Concepts of Operating Systems And Linux Operating Systems

Classification of OS

- OS can be categorized based on the target system (computers).
  - Mainframe systems
  - Desktop systems
  - Multi-processor (Parallel) systems
  - Distributed systems
  - Hand-held systems
  - Real-time systems

Mainframe systems

- Examples: UNIX and its flavours, IBM-360, etc.
- **Resident Monitor**
  - Early (oldest) OS resides in memory and monitor execution of the programs. If it fails, error is reported.
  - OS provides hardware interfacing that can be reused by all the programs.
- **Batch Systems**
  - The batch/group of similar programs is loaded in the computer, from which OS loads one program in the memory and execute it.
  - The programs are executed one after another.
  - In this case, if any process is performing IO, CPU will wait for that process and hence not utilized efficiently.

• Multi-Programming

- In multi-programming systems, multiple program can be loaded in the memory.
- The number of program that can be loaded in the memory at the same time, is called as "**degree of multi-programming**".
- In these systems, if one of the process is performing IO, CPU can continue execution of another program.
- This will increase CPU utilization.
- Each process will spend some time for CPU computation (CPU burst) and some time for IO (IO burst).
  - If CPU burst > IO burst, then process is called as "**CPU bound**".
  - If IO burst > CPU burst, then process is called as "**IO bound**".
- To efficiently utilize CPU, a good mix of CPU bound and IO bound processes should be loaded into memory.
- This task is performed by an unit of OS called as "**Job scheduler**" OR "**Long term scheduler**".
- If multiple programs are loaded into the RAM by job scheduler, then one of process need to be executed (dispatched) on the CPU.
- This selection is done by another unit of OS called as "**CPU scheduler**" OR "**Short term scheduler**".

- **Multi-tasking OR time-sharing**
  - CPU time is shared among multiple processes in the main memory is called as "**multi-tasking**".
  - In such system, a small amount of CPU time is given to each process repeatedly, so that response time for any process < 1 sec.
  - With this mechanism, multiple tasks (ready for execution) can execute concurrently.
  - There are two types of multi-tasking:
    - **Process based multitasking**:
      - Multiple independent processes are executing concurrently.
      - Processes running on multiple processors called as "**multi-processing**".
    - **Thread based multi-tasking OR multi-threading:**
      - Multiple parts/functions in a process are executing concurrently.

Multiprocessor systems

- The systems in which multiple processors are connected in a close circuit is called as "**multiprocessor computer".**
- The programs/OS take advantage of multiple processors in the computer are called as " Multi-procssing " programs/OS.
  - Windows Vista : First Windows OS designed for multi-processing.
  - Linux 2.5+ : Linux started supporting multi-processing.
  - terminal> uname -a
- Since multiple tasks can be executed on these processors simultaneously, such systems are also called as **"parallel systems".**
- Parallel systems have more throughput (Number of tasks done in unit time).
- There are two types of multiprocessor systems:
  - Asymmetric Multi-processing
  - Symmetric Multi-processing

- **Asymmetric Multi-processing**
  - OS treats one of the processor as master processor and schedule task for it.
  - The task is in turn divided into smaller tasks and get them done from other processors.

- **Symmetric Multi-processing**
  - OS considers all processors at same level and schedule tasks on each processor individually.
  - All modern desktop systems are SMP.

- **Multi-user**
  - Multiple users can execute multiple tasks concurrently on the same systems.
  - e.g. IBM 360, UNIX, Windows Servers, etc.
  - Each user can access system via different terminal.
  - There are many UNIX commands to track users and terminals.
    - **tty** (teletype) : It prints the name of the current terminal.
    - **who** : Information about currently logged in users, system boot time, runlevel , processes,…
    - **who am i** : Gives you the name of the current user, the terminal they are logged in at, the date and time when they logged in.
    - **whoami** : It gives username of the current user
    - **w** : Displays the users.

Desktop systems

- Personal computers -- desktop and laptops
- User convenience and Responsiveness
- Examples: Windows, Mac, Linux, few UNIX, ...

- **Distributed systems**
  - Multiple computers connected together in a close network is called as "distributed system".
  - Its advantages are high availability (24x7), high scalability (many clients, huge data), fault tolerance (any computer may fail).
  - The requests are redirected to the computer having less load using "load balancing" techniques.
  - The set of computers connected together for a certain task is called as "cluster".
  - Examples: Linux

- **Handheld systems**
  - OS installed on handheld devices like mobiles, PDAs, iPODs, etc.
  - Challenges:
    - Small screen size
    - Low end processors
    - Less RAM size
    - Battery powered
  - Examples: Symbian, iOS, Linux, PalmOS, WindowsCE, etc.

- **Real-time Systems**
  - The OS in which accuracy of results depends on accuracy of the computation as well as time duration in which results are produced, is called as "RTOS".
  - If results are not produced within certain time (deadline), catastrophic effects may occur.
  - These OS ensure that tasks will be completed in a definite time duration.
  - Time from the arrival of interrupt till begin handling of the interrupt is called as "Interrupt Latency".
  - RTOS have very small and fixed interrupt latencies.
  - RTOS Examples: uC-OS, VxWorks, pSOS, RTLinux, FreeRTOS, etc

➢Process Management

- When we say an OS does process management it means **an OS is responsible for process creation, to provide environment for an execution of a process, resource allocation, scheduling, resources management, inter process communication, process coordination, and terminate the process.**

Q. What is a Program?

❑User view:

- Program is a finite set of instructions written in any programming language given to the machine to do specific task.

❑System view:

- Program is an executable file in HDD which divided logically into sections like exe header, bss section, data section, rodata section, code section, symbol table.

