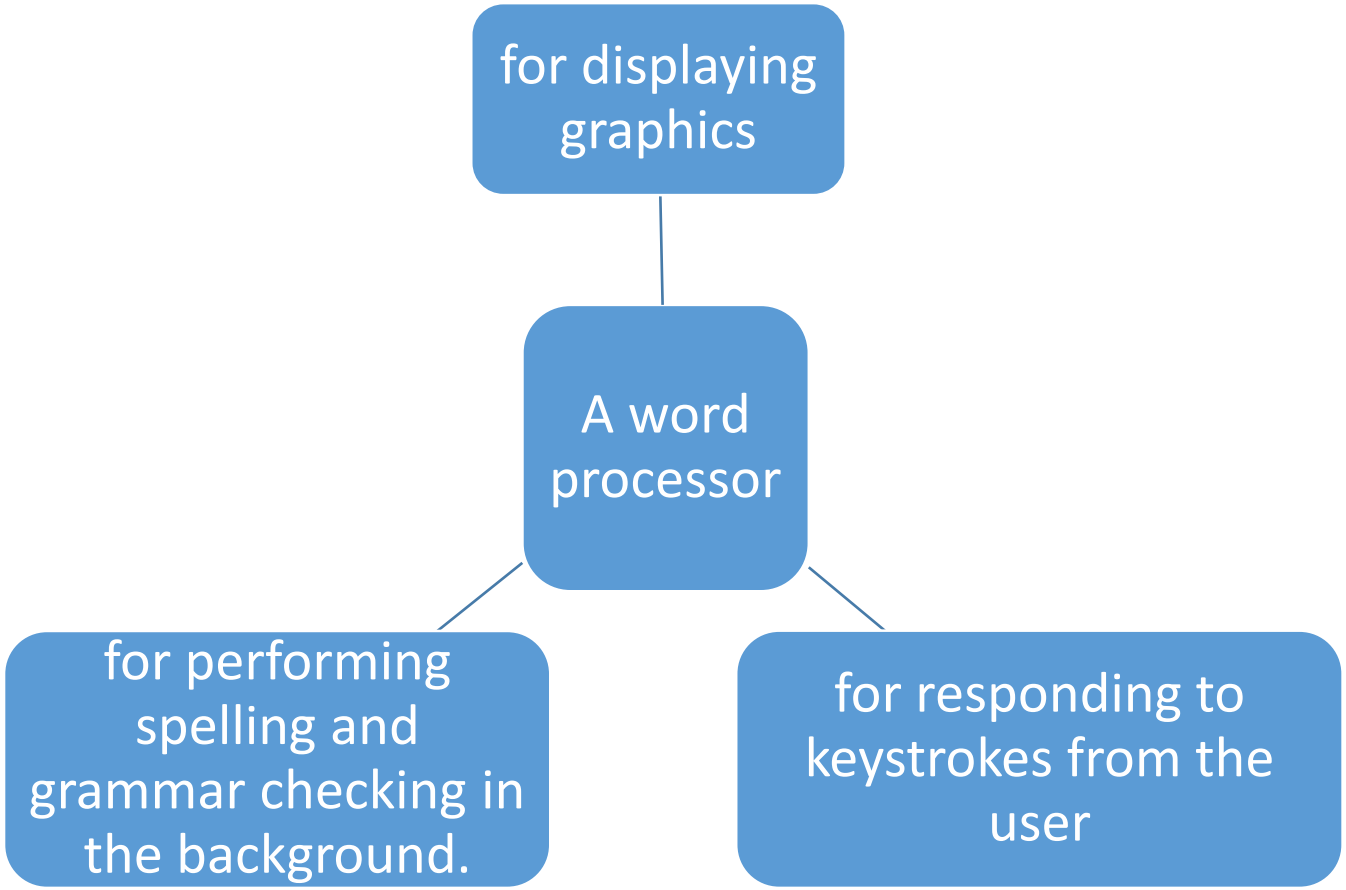



**Figure**: Multithreaded server architecture




**Multithreading Example:** Word Processor

- **Benefits of Multithreaded Programming**

- **Responsiveness:** may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby **increasing responsiveness to the user.**

- **Resource Sharing:** threads **share the memory and the resources** of the process to which they belong by default. The benefit of sharing code and data is that, it allows an application to have several different threads of activity **within the same address space.**

- **Economy:** as threads share the resources of the process to which they belong, it is more economical to create and context-switch threads.

  In general it is significantly more **time consuming** to create and manage processes than threads.

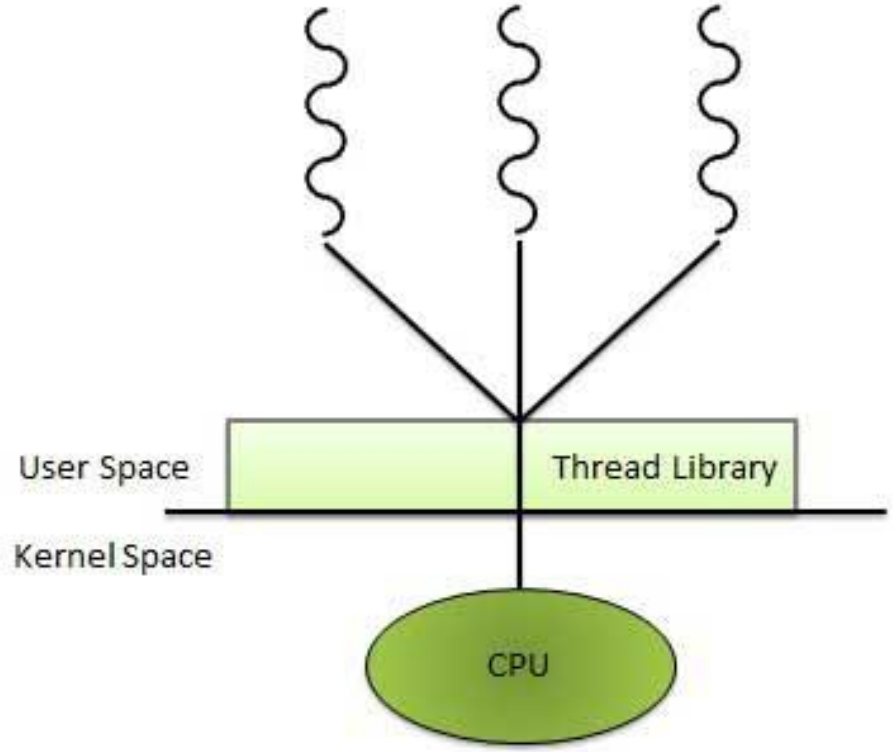
- **Scalability:** the benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores.

## Types of Thread

Threads are implemented in following two ways:

- **User Level Threads** - User managed threads
- **Kernel Level Threads** - Operating System managed threads acting on kernel

- **User Level Threads:**
  - Application handles thread management.
  - Thread Library : used to create and destroy thread, scheduling, message passing etc.



- **Kernel Level Threads:**
  - **Thread management** done **by the Kernel**.
  - Kernel threads are **supported directly by OS**.
  - **Scheduling by the Kernel** is done on a thread basis.
  - Kernel performs thread creation, scheduling and management **in Kernel space**.
  - Kernel threads are generally **slower to create and manage**.

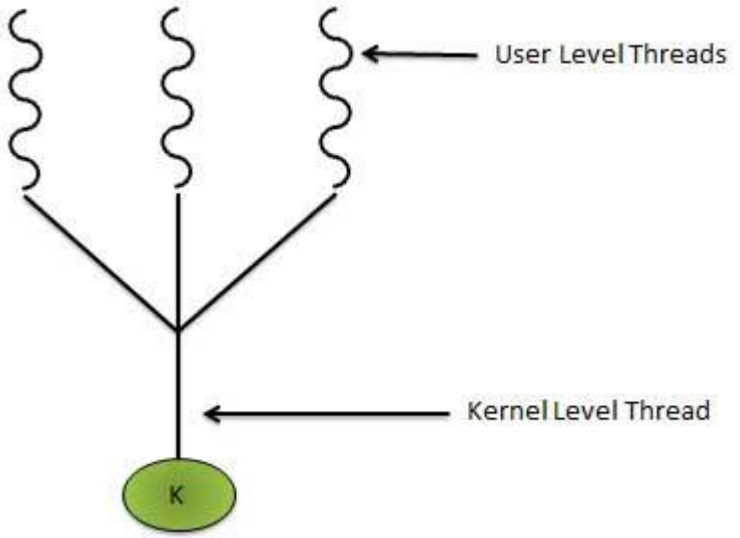
- **Difference between User Level thread and Kernel Level thread**

| User Level Threads | Kernel Level Thread |
|---|---|
| User threads are **implemented by users**. | kernel threads are **implemented by OS**. |
| **OS doesn't recognize** user level threads. | Kernel threads are **recognized by OS.** |
| **Implementation** of User threads is **easy.** | **Implementation** of Kernel thread is **complicated**. |
| Context switch time is **less.** | Context switch time is **more.** |
| Context switch requires **no hardware support.** | **Hardware support is needed.** |
| If one user level thread perform blocking operation then **entire process will be blocked.** | If one kernel thread perform blocking operation then another thread can **continue execution.** |
| User level threads are designed as **dependent threads.** | Kernel level threads are designed as **independent threads.** |
| Example : Java thread | Example : Windows |
| | |

