CT 656
OPERATING SYSTEM

LECTURE – 2
Process Management

Er. Suroj Maharjan

WHERE ARE WE ?

1. Introduction
2. Process Management
3. Process Communication and Synchronizing
4. Memory Management
5. File Systems
6. I/O Management and Disk Scheduling
7. Deadlock
8. Security
9. System Administration

Introduction to Process

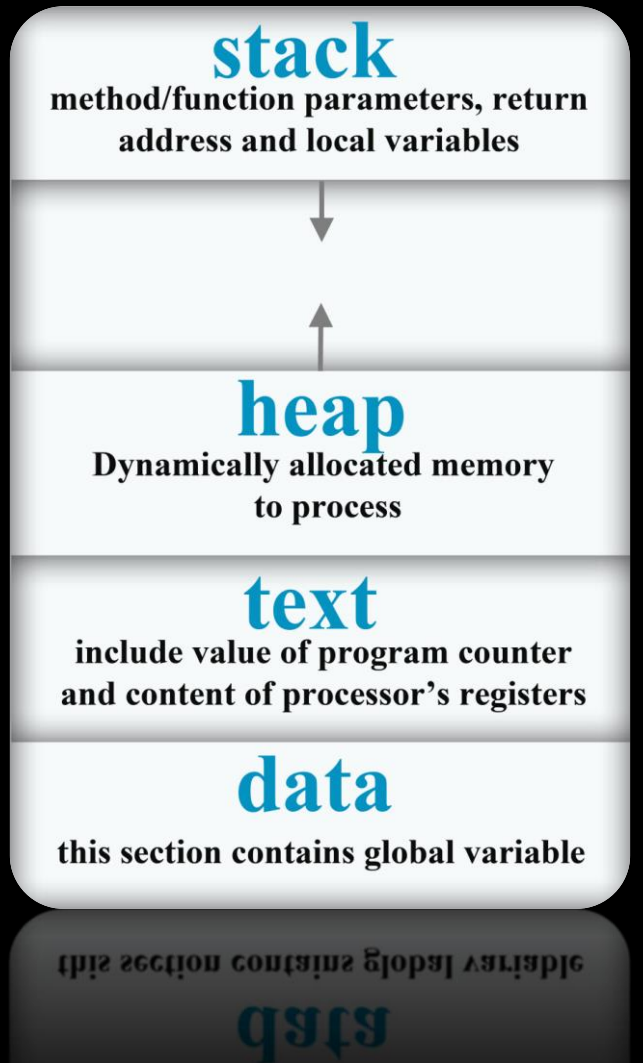
- A process is instance of a program in execution
- A program by itself is not a process
- Program contains the instructions to be executed by processor. A program does not perform any action by itself.
- Program becomes a process when executable file is loaded into memory
- One program can have multiple process. E.g. opening a new tab In a excel or web browser may create a separate process
- Program Vs Process
  - A process is an 'active' entity as opposed to program which is considered to be a 'passive/static' entity
  - A program is loaded in secondary memory devices whereas process is loaded in main memory
  - Time span of program is unlimited, in contrast to a process whose lifespan is limited

Process Memory

- Associated with each process is its address space, a list of memory location from 0 to some maximum, which the process can use
- Process memory is divided into four sections for efficient working
  - The **text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
  - The **data section** is made up the global and static variables
  - The **heap** is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
  - The **stack** is used for storing temporary data (such as method parameters, return addresses, and local variables)

Process Memory

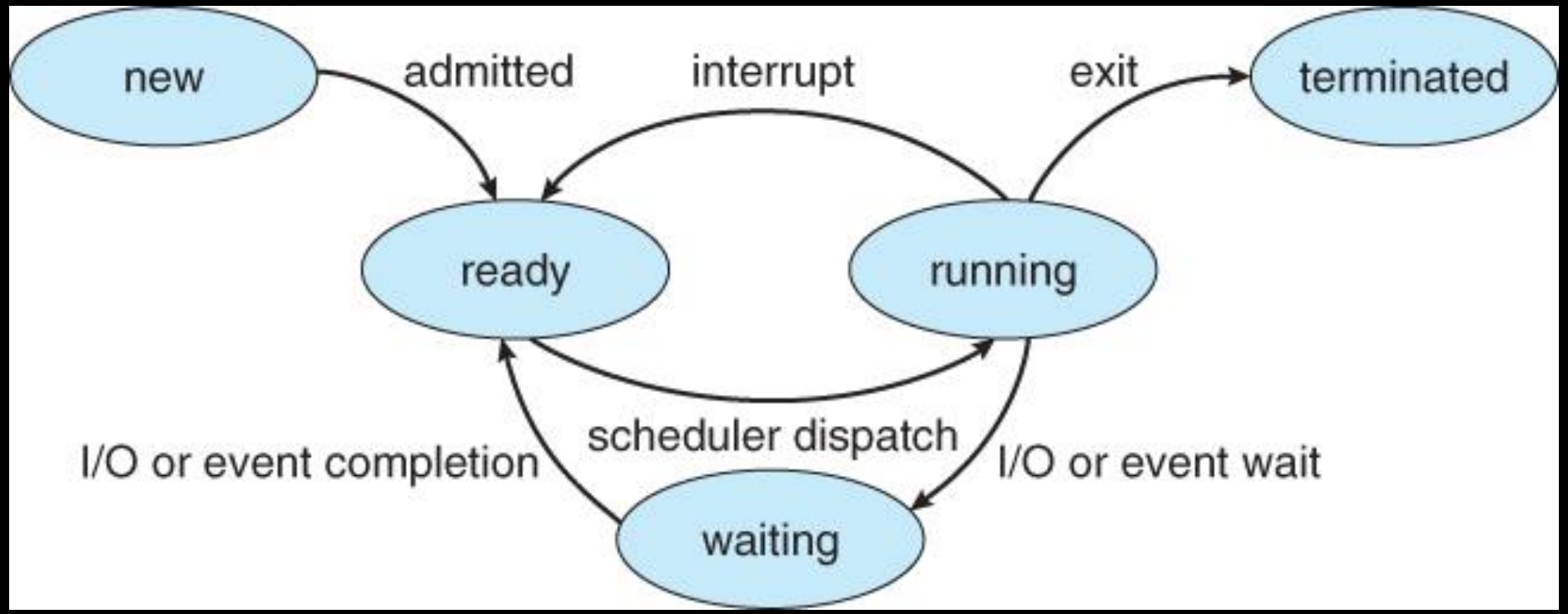
- Note that the stack