Q. What is a Process?

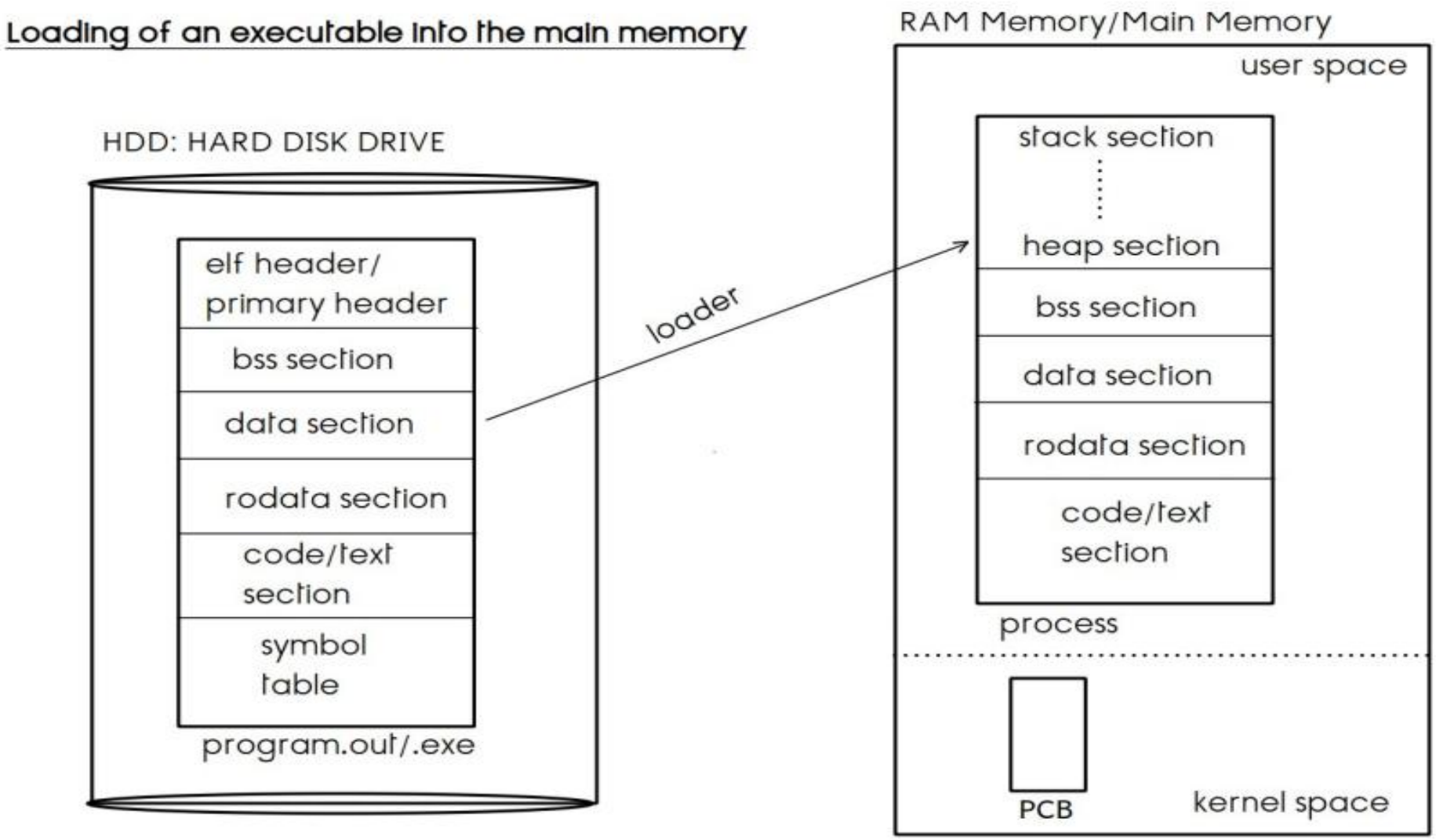
❑ **User view:**

- Program in execution is called as **a process.**
- Running program is called as **a process.**
- When a program gets loaded into the main memory it is referred as **a process.**
- Running instance of a program is referred as **a process**.

❑ **System view:**

- Process is a program loaded into the main memory which has got PCB into the main memory inside kernel space and program itself into the main memory inside user space has **got bss section, rodata section, code section, and two new sections gets added for the process .**

- **stack section:** contains function activation records of called functions.
- **heap section:** dynamically allocated memory


Loading of an executable into the main memory
HDD: HARD DISK DRIVE
elf header/
primary header
bss section
data section
rodata section
code/text
section
symbol
table
program.out/.exe
loader
RAM Memory/Main Memory
user space
stack section
heap section
bss section
data section
rodata section
code/text
section
process
PCB
kernel space

- As a kernel, core program of an a OS runs continuously into the main memory, part of the main memory which is occupied by the kernel **referred as kernel space** and whichever part is left **is referred as an user space**, so **main memory is divided logically into two parts: kernel space & user space.**
- **User programs gets loaded into the user space only.**
- When we execute a program or upon submission of a process very first one structure gets created into the main memory inside kernel space by an OS in which all **the information which is required to control an execution of that process** can be kept, this structure is referred as a **PCB.**
- **PCB : Process Control Block, is also called as a Process Descriptor.**
- Per process one PCB gets created and PCB remains inside the main memory throughout an execution of a program, upon exit PCB gets destroyed from the main memory.

❖PCB mainly contains:

- PID : Process ID
- PPID : Parent Processes ID
- PC : Program Counter
- Execution context
- Kernel stack
- Exit status

- CPU sched information, memory management information, information about resources allocated for that process, execution context etc...

➢To keep track on all running programs, an OS maintains few data structures referred **as kernel data structures:**

1. **Job queue:** it contains list of PCB's of all submitted processes.
2. **Ready queue:** it contains list of PCB's of processes which are in the main memory and waiting for the CPU time.
3. **Waiting queue:** it contains list of PCB's of processes which are requesting for that particular device.