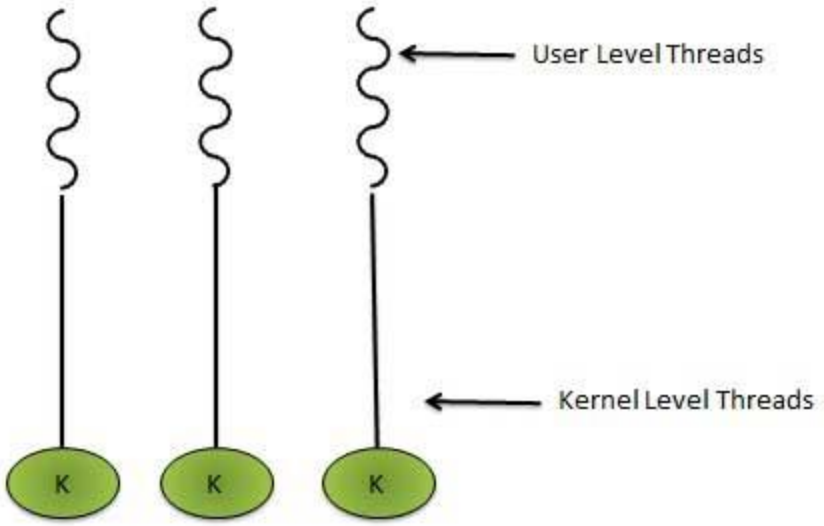
## Multithreading Models

- Many to one relationship
- One to one relationship
- Many to many relationship

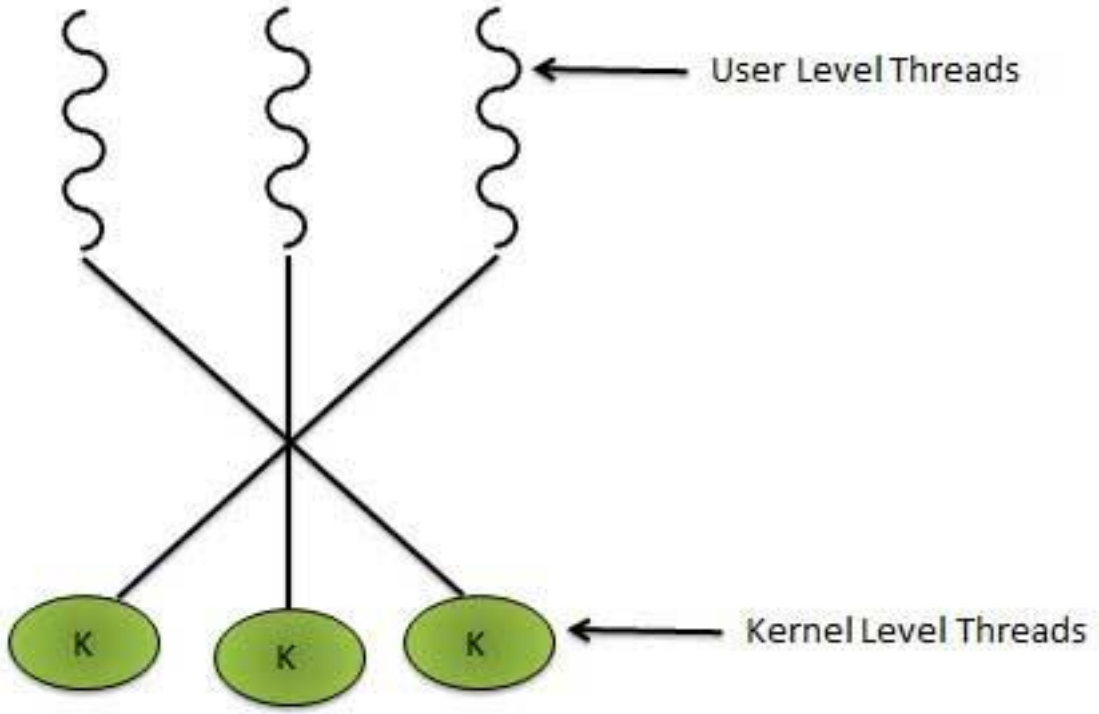
- **Many to one Model**
  - Many user level threads to one Kernel level thread.
  - Only one thread can access the Kernel at a time
  - Unable to run in parallel on multiprocessors.
  - When thread makes a blocking system call, the entire process will be blocked.



- **One to one Model**
  - One to one relationship between user level thread and kernel level thread
  - Provides **more concurrency** than the many to one model
  - Multiple thread executes in parallel on multiprocessors
  - **Disadvantage** : creating user thread requires the corresponding Kernel thread

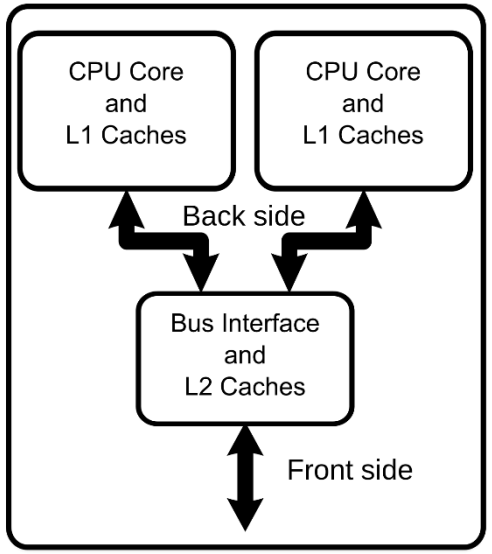


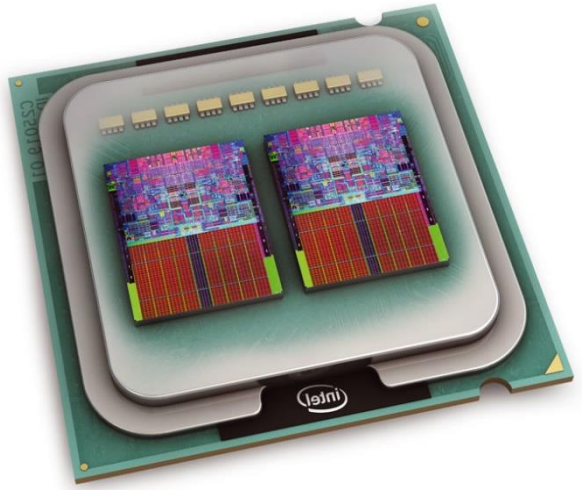
- **Many to one Model**
  - Many user level threads multiplexes to the Kernel thread of smaller or equal numbers
  - No. of Kernel threads may be specific to either a particular application or a particular machine



## ❖ Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- **Multithreaded programming** provides a mechanism for more efficient use of these multiple computing cores and improved concurrency.



## Multithreaded Multicore System

- On a system with multiple cores, however, **concurrency means that the threads can run in parallel**, because the system can assign a **separate thread to each core.**
- A system is **parallel** if it can perform more than one task simultaneously.
- In contrast, a **concurrent** system supports more than one task by allowing all the tasks to make progress.

- Thus, **it is possible to have concurrency without parallelism**.

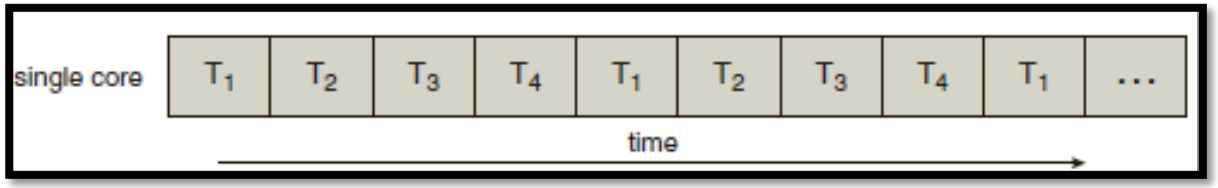



**Figure:** Concurrent execution on a single-core system

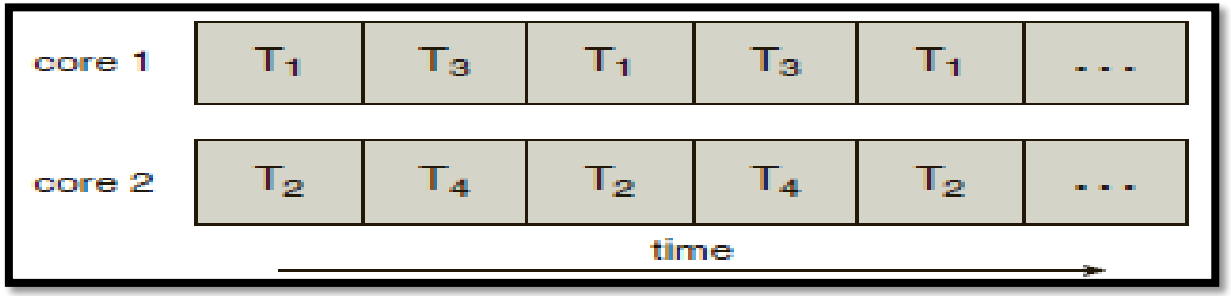



**Figure:** Parallel execution on a multicore system

# Process Scheduling

## ❖ Process Scheduling

- The **objective of multiprogramming** is to have some process running at all times, to **maximize CPU utilization**.

- The **objective of time sharing** is to switch the CPU among processes so frequently that users can interact with each program while it is running.

- To meet these objectives, the process scheduler selects an available process from a set of several available processes or program execution on the CPU.

- The process scheduler 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
    - A ready-queue **header contains pointers to the first and final PCBs in the list**.
  - **Device queues:**
    - Set of processes waiting for an I/O device
    - Each device has its **own device queue**
  - Processes migrate among the various queues for program execution on the CPU.