➢**Scheduler :**

1. **Job Scheduler/Long Term Scheduler:** it is a system program which selects/schedules jobs/processes from job queue to **load them onto the ready queue**.
2. **CPU Scheduler/Short Term Scheduler:** it is a system program which selects/schedules job/process **from ready queue to load it onto the CPU.**
   - **Dispatcher:** it is a system program which loads a process onto the CPU which is scheduled by the CPU scheduler, and **the time required for the dispatcher to stops an execution of one process and to starts an execution of another process is referred as dispatcher latency.**

➢Context Switch:

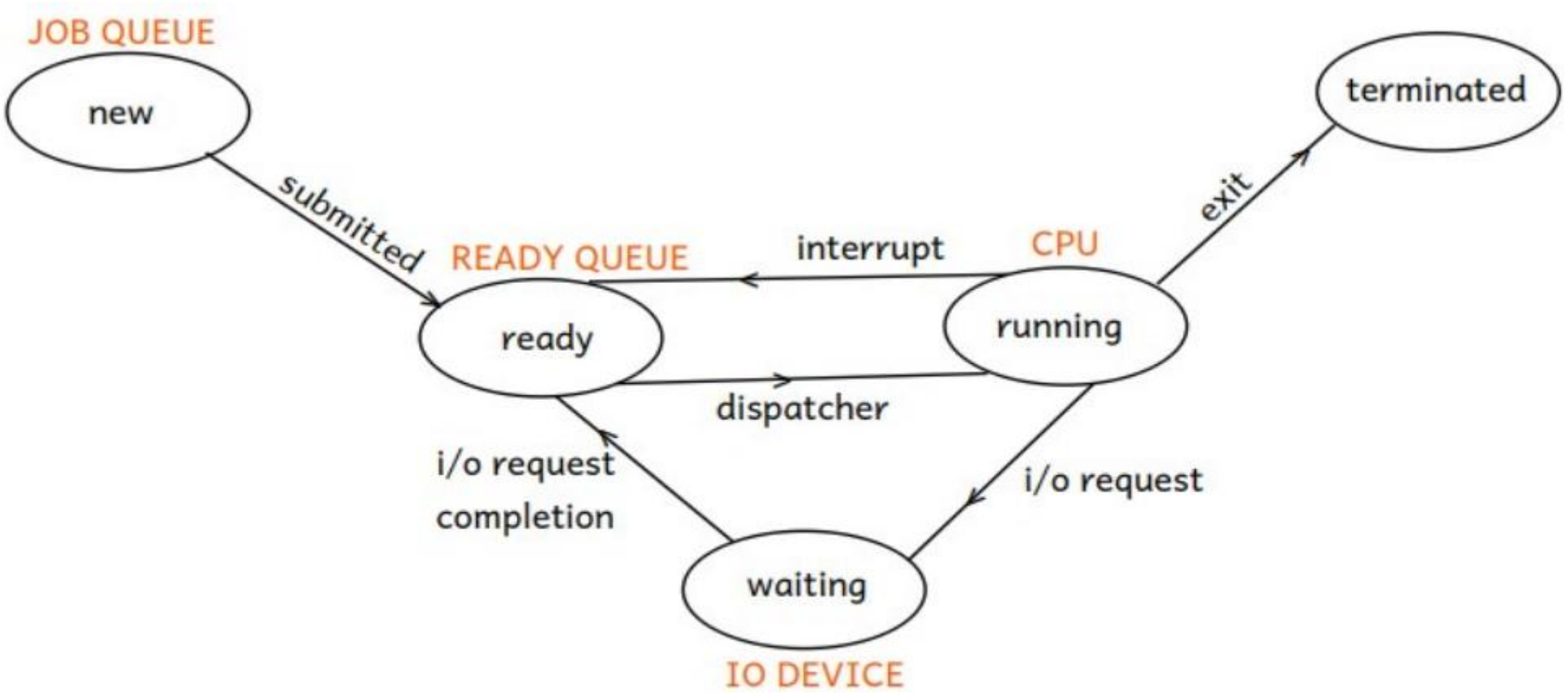
- As during context-switch, the CPU gets switched from an execution context of one process onto an execution context of another process, and hence it is referred as **"context-switch".**
- **context-switch = state-save + state-restore**
- **state-save** of suspended process can be done i.e. an execution context of suspended process gets saved into its PCB.
- **state-restore** of a process which is scheduled by the CPU scheduler can be done by the dispatcher, dispatcher copies an execution context of process scheduled by the cpu scheduler from its PCB and restore it onto the CPU registers.
- When a high priority process arrived into the ready queue, low priority process gets suspended by means of sending an interrupt, and control of the CPU gets allocated to the high priority process, and its execution gets completed first, then low priority process can be resumed back, i.e. the CPU starts executing suspended process from the point at which it was suspended and onwards.

Process States:

❖Throughout execution, process goes through different states out of which at a time it can be only in a one state.

1. **New state:**
   - New process PCB is created and added into job queue. PCB is initialized and process get ready for execution.
2. **Ready state:**
   - The ready process is added into the ready queue. Scheduler pick a process for scheduling from ready queue and dispatch it on CPU
3. **Running state:**
   - The process runs on CPU. If process keeps running on CPU, the timer interrupt is used to forcibly put it into ready state and allocate CPU time to other process
4. **Waiting state:**
   - If running process request for IO device, the process waits for completion of the IO. The waiting state is also called as sleeping or blocked state
5. **Terminated state:**
   - If running process exits, it is terminated

- **Linux:**
  - TASK_RUNNING (R), TASK_INTERRUPTIBLE (S), TASK_UNINTERRUPTIBLE (D), TASK_STOPPED(T), TASK_ZOMBIE (Z), TASK_DEAD (X)



PROCESS STATE DIAGRAM

❖CPU Scheduler gets called in the following four cases:

Case-1: Running -> Terminated

Case-2: Running -> Waiting

Case 3: Running -> Ready

Case-4: Waiting -> Ready

- There are two types of CPU scheduling:

1. **Non-preemptive:**
   - The current process gives up CPU voluntarily (for IO, terminate or yield).
   - Then CPU scheduler picks next process for the execution.
   - If each process yields CPU so that other process can get CPU for the execution, it is referred as "Co operative scheduling".

   - e.g. in above case 1 & case 2

2. **Preemptive:**
   - The current process may give up CPU voluntarily or paused forcibly (for high priority process or upon completion of its time quantum).
   - **- e.g. in above case 3 & 4.**

Following algorithms used for CPU Scheduling:

1. **FCFS (First Come First Served) CPU Scheduling**
2. **SJF (Shortest Job First) CPU Scheduling**
3. **Round Robin CPU Scheduling**
4. **Priority CPU Scheduling**

- Multiple algorithms are there for CPU scheduling, so there is need to decide which algorithm is best suited at specific situation and which algorithm is an efficient one, to decide this there are certain criteria's **called as scheduling criteria's: cpu utilization, throughput, waiting time, response time and turn-around-time.**

CPU Scheduling Criteria's:

1. **CPU Utilization:**
   - Select such an algorithm in which utilization of the CPU must be as **maximum as a possible**.
   - On server systems, CPU utilization should be more than 90%. On desktop systems, CPU utilization should around 70%.
2. **Throughput**:
   - Total work done per unit time. One need to select such an algorithm in which throughput must be **as maximum possible.**
3. **Waiting Time:**
   - It is the total amount of time spent by the process into the ready queue for waiting to get control of the CPU from its time of submission.
   - Select such an algorithm in which waiting time must be as **minimum as possible.**
4. **Response Time:**
   - It is a time required for the process to get first response from the CPU from its time of submission. One need to select such an algorithm in which response time must be as **minimum as possible**

5. Turn-Around -Time:

- It is the total amount of time required for the process to complete its execution from its time of submission.

- One need to select such an algorithm in which turn-around-time must be as **minimum as possible.**
- **Execution Time:** it is the **total amount of time spent by the process onto the CPU to complete its execution**.
  - **OR CPU Burst Time:** total no. of CPU cycles required for the process to complete its execution.
- **Turn-Around-Time = Waiting Time + Execution Time.**
- **Turn-around-time is the sum of periods spent by the process into ready queue for waiting and onto the CPU for execution from its time of submission.**

➢CPU Scheduling

1. FCFS (First Come First Served) CPU Scheduling

- In this algorithm, process which is arrived first into the ready queue gets the control of the CPU first i.e. control of the CPU gets allocated for processes as per their order of an arrival into the ready queue.
- This algorithm is simple to impalement and can be implemented by using fifo queue.
- It is a non-preemptive scheduling algorithm.

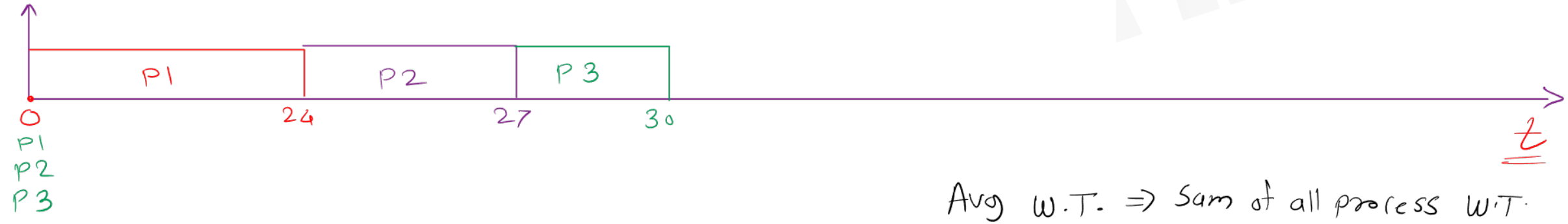
❑**Convoy effect:** in fcfs, due to an arrival of longer process before shorter processes, shorter processes has to wait for longer duration and due to which average waiting time gets increases, which results into an increase in an average turn-around-time and hence overall system performance gets down.

➢FCFS Scheduling

CPU Burst + Waite time

Process	Arrival Time	CPU Burst	Wait Time	Turn Around Time
P1	0	24	0	24
P2	0	3	24	27
P3	0	3	27	30

Gann't chart



$$\text{Avg. T.A.T} \Rightarrow \frac{\text{Sum of all processes T.A.T}}{\text{Num of process}}$$

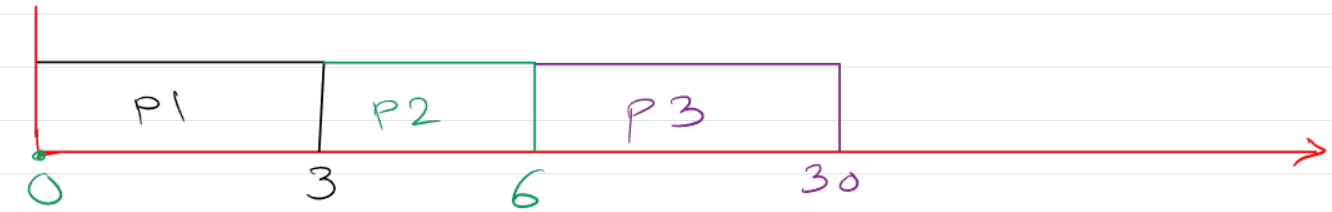
$$\Rightarrow \frac{24+27+30}{3} \Rightarrow \frac{81}{3} \Rightarrow 27$$