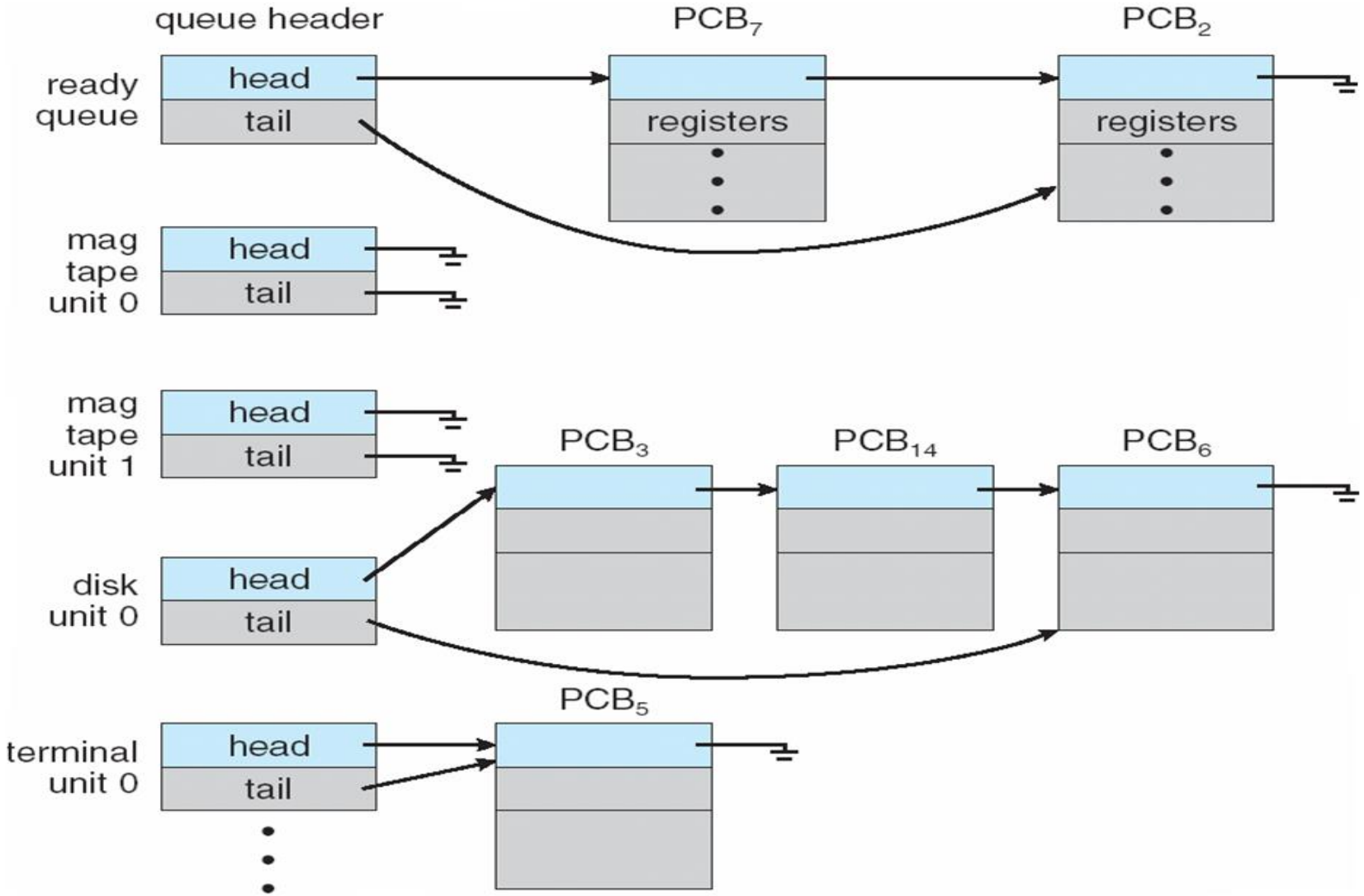
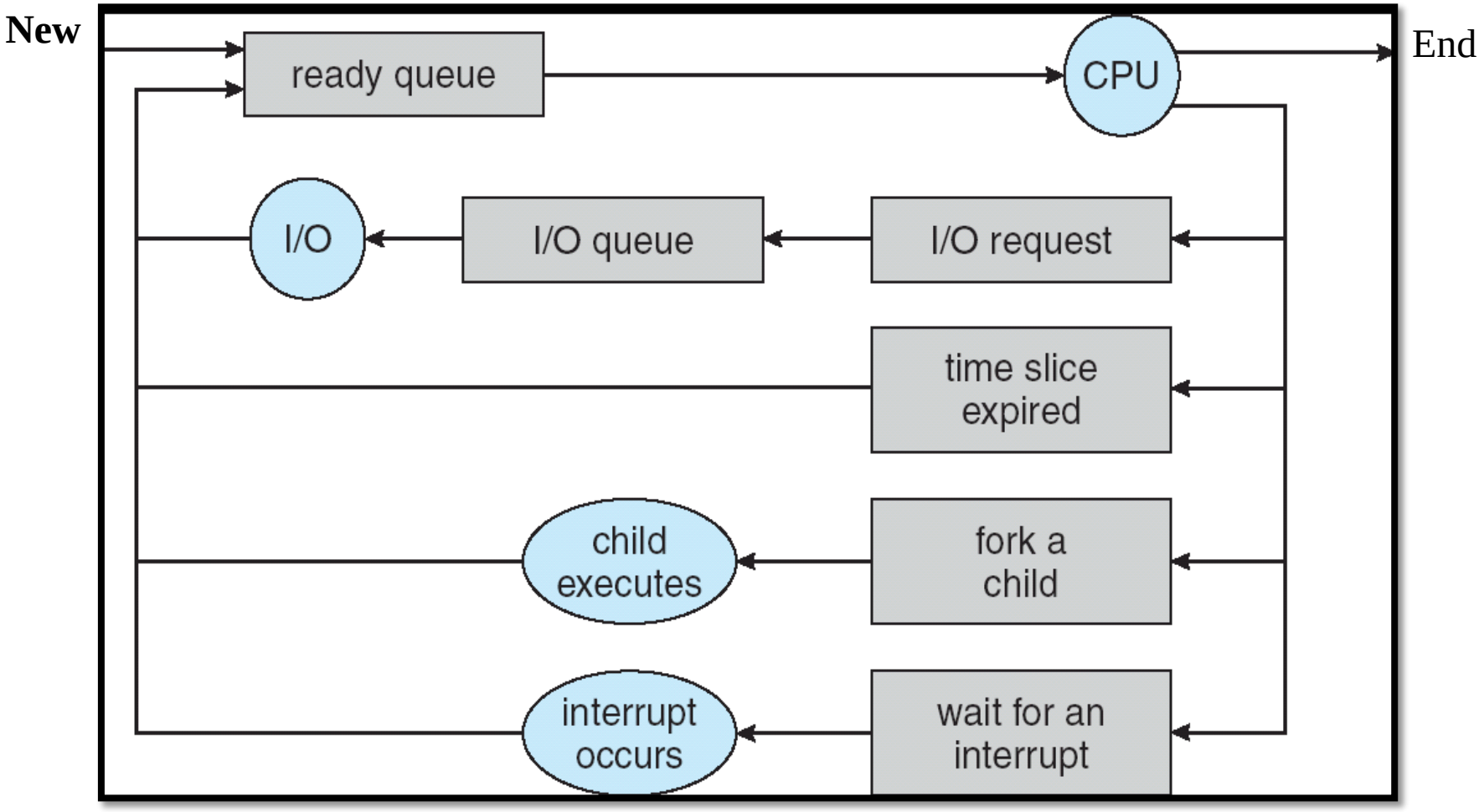