$$\text{Avg W.T.} \Rightarrow \frac{\text{Sum of all process W.T.}}{\text{Num of process}}$$

$$= \frac{0+24+27}{3} \Rightarrow \frac{51}{3} \Rightarrow 17$$

FCFS

Process	A.T	C.B.T	W.T	T.A.T
P1	0	3 X	0	3
P2	0	3	3	6
P3	0	24	6	30



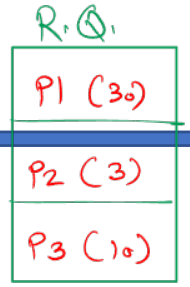
P1
P2
P3

Avg WT $\Rightarrow \frac{0+3+6}{3}$

$= \frac{9}{3} = 3$

Avg. TAT $\Rightarrow \frac{3+6+30}{3}$

$= \frac{39}{3} \Rightarrow \underline{13}$



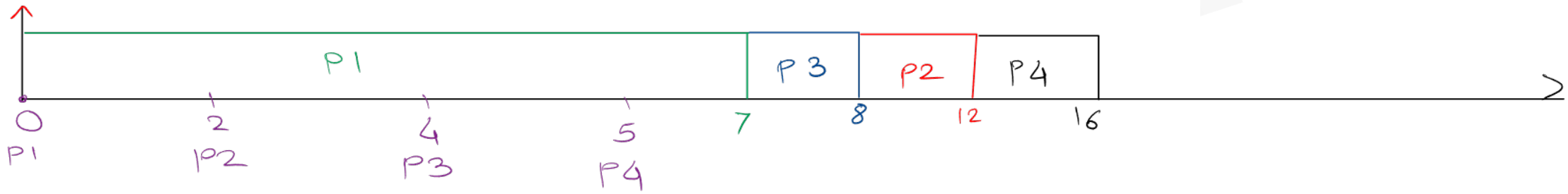
2. SJF(Shortest Job First) CPU Scheduling:

- In this algorithm, process which is having minimum CPU burst time gets the control of the CPU first, and whenever tie is there it can be resolved by using fcfs. - SJF algorithm ensures minimum waiting time.
- Under non-preemptive SJF, algorithm fails if the submission time of processes are not same, and hence it can be implemented as preemptive as well.
- Non-preemptive SJF is also called as SNTF(Shortest-Next-Time-First).
- Preemptive SJF is also called as SRTF(Shortest-Remaining-Time-First).

❑ **Starvation:** in this algorithm, as shorter processes has got higher priority, process which is having larger CPU burst time may gets blocked i.e. control of the CPU will never gets allocated for it, such situation is called as starvation/indefinite blocking.

➢ SJF/SNTF Scheduling (Non-Preemptive)

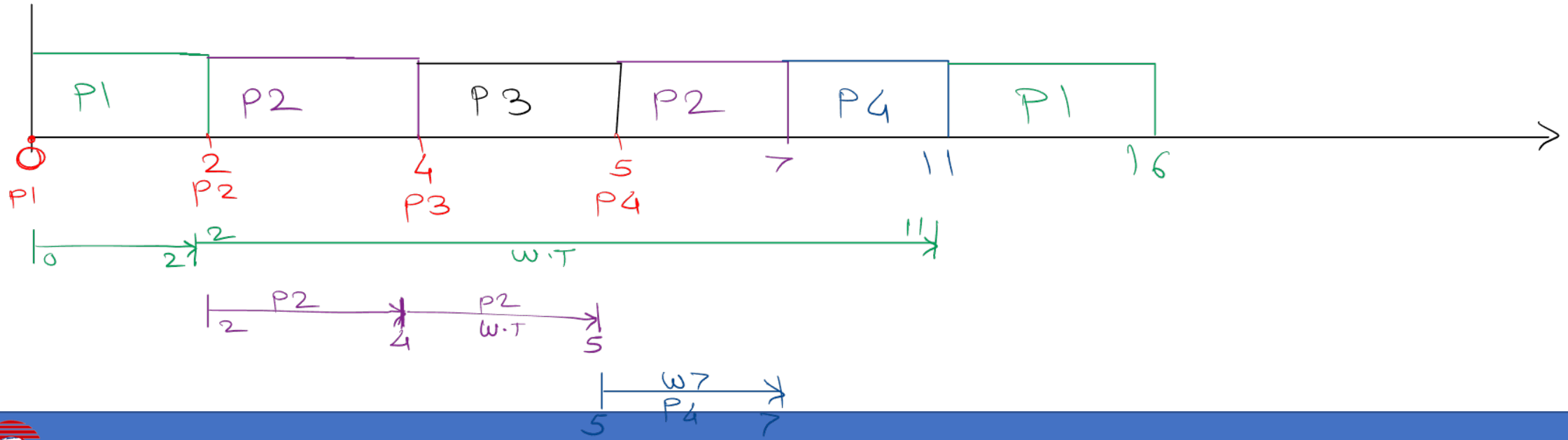
Process	Arrival Time	CPU Burst (Job)	Wait Time	Turn Around Time
P1	0	7	0	7
P2	2	4	6	10
P3	4	1	3	4
P4	5	4	7	11



➢ SRTF Scheduling

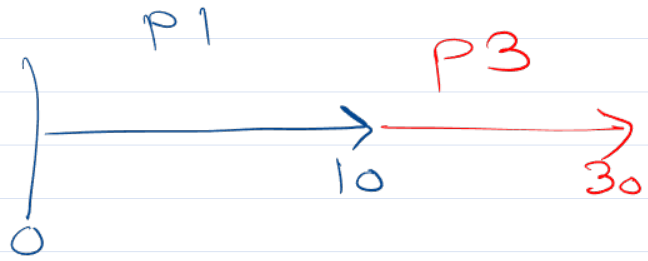
SJF (Preemptive)

Process	Arrival Time	CPU Burst	Remaining Time	Wait Time	Turn Around Time
P1	0	7	5	9	16
P2	2	4	2	1	5
P3	4	1	0	0	1
P4	5	4	4	2	6

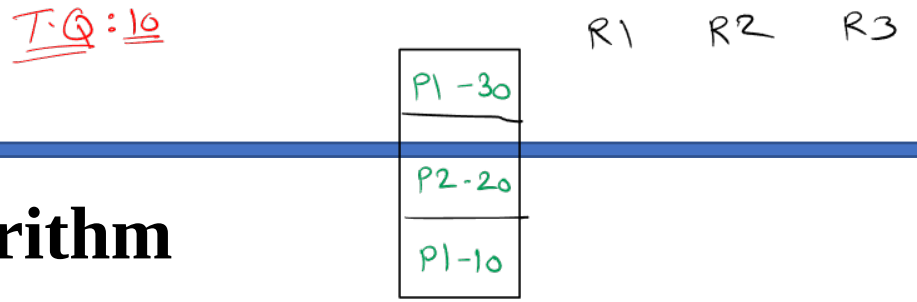


✗ Starvation

	AT	CBT
p1	0	50
p2	0	10
p3	5	20
p4	25	15



Process block continuse due to low priority i.e. starvation.



3. Round Robin Scheduling Algorithm

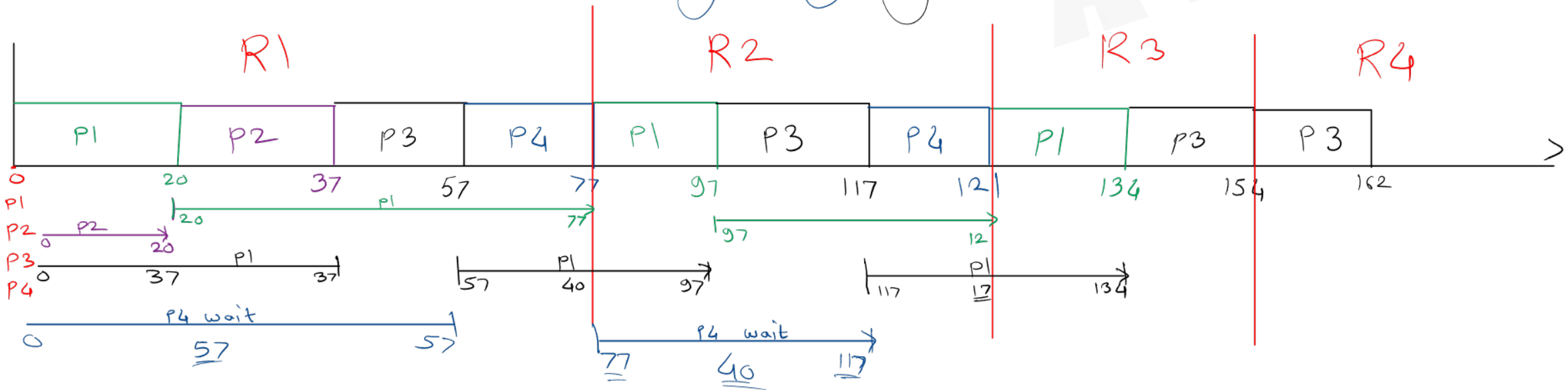
- In this algorithm, before allocating the CPU for processes, **some fixed time slice or time quantum** gets decided in advanced, and at any given time control of the CPU may remains allocated with any process maximum for that decided time-slice, once the given time slice is finished of that process, it gets suspended and control of the CPU will be allocated to the next process again for maximum that decided time slice and so on..., each process gets control of the CPU in a round robin manner i.e. cpu gets shared among processes equally.
- If any process completes its execution before allocated time slice then control of the CPU will be released by that process and CPU gets allocated to the next process as soon as it is completed for effective utilization of the CPU.
- There **is no starvation in RR Scheduling algorithm**.
- This algorithm is **purely preemptive.**
- This algorithm **ensures minimum response time**.
- **If time slice is minimum then there will be extra overhead onto the CPU due to frequent context-switch.**

➢ Round Robin Scheduling

T.Q. = 20

Process	A.T.	CPU Burst	Remaining Time	Wait Time	Response Time
P1	0	53	33 13 X	57+24 => 81	0
P2	0	17 X	X X X	20	20
P3	0	68	48 28 8	37+40+17 => 94	37
P4	0	24	4 X X	57+40 => 97	57