**Figure:** The ready queue and various I/O device queues.

## ❖ Queuing
## (Scheduling Queue / Representation of Process Scheduling)



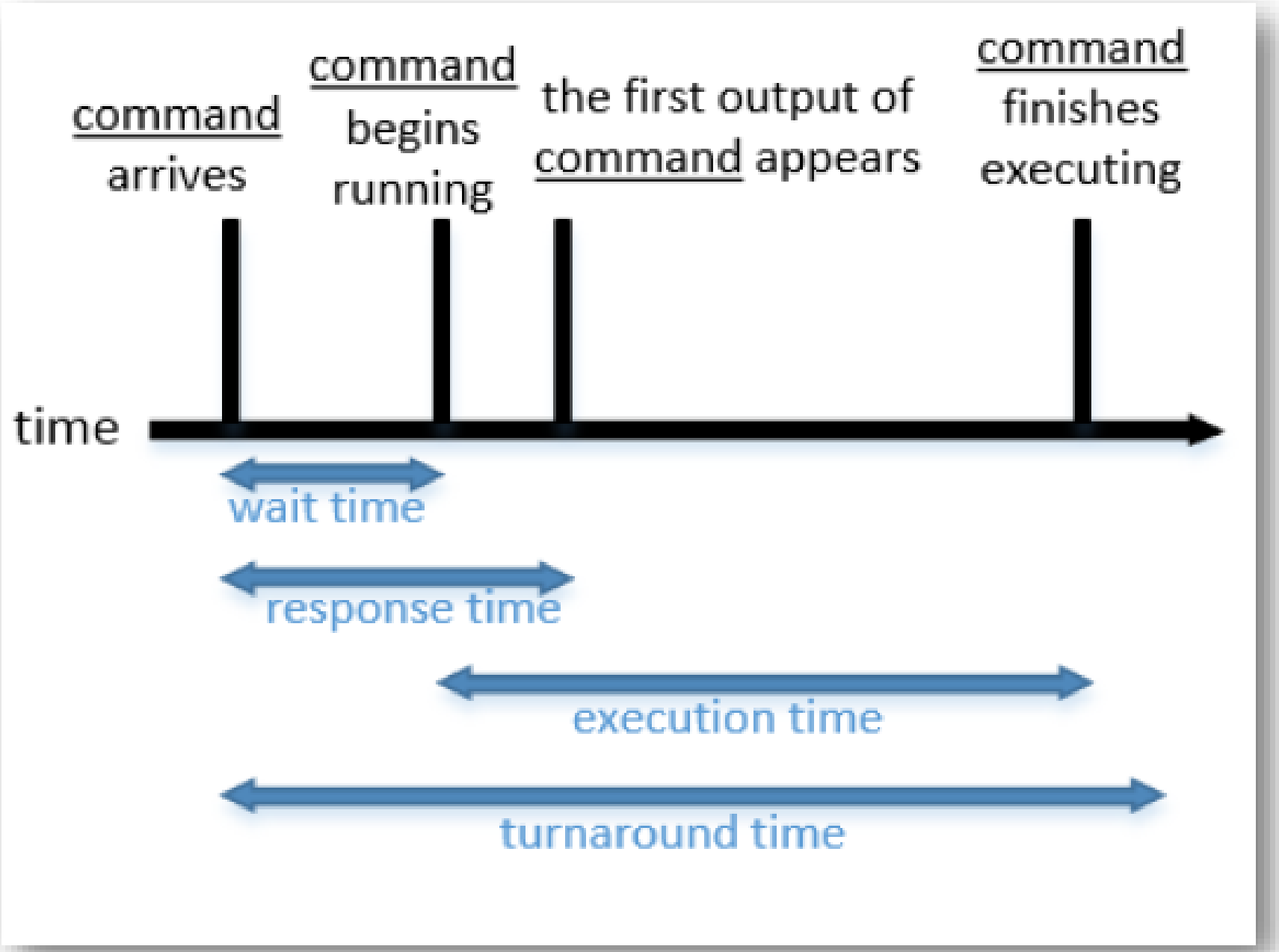
## ❖Scheduling

Set of policies & mechanisms in OS which covers an order in which work is completed by a computer system.

- **Scheduling Criteria/Objectives of Scheduling**

  - CPU Utilization
  - Throughput
  - Turnaround Time
  - Waiting Time
  - Response Time

- **CPU Utilization** – keep the CPU as busy as possible **(Maximize)**
- **Throughput** – no. of processes those complete their execution per time unit **(Maximize)**
- **Turnaround Time** – time to execute a process from submission to completion **(Minimize)**
- **Waiting Time** – amount of time a process has been waiting in the ready queue **(Minimize)**
- **Response Time** – time it takes from when a request was submitted until the first response is produced **(Minimize)**


command arrives
command begins running
the first output of command appears
command finishes executing
time
wait time
response time
execution time
turnaround time

# ❖ Scheduler

- **Special system software** which handles process scheduling.
- Main task **is to select the jobs to be submitted** into the system.

- **Types of Scheduler:**

- **Long Term Scheduler(LTS)**

  (Job Scheduler/ High level scheduler)

- **Middle Term Scheduler(MTS)**

  (Intermediate Scheduler)

- **Short Term Scheduler(STS)**

  (Low Level Scheduler/CPU Scheduler)

**Types of Scheduler Continued…**

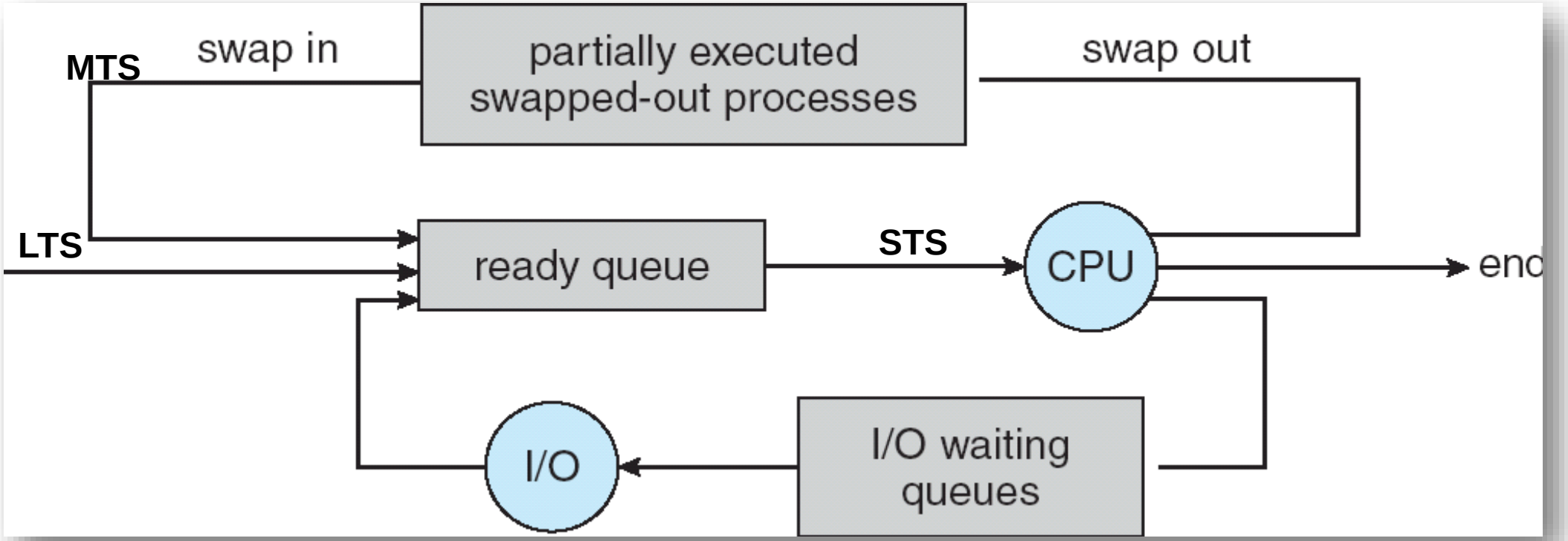



**Figure:** Types of Schedulers

- **Long Term Scheduler**
  - Also called **job scheduler**
  - Executes much less frequently
  - Takes more time to make decision
  - Make a careful selection of process
  - In-charge of handling **new to ready transition**
- **Short Term Scheduler**
  - Also called **CPU scheduler**
  - In-charge of handling **ready to running transition**
  - Selects new process frequently
    - Executes at least once in every 100 milliseconds

- **Medium Term Scheduler**
  - Reduces the **degree of multiprogramming**
  - In-charge of handling the **swapped out-processes**.
  - Swapping may be necessary to improve the process mix.

- **Comparison of Types of Schedulers**

| S.N. | Long Term Scheduler | Short Term Scheduler | Medium Term Scheduler |
|---|---|---|---|
| 1 | It is a job scheduler | It is a CPU scheduler | It is a process swapping scheduler. |
| 2 | Speed is lesser than short term scheduler | Speed is fastest among other two | Speed is in between both short and long term scheduler. |
| 3 | It controls the degree of multiprogramming | It provides lesser control over degree of multiprogramming | It reduces the degree of multiprogramming. |
| 4 | It is almost absent or minimal in time sharing system | It is also minimal in time sharing system | It is a part of Time sharing systems. |
| 5 | It selects processes from pool and loads them into memory for execution | It selects those processes which are ready to execute | It can re-introduce the process into memory and execution can be continued. |