R1 R2 R3 R4

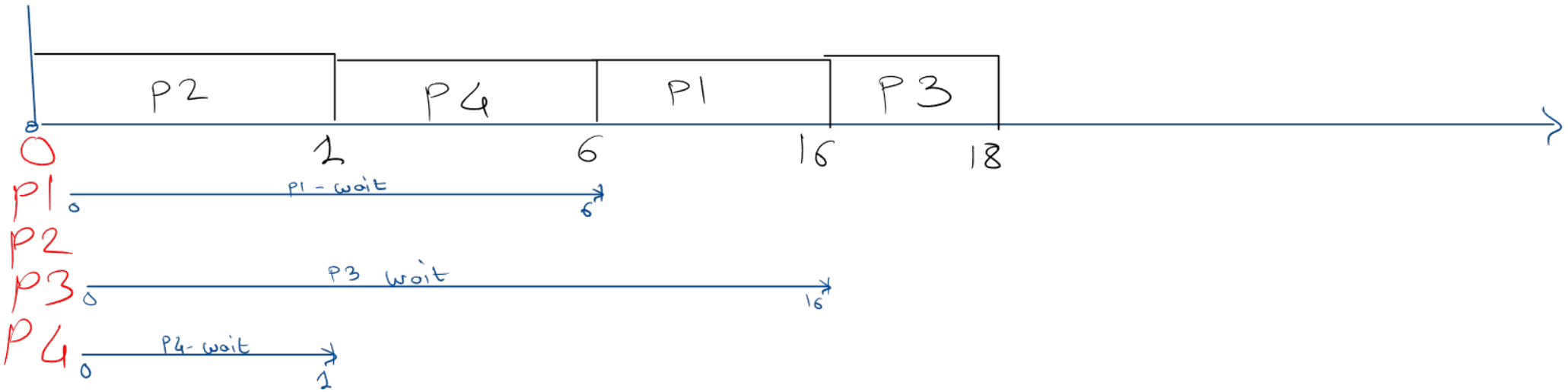


4. Priority Scheduling

- In this algorithm, process which is having highest priority gets control of the CPU first**, each process is having priority in its PCB**.
- priority for a process can be decided by two ways:
    1. **internally** – priority for process can be decided by an OS depends on no. of resources required for it.
    2. **externally** – priority for process can be decided by the user depends on requirement.
- **Minimum priority value indicates highest priority.**
- This algorithm **is purely preemptive.**
- **Due to the very low priority process may gets blocked into the ready queue and control of the CPU will never gets allocated for such a process, this situation is referred as a starvation or indefinite blocking**.
- **Ageing:** it is a technique in which, an OS gradually increments priority of blocked process, i.e. priority of blocked process gets incremented after some fixed time interval by an OS, so that priority of blocked process becomes sufficient enough to get control of the CPU, and starvation can be avoided.

➢ Priority Scheduling

Process	Arrival Time	CPU Burst	Priority	Wait Time
P1	0	10	3	6
P2	0	1	1	0
P3	0	2	4	16
P4	0	5	2	1



➢Multi-level queue :

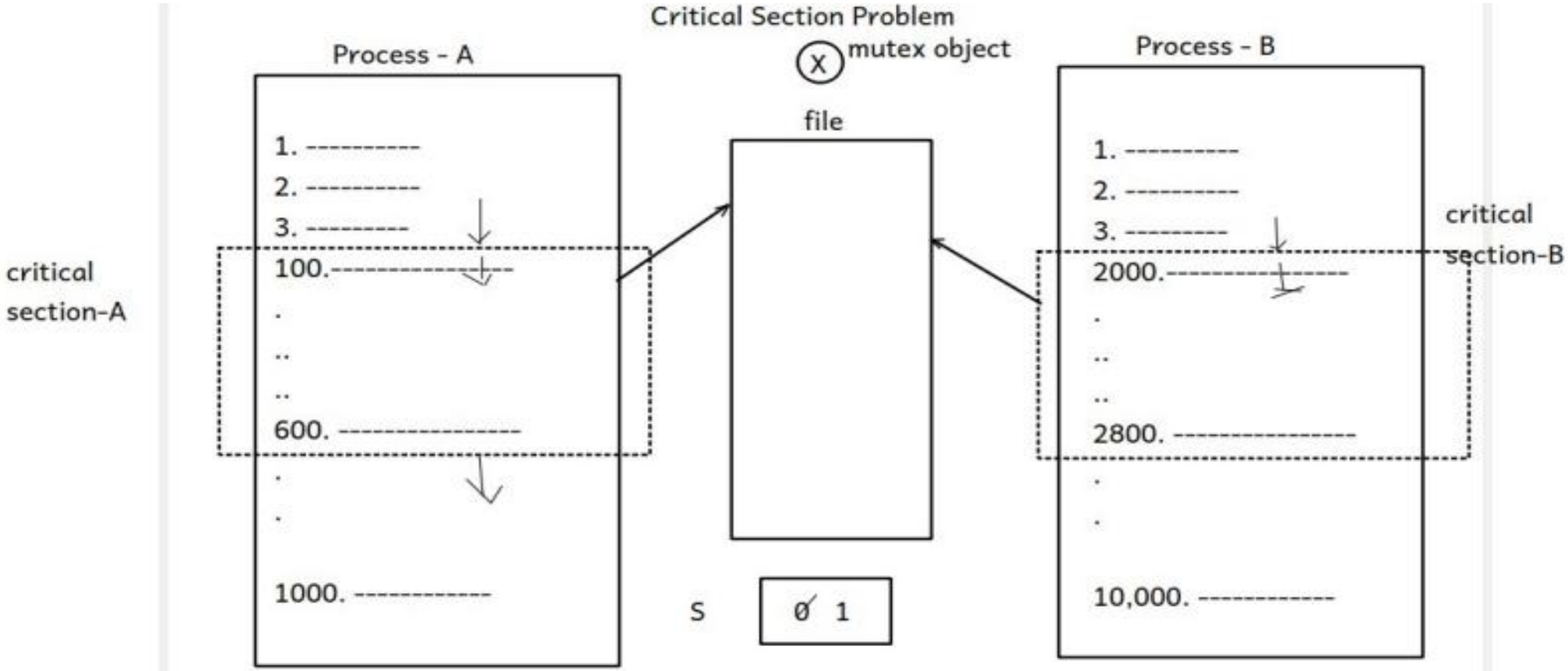
- In modern OS, the ready queue can be divided into multiple sub-queues and processes are arranged in them depending on their scheduling requirements. This structure is called as "Multi-level queue".
- If a process is starving in some sub-queue due to scheduling algorithm, it may be shifted into another sub-queue. This modification is referred as "Multi-level feedback queue".
- The division of processes into sub - queues may differ from OS to OS.