- **CPU Burst & I/O Burst**

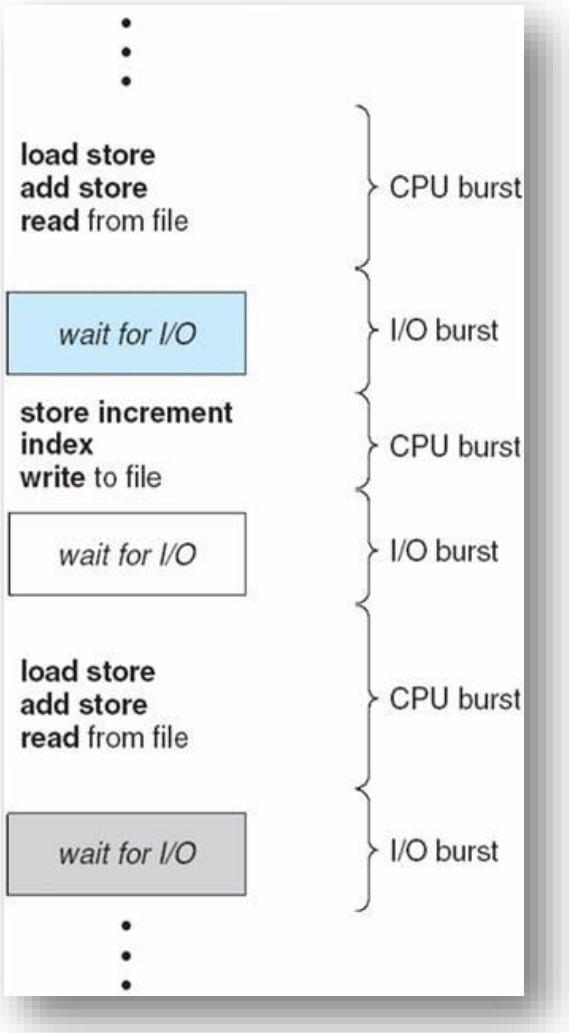
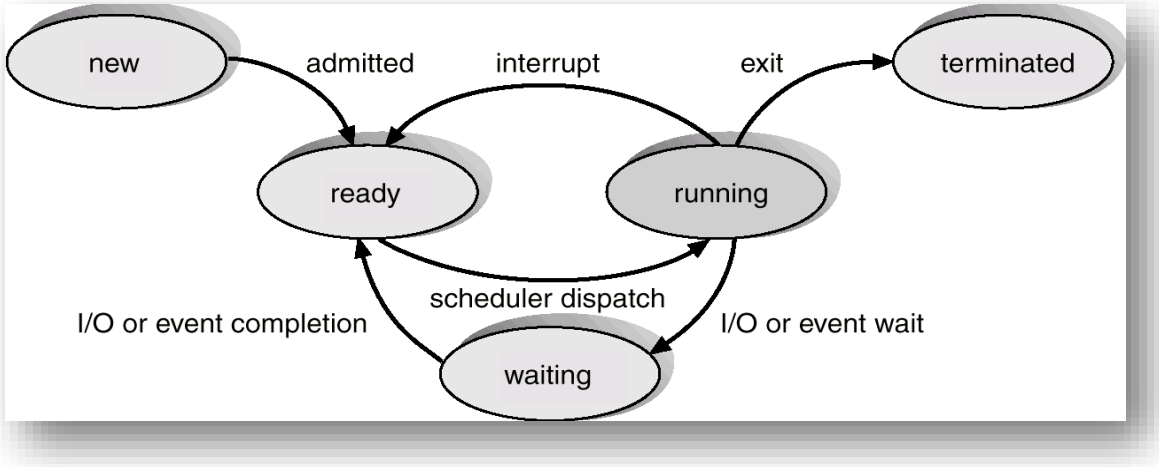



**Figure:** Alternating sequence of CPU and I/O bursts

- **CPU Scheduler**

- Selects from 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




- When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is **nonpreemptive** or **cooperative.**
- Otherwise, it is **preemptive.**

**Continued…**

- **Scheduling Types**

  - **nonpreemptive** –
    - Once CPU given to the process it cannot be preempted until the process completes its CPU burst.
    - 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.
    - Ex. Windows 3.x, Windows 95

  - **preemptive** – if a new process arrives with CPU burst length less than remaining time of currently executing process/ with higher priority, preempts the execution.

    - Mac OS X

- **Dispatcher**

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch Latency** – time it takes for the dispatcher to stop one process and start another running

# Scheduling Algorithms

- FCFS Scheduling Algorithm
- SJF (Non Preemptive)Scheduling Algorithm
- SJF (Preemptive)Scheduling Algorithm / SRTN (Shortest Remaining Time Next)
- Priority (Non Preemptive)Based Scheduling Algorithm
- Priority (Preemptive)Based Scheduling Algorithm
- Round Robin Scheduling Algorithm

# First-Come First-Served (FCFS) Scheduling

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- Suppose that the processes arrive in the order: $P_1$ , $P_2$ , $P_3$
- The **Gantt Chart** for the schedule is:


The Convoy Effect, visualized

## Shortest-Job-First (SJF) Scheduling

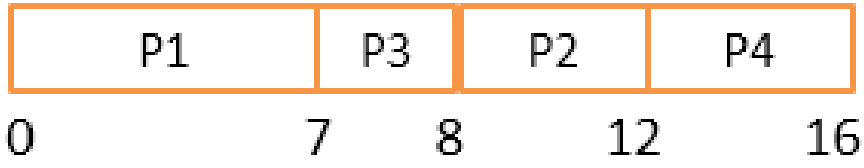
- Use the **length of next CPU burst** of a process to schedule the process with the shortest time
- Two schemes:
  - **nonpreemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst
  - **preemptive** – if a new process arrives with CPU burst length less than remaining time of currently executing process, preempts the execution.
    This scheme is also known as the **Shortest-Remaining-Time-First** (SRTF) or **Shortest-Remaining-Time-Next** (SRTN)
- SJF is **optimal** – gives minimum average waiting time for a given set of processes

**Preemptive- forcefully stoppage**

# Example of Non Preemptive SJF

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 2 | 4 |
| $P_3$ | 4 | 1 |
| $P_4$ | 5 | 4 |

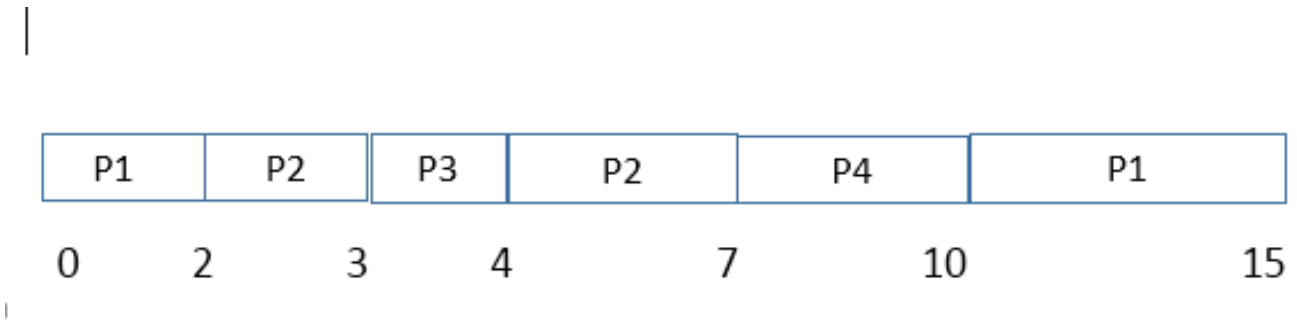
- SJF (non-preemptive)



# Example of Preemptive SJF/ SRTN (Shortest Remaining Time Next)

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 2 | 4 |
| $P_3$ | 3 | 1 |
| $P_4$ | 4 | 3 |

**SJF (preemptive)**

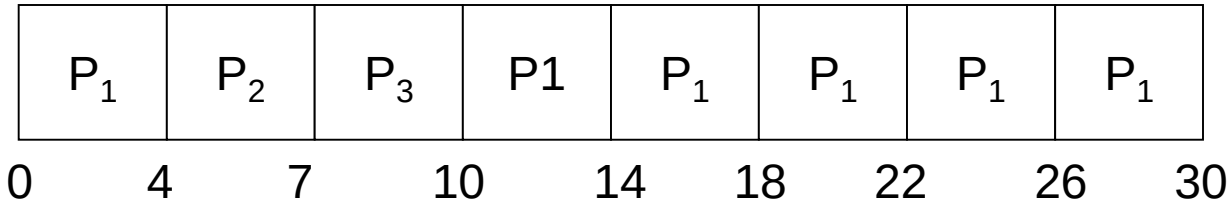


# Round Robin (RR) Scheduling

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

Time Quantum = 4

The Gantt chart is:




Typically, higher average turnaround than SJF, but better *response.*

# Priority Based Scheduling

- A priority number (integer) is associated with each process.
- The CPU is allocated to the process with the highest priority (eg: smaller integer ≈ higher priority)
- Two schemes:
  - Preemptive
  - nonpreemptive
- SJF is a priority scheduling where priority is '**the predicted next CPU burst time**'
- **Problem** => **Starvation** – low priority processes may never execute
- **Solution** => **Aging** – as time progresses increase the priority of the processes.

## Consider the following set of processes executing on a uniprocessor system

| Process | A.T. | B.T. | Priority |
|---|---|---|---|
| $P_1$ | 0 | 15 | 6 |
| $P_2$ | 1 | 13 | 5 |
| $P_3$ | 3 | 10 | 4 |
| $P_4$ | 5 | 8 | 3 |
| $P_5$ | 5 | 7 | 2 |
| $P_6$ | 7 | 4 | 3 |
| $P_7$ | 10 | 2 | 1 |

Draw Gantt chart & calculate average Turnaround Time and Waiting Time for following CPU Scheduling algorithms:

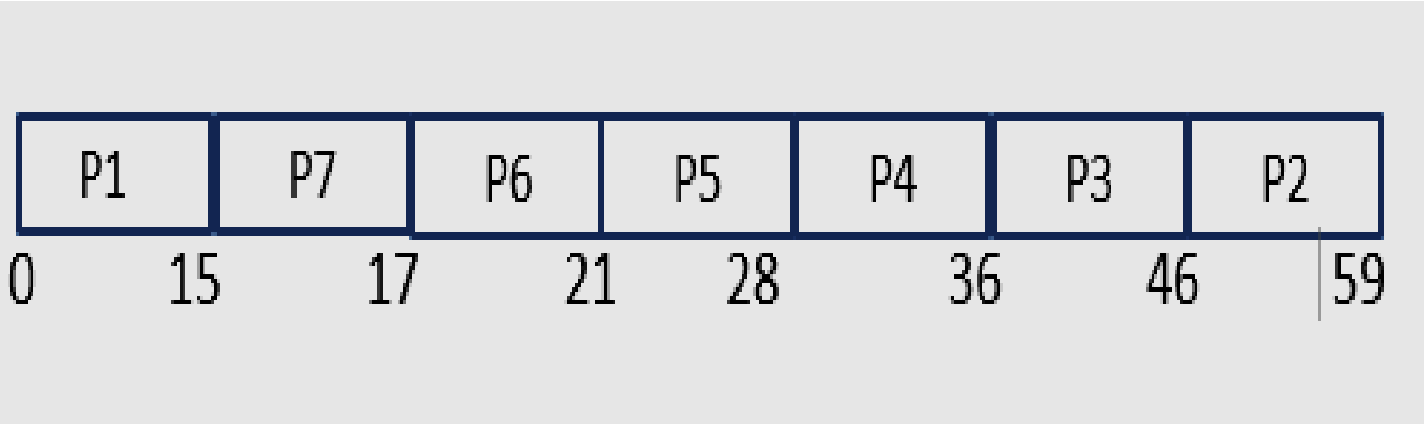
1) SJF non preemptive
2) SJF preemptive
3) Priority based preemptive

(Consider lower the number – higher its priority)

- Solution:

| Process | A.T. | B.T. | Priority |
|---|---|---|---|
| $P_1$ | 0 | 15 | 6 |
| $P_2$ | 1 | 13 | 5 |
| $P_3$ | 3 | 10 | 4 |
| $P_4$ | 5 | 8 | 3 |
| $P_5$ | 5 | 7 | 2 |
| $P_6$ | 7 | 4 | 3 |
| $P_7$ | 10 | 2 | 1 |

1) SJF non preemptive



**Turnaround Time (TT) = (Time of complete execution) – (Arrival Time)**

| | | | | | |
|---|---|---|---|---|---|
| TT 1 (P1) | = | 15 | - | 0 | = 15 |
| TT 2 (P2) | = | 59 | - | 1 | = 58 |
| TT 3 (P3) | = | 46 | - | 3 | = 43 |
| TT 4 (P4) | = | 36 | - | 5 | = 31 |
| TT 5 (P5) | = | 28 | - | 5 | = 23 |
| TT 6 (P6) | = | 21 | - | 7 | = 14 |
| TT 7 (P7) | = | 17 | - | 10 | = 7 |

**Avg TT = 191/7 = 27.42**

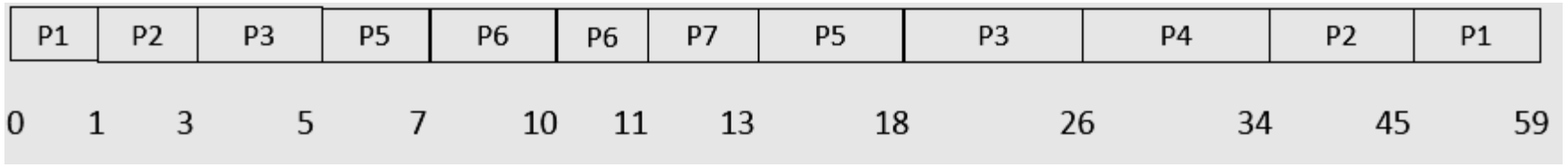
- **Waiting Time (WT) = (Turnaround Time) – (Burst Time)**

| | | | | | | |
|---|---|---|---|---|---|---|
| **$WT_1$ (P1)** | **=** | **15** | **-** | **15** | **=** | **0** |
| **$WT_2$ (P2)** | **=** | **58** | **-** | **13** | **=** | **45** |
| **$WT_3$ (P3)** | **=** | **43** | **-** | **10** | **=** | **33** |
| **$WT_4$ (P4)** | **=** | **31** | **-** | **8** | **=** | **23** |
| **$WT_5$ (P5)** | **=** | **23** | **-** | **7** | **=** | **16** |
| **$WT_6$ (P6)** | **=** | **14** | **-** | **4** | **=** | **10** |
| **$WT_7$ (P7)** | **=** | **7** | **-** | **2** | **=** | **5** |
| **Avg WT** | **=** | **132/7** | | | **=** | **18.85** |

- Solution:

| Process | A.T. | B.T. | Priority |
|---|---|---|---|
| $P_1$ | 0 | 15 | 6 |
| $P_2$ | 1 | 13 | 5 |
| $P_3$ | 3 | 10 | 4 |
| $P_4$ | 5 | 8 | 3 |
| $P_5$ | 5 | 7 | 2 |
| $P_6$ | 7 | 4 | 3 |
| $P_7$ | 10 | 2 | 1 |

## 2) SJF preemptive



- **Turnaround Time (TT) = (Time of complete execution) – (Arrival Time)**

| | | | | | | |
|---|---|---|---|---|---|---|
| **TT 1 (P1)** | **=** | **59** | **-** | **0** | **=** | **59** |
| **TT 2 (P2)** | **=** | **45** | **-** | **1** | **=** | **44** |
| **TT 3 (P3)** | **=** | **26** | **-** | **3** | **=** | **23** |
| **TT 4 (P4)** | **=** | **34** | **-** | **5** | **=** | **29** |
| **TT 5 (P5)** | **=** | **18** | **-** | **5** | **=** | **13** |
| **TT 6 (P6)** | **=** | **11** | **-** | **7** | **=** | **4** |
| **TT 7 (P7)** | **=** | **13** | **-** | **10** | **=** | **3** |

**Avg TT = 175/7 = 25**

- **Waiting Time (WT) = (Turnaround Time) – (Burst Time)**

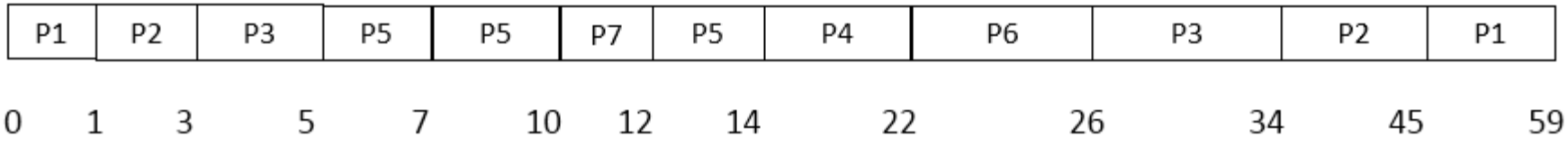
| | | | | | | |
|---|---|---|---|---|---|---|
| **$WT_1$ (P1)** | **=** | **59** | **-** | **15** | **=** | **44** |
| **$WT_2$ (P2)** | **=** | **44** | **-** | **13** | **=** | **31** |
| **$WT_3$ (P3)** | **=** | **23** | **-** | **10** | **=** | **13** |
| **$WT_4$ (P4)** | **=** | **29** | **-** | **8** | **=** | **21** |
| **$WT_5$ (P5)** | **=** | **13** | **-** | **7** | **=** | **6** |
| **$WT_6$ (P6)** | **=** | **4** | **-** | **4** | **=** | **0** |
| **$WT_7$ (P7)** | **=** | **3** | **-** | **2** | **=** | **1** |

**Avg WT = 116/7 = 16.57**

# 3)Priority Based Preemptive

| Process | A.T. | B.T. | Priority |
|---|---|---|---|
| $P_1$ | 0 | 15 | 6 |
| $P_2$ | 1 | 13 | 5 |
| $P_3$ | 3 | 10 | 4 |
| $P_4$ | 5 | 8 | 3 |
| $P_5$ | 5 | 7 | 2 |
| $P_6$ | 7 | 4 | 3 |
| $P_7$ | 10 | 2 | 1 |

**Solution:**



- **Turnaround Time (TT) = (Time of complete execution) – (Arrival Time)**

| | | | | | | |
|---|---|---|---|---|---|---|
| **TT 1 (P1)** | **=** | **59** | **-** | **0** | **=** | **59** |
| **TT 2 (P2)** | **=** | **45** | **-** | **1** | **=** | **44** |
| **TT 3 (P3)** | **=** | **34** | **-** | **3** | **=** | **31** |
| **TT 4 (P4)** | **=** | **22** | **-** | **5** | **=** | **17** |
| **TT 5 (P5)** | **=** | **14** | **-** | **5** | **=** | **9** |
| **TT 6 (P6)** | **=** | **26** | **-** | **7** | **=** | **19** |
| **TT 7 (P7)** | **=** | **12** | **-** | **10** | **=** | **2** |

**Avg TT = 181/7 = 25.87**

- **Waiting Time (WT) = (Turnaround Time) – (Burst Time)**

| | | | | | | |
|---|---|---|---|---|---|---|
| **$WT_1$ (P1)** | **=** | **59** | **-** | **15** | **=** | **44** |
| **$WT_2$ (P2)** | **=** | **44** | **-** | **13** | **=** | **31** |
| **$WT_3$ (P3)** | **=** | **31** | **-** | **10** | **=** | **21** |
| **$WT_4$ (P4)** | **=** | **17** | **-** | **8** | **=** | **9** |
| **$WT_5$ (P5)** | **=** | **9** | **-** | **7** | **=** | **2** |
| **$WT_6$ (P6)** | **=** | **19** | **-** | **4** | **=** | **15** |
| **$WT_7$ (P7)** | **=** | **2** | **-** | **2** | **=** | **0** |

**Avg WT = 122/7 = 17.42**

**Consider the following set of processes executing on a uniprocessor system:**

| Process | A.T. | B.T. |
|---|---|---|
| $P_1$ | 0 | 12 |
| $P_2$ | 1 | 10 |
| $P_3$ | 2 | 7 |
| $P_4$ | 2 | 4 |
| $P_5$ | 5 | 2 |