1. Important Services :Priority Scheduling
2. Background Task :SJF
3. GUI Tasks :RR
4. Other Tasks :FCFS

Process Coordination / Process Synchronization

Why Process Co-ordination/Synchronization?

- If concurrently executing co-operative processes are accessing common resources, then conflictions may takes place, which may results into the problem of **data inconsistency**, and hence to avoid this problem coordination / synchronization between these processes is required.
- Multiple processes accessing same resource at the same time, is known as **"race condition".** When race condition occurs, resource may get corrupted (unexpected results).
- **Peterson's problem,** if two processes are trying to modify same variable at the same time, it can produce unexpected results.
- Code block to be executed by only one process at a time is referred as **Critical section**.
- If multiple processes execute the same code concurrently it may produce undesired results.
- To resolve race condition problem, one process can access resource at a time. This can be done using **sync** objects/primitives given by OS.



"data incosistency" problem occure in above case only when both the sections of process A & B are runniing at a same time, and hence these sections are reffered as critical section, and hence data inconsistency problem may occure when two or more processes are running in their critical sections at a same time, and this problem is also reffered as "critical section problem".

➢Synchronization Tools:

1. **Semaphore:**
2. **Mutex :**

➢Semaphore:

- Semaphore was suggested by Dijkstra scientist (dutch math)
- Semaphore is a counter
- Semaphore is a sync primitive given by OS.

❑**On semaphore two operations are supported:**

➢**wait operation: decrement op: P operation:**

1. Semaphore count is decremented by 1.
2. If cnt < 0, then calling process is blocked(block the current process).
3. Typically wait operation is performed before accessing the resource.

➢**signal operation: increment op: V operation:**

1. semaphore count is incremented by 1.
2. if one or more processes are blocked on the semaphore, then wake up one of the process.
3. typically signal operation is performed after releasing the resource.

Q. If sema count = -n, how many processes are waiting on that semaphore?

Answer: "n" processes waiting

• There are two types of semaphore

i. **Binary semaphore :** can be used when at a time resource can be acquired by only one process.

- Allows only 1 process to access resource at a time or used as a flag/condition.

ii. **Counting / Classic semaphore :** can be used when at a time resource can be acquired by more than one processes

2. Mutex Object:

- Mutex is used to ensure that only one process can access the resource at a time.
- Functionally it is same as "binary semaphore".
- Mutex can be unlocked by the same process/thread, which had locked it.

➢ Semaphore vs Mutex

- ✓S: Semaphore can be decremented by one process and incremented by same or another process.
- ✓M: The process locking the mutex is owner of it. Only owner can unlock that mutex.
- ✓S: Semaphore can be counting or binary.
- ✓M: Mutex is like binary semaphore. Only two states: locked and unlocked.
- ✓S: Semaphore can be used for counting, mutual exclusion or as a flag.
- ✓M: Mutex can be used only for mutual exclusion.

Producer-Consumer Problem

- A classic multi-process synchronization problem where two processes share a common, fixed-size buffer used as a queue.
- **The Processes:**
  - **Producer:** Generates data and puts it into the buffer.
  - **Consumer:** Removes data from the buffer and consumes it.
- **Core Challenges:**
  - Prevent the **producer** from adding data when the buffer is **full**.
  - Prevent the **consumer** from removing data when the buffer is **empty**.
  - Prevent both from **accessing the buffer at the same time** to avoid race conditions.
- **The Solution Mechanism:**
  - If the buffer is full, the **producer** must **sleep or discard data**. It is later **woken up** by the consumer.
  - If the buffer is empty, the **consumer** must **sleep**. It is later **woken up** by the producer.
  - This coordination ensures efficient and safe use of the shared buffer.

➢Deadlock:

- There are four necessary and sufficient conditions to occur deadlock / characteristics of deadlock:

1. **Mutual Exclusion:** at a time resource can be acquired by only one process.
2. **No Preemption:** control of the resource cannot be taken away forcefully from any process.
3. **Hold & Wait:** every process is holding one resource and waiting for the resource which is held by another process.
4. **Circular Wait:** if process P1 is holding resource and waiting for the resource held by another process P2, and process P2 is also holding one resource and waiting for the resource held by process P1.

Three deadlock handling methods are there:

1. **Deadlock Prevention:**
   - OS Syscalls are designed so that at least one deadlock condition does not hold true.
   - In UNIX multiple semaphore operations can be done at the same time.

2. **Deadlock Detection & Avoidance:**
   - before allocating resources for processes all input can be given to deadlock detection algorithm in advanced and if there are chances to occur deadlock then it can be avoided by doing necessary changes.

❖**There are two deadlock detection & avoidance algorithms:**

1. **Resource Allocation Graph Algorithm**
2. **Banker's Algorithm**
3. Safe state Algorith

- Processes declare the required resources in advanced, based on which OS decides whether resource should be given to the process or not.
- **Algorithms used for this are:**
  - **Resource allocation graph:**
    - OS maintains graph of resources and processes.
    - A cycle in graph indicate circular wait will occur.
    - In this case OS can deny a resource to a process.
  - **Banker's algorithm:**
    - A bank always manage its cash so that they can satisfy all customers.
  - **Safe state algorithm:**
    - OS maintains statistics of number of resources and number processes.
    - Based on stats it decides whether giving resource to a process is safe or not (using a formula):
      - **Max num of resources required < Num of resources + Num of processes**
        - If condition is true, deadlock will never occur.
        - If condition is false, deadlock may occur

❑Starvation:

- The process not getting enough CPU time for its execution.
- Process is in ready state/queue.
- Reason: Lower priority (CPU is busy in executing high priority process).

❑Deadlock:

- The process not getting the resource for its execution.
- Process is in waiting state/queue indefinitely.
- Reason: Resource is blocked by another process (and there is circular wait).

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com