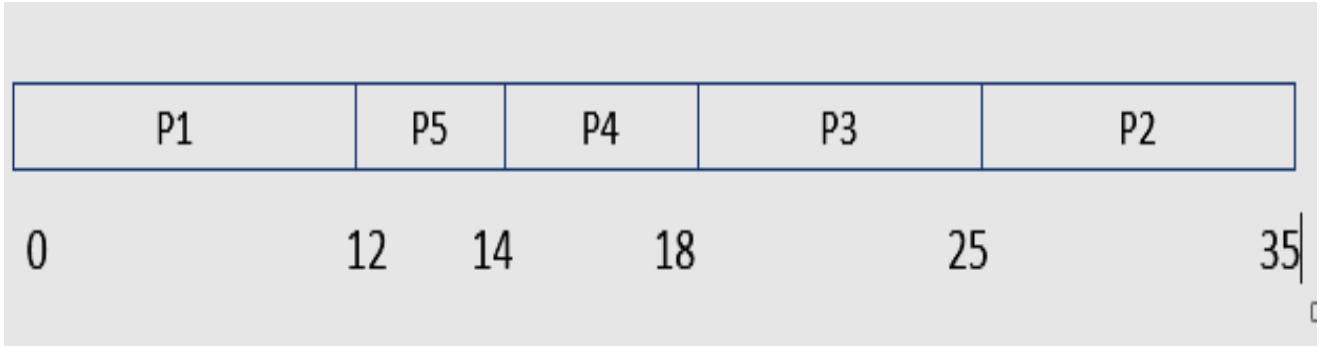
Draw Gantt chart & calculate average Turnaround Time and Waiting Time for following CPU Scheduling algorithms:

1) SJF non preemptive
2) SJF preemptive

**Solution:**

| Process | A.T. | B.T. |
|---|---|---|
| $P_1$ | 0 | 12 |
| $P_2$ | 1 | 10 |
| $P_3$ | 2 | 7 |
| $P_4$ | 2 | 4 |
| $P_5$ | 5 | 2 |

**1) SJF non preemptive**



- **Turnaround Time (TT) = (Time of complete execution) – (Arrival Time)**

| | | | | | |
|---|---|---|---|---|---|
| **TT 1 (P1)** | **=** | **12** | **-** | **0** | **= 12** |
| **TT 2 (P2)** | **=** | **35** | **-** | **1** | **= 34** |
| **TT 3 (P3)** | **=** | **25** | **-** | **2** | **= 23** |
| **TT 4 (P4)** | **=** | **18** | **-** | **2** | **= 16** |
| **TT 5 (P5)** | **=** | **14** | **-** | **5** | **= 9** |
| **Avg TT** | **=** | **94/5 = 18.8** | | | |

- **Waiting Time (WT) = (Turnaround Time) – (Burst Time)**

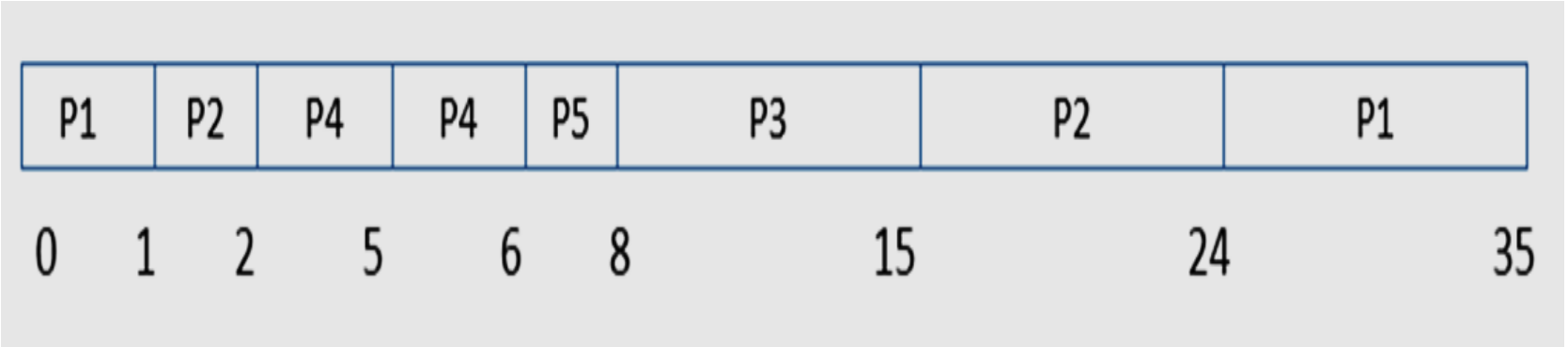
| | | | | | |
|---|---|---|---|---|---|
| **$WT_1$ (P1)** | **=** | **12** | **-** | **12** | **= 0** |
| **$WT_2$ (P2)** | **=** | **34** | **-** | **10** | **= 24** |
| **$WT_3$ (P3)** | **=** | **23** | **-** | **7** | **= 16** |
| **$WT_4$ (P4)** | **=** | **16** | **-** | **4** | **= 12** |
| **$WT_5$ (P5)** | **=** | **9** | **-** | **2** | **= 7** |

**Avg WT = 59/5 = 11.8**

**Solution:**

| Process | A.T. | B.T. |
|---|---|---|
| $P_1$ | 0 | 12 |
| $P_2$ | 1 | 10 |
| $P_3$ | 2 | 7 |
| $P_4$ | 2 | 4 |
| $P_5$ | 5 | 2 |

**2) SJF preemptive**



- **Turnaround Time (TT) = (Time of complete execution) – (Arrival Time)**

| | | | | | |
|---|---|---|---|---|---|
| **TT 1 (P1)** | **=** | **35** | **-** | **0** | **= 35** |
| **TT 2 (P2)** | **=** | **24** | **-** | **1** | **= 23** |
| **TT 3 (P3)** | **=** | **15** | **-** | **2** | **= 13** |
| **TT 4 (P4)** | **=** | **6** | **-** | **2** | **= 4** |
| **TT 5 (P5)** | **=** | **8** | **-** | **5** | **= 3** |

**Avg TT = 78/5 = 15.6**

- **Waiting Time (WT) = (Turnaround Time) – (Burst Time)**

| | | | | | |
|---|---|---|---|---|---|
| **$WT_1$ (P1)** | **=** | **35** | **-** | **12** | **= 23** |
| **$WT_2$ (P2)** | **=** | **23** | **-** | **10** | **= 13** |
| **$WT_3$ (P3)** | **=** | **13** | **-** | **7** | **= 6** |
| **$WT_4$ (P4)** | **=** | **4** | **-** | **4** | **= 0** |
| **$WT_5$ (P5)** | **=** | **3** | **-** | **2** | **= 1** |
| **Avg WT** | **=** | **43/5 = 8.6** | | | |

## Consider the following set of processes executing on a uniprocessor system

| Process | A.T. | B.T. | Priority |
|---|---|---|---|
| $P_1$ | 0 | 13 | 6 |
| $P_2$ | 1 | 11 | 5 |
| $P_3$ | 2 | 9 | 4 |
| $P_4$ | 4 | 7 | 3 |
| $P_5$ | 4 | 5 | 2 |
| $P_6$ | 8 | 3 | 3 |
| $P_7$ | 10 | 2 | 1 |

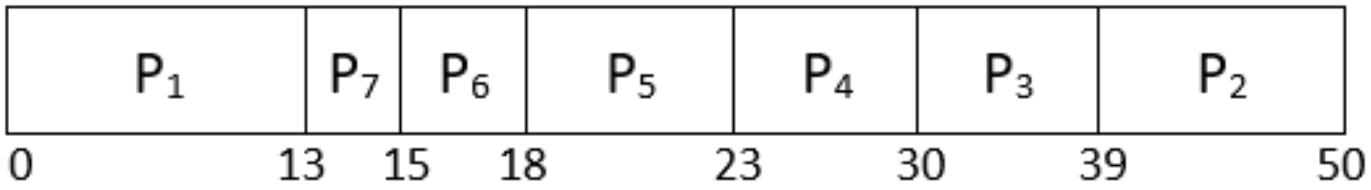
Draw Gantt chart & calculate average Turnaround Time and Waiting Time for following CPU Scheduling algorithms:

1) SJF non preemptive
2) SJF preemptive
3) Priority based preemptive

(Consider lower the number – higher its priority)

| Process | A.T. | B.T. | Priority |
|---|---|---|---|
| $P_1$ | 0 | 13 | 6 |
| $P_2$ | 1 | 11 | 5 |
| $P_3$ | 2 | 9 | 4 |
| $P_4$ | 4 | 7 | 3 |
| $P_5$ | 4 | 5 | 2 |
| $P_6$ | 8 | 3 | 3 |
| $P_7$ | 10 | 2 | 1 |

1) SJF non preemptive



**Turnaround Time (TT) = (Time of complete execution) – (Arrival Time)**

| | | | | | | |
|---|---|---|---|---|---|---|
| TT 1 (P1) | = | 13 | - | 0 | = | 13 |
| TT 2 (P2) | = | 50 | - | 1 | = | 49 |
| TT 3 (P3) | = | 39 | - | 2 | = | 37 |
| TT 4 (P4) | = | 30 | - | 4 | = | 26 |
| TT 5 (P5) | = | 23 | - | 4 | = | 19 |
| TT 6 (P6) | = | 18 | - | 8 | = | 10 |
| TT 7 (P7) | = | 15 | - | 10 | = | 5 |

**Avg TT = 159 / 7 = 22.71**

- **Waiting Time (WT) = (Turnaround Time) – (Burst Time)**

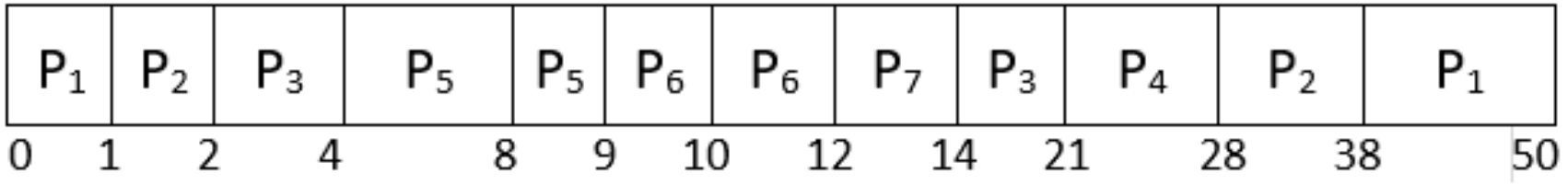
| | | | | | | |
|---|---|---|---|---|---|---|
| **$WT_1$ (P1)** | **=** | **13** | **-** | **13** | **=** | **0** |
| **$WT_2$ (P2)** | **=** | **49** | **-** | **11** | **=** | **38** |
| **$WT_3$ (P3)** | **=** | **37** | **-** | **9** | **=** | **28** |
| **$WT_4$ (P4)** | **=** | **26** | **-** | **7** | **=** | **19** |
| **$WT_5$ (P5)** | **=** | **19** | **-** | **5** | **=** | **14** |
| **$WT_6$ (P6)** | **=** | **10** | **-** | **3** | **=** | **7** |
| **$WT_7$ (P7)** | **=** | **5** | **-** | **2** | **=** | **3** |

**Avg WT = 109 / 7 = 15.57**

- Solution:

| Process | A.T. | B.T. | Priority |
|---|---|---|---|
| $P_1$ | 0 | 13 | 6 |
| $P_2$ | 1 | 11 | 5 |
| $P_3$ | 2 | 9 | 4 |
| $P_4$ | 4 | 7 | 3 |
| $P_5$ | 4 | 5 | 2 |
| $P_6$ | 8 | 3 | 3 |
| $P_7$ | 10 | 2 | 1 |

## 2) SJF preemptive



- **Turnaround Time (TT) = (Time of complete execution) – (Arrival Time)**

| | | | | | | |
|---|---|---|---|---|---|---|
| **TT 1 (P1)** | **=** | **50** | **-** | **0** | **=** | **50** |
| **TT 2 (P2)** | **=** | **38** | **-** | **1** | **=** | **37** |
| **TT 3 (P3)** | **=** | **21** | **-** | **2** | **=** | **19** |
| **TT 4 (P4)** | **=** | **28** | **-** | **4** | **=** | **24** |
| **TT 5 (P5)** | **=** | **9** | **-** | **4** | **=** | **5** |
| **TT 6 (P6)** | **=** | **12** | **-** | **8** | **=** | **4** |
| **TT 7 (P7)** | **=** | **14** | **-** | **10** | **=** | **4** |

**Avg TT = 143 / 7 = 20.42**

- **Waiting Time (WT) = (Turnaround Time) – (Burst Time)**

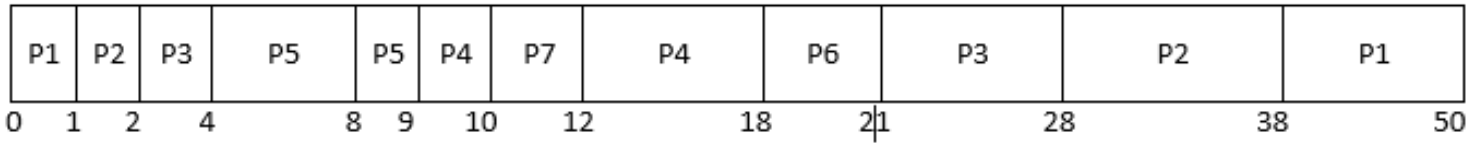
| | | | | | | |
|---|---|---|---|---|---|---|
| **$WT_1$ (P1)** | **=** | **50** | **-** | **13** | **=** | **37** |
| **$WT_2$ (P2)** | **=** | **37** | **-** | **11** | **=** | **26** |
| **$WT_3$ (P3)** | **=** | **19** | **-** | **9** | **=** | **10** |
| **$WT_4$ (P4)** | **=** | **24** | **-** | **7** | **=** | **17** |
| **$WT_5$ (P5)** | **=** | **5** | **-** | **5** | **=** | **0** |
| **$WT_6$ (P6)** | **=** | **4** | **-** | **3** | **=** | **1** |
| **$WT_7$ (P7)** | **=** | **4** | **-** | **2** | **=** | **2** |

**Avg WT = 93 / 7 = 13.28**

- Solution:

| Process | A.T. | B.T. | Priority |
|---|---|---|---|
| $P_1$ | 0 | 13 | 6 |
| $P_2$ | 1 | 11 | 5 |
| $P_3$ | 2 | 9 | 4 |
| $P_4$ | 4 | 7 | 3 |
| $P_5$ | 4 | 5 | 2 |
| $P_6$ | 8 | 3 | 3 |
| $P_7$ | 10 | 2 | 1 |

## 3)Priority based preemptive



- **Turnaround Time (TT) = (Time of complete execution) – (Arrival Time)**

| | | | | | | |
|---|---|---|---|---|---|---|
| **TT 1 (P1)** | **=** | **50** | **-** | **0** | **=** | **50** |
| **TT 2 (P2)** | **=** | **38** | **-** | **1** | **=** | **37** |
| **TT 3 (P3)** | **=** | **28** | **-** | **2** | **=** | **26** |
| **TT 4 (P4)** | **=** | **18** | **-** | **4** | **=** | **14** |
| **TT 5 (P5)** | **=** | **9** | **-** | **4** | **=** | **5** |
| **TT 6 (P6)** | **=** | **21** | **-** | **8** | **=** | **13** |
| **TT 7 (P7)** | **=** | **12** | **-** | **10** | **=** | **2** |

**Avg TT = 147 / 7 = 21**

- **Waiting Time (WT) = (Turnaround Time) – (Burst Time)**

| | | | | | | |
|---|---|---|---|---|---|---|
| **$WT_1$ (P1)** | **=** | **50** | **-** | **13** | **=** | **37** |
| **$WT_2$ (P2)** | **=** | **37** | **-** | **11** | **=** | **26** |
| **$WT_3$ (P3)** | **=** | **26** | **-** | **9** | **=** | **17** |
| **$WT_4$ (P4)** | **=** | **14** | **-** | **7** | **=** | **7** |
| **$WT_5$ (P5)** | **=** | **5** | **-** | **5** | **=** | **0** |
| **$WT_6$ (P6)** | **=** | **13** | **-** | **3** | **=** | **10** |
| **$WT_7$ (P7)** | **=** | **2** | **-** | **2** | **=** | **0** |

**Avg WT = 97/7 = 13.85**

# Q. Consider following jobs to be executing on a uniprocessor system

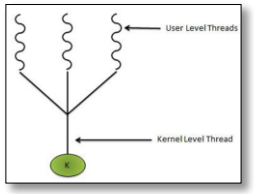
| Process | AT | BT |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 1 | 5 |
| P3 | 3 | 1 |
| P4 | 5 | 4 |
| P5 | 7 | 3 |

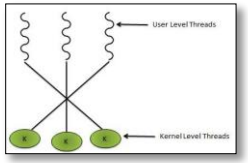
Draw Gantt chart and calculate average waiting time and turnaround time for following scheduling algorithms:

i)FCFS ii) SJF (Non-Preemptive) iii) SJF (Preemptive)

Justify which will be the best scheduling algorithm out of these three.

# Introduction to Thread Scheduling




- User-level threads are managed by a **thread library** and the kernel is unaware of them.
- To run on a CPU, user-level threads must ultimately be **mapped to an associated** kernel-level thread.
- One **distinction between** user-level and kernel-level threads lies in **how they are scheduled.**
- On systems implementing the **many-to-one and many-to-many models**, the thread library schedules user-level threads to run on an available LWP.
- This scheme is known as **Process-Contention Scope (PCS)** since **competition for the CPU** takes place **among threads belonging to the same process.**

**Continued…**

- To decide which **kernel-level thread** to schedule onto a CPU, the kernel uses **System-Contention Scope (SCS).** Competition for the CPU with SCS scheduling takes place **among all threads in the system.**

- Typically, **PCS** is done according to **priority** - the scheduler selects the runnable thread with the highest priority to run.

- **User-level thread priorities** are set **by the programmer** and are not adjusted by the thread library. (although some thread libraries may allow the programmer **to change the priority** of a thread.)

- It is important to note that PCS will typically **preempt** the thread currently running **in favor of a** higher-priority thread.

# References


**Text Books:**

1. **William Stallings**, O**perating System: Internals and Design Principles**, 8th Edition, Prentice Hall, 2014.
2. Abraham Silberschatz, Peter Baer **Galvin** and Greg Gagne, **Operating System Concepts**, 9th Edition, John Wiley & Sons, Inc., 2016.
3. Andrew **Tannenbaum**, **Operating System Design and Implementation**, 3rd Edition, Pearson, 2015.

**Reference Books:**

1. Maurice J. Bach, Design of UNIX Operating System, 2nd Edition, PHI, 2004.
2. Achyut Godbole and Atul Kahate, Operating Systems, 3rd Edition, McGraw Hill Education, 2017.
3. The Linux Kernel Book, Remy Card, Eric Dumas, Frank Mevel, 1st Edition, Wiley Publications, 2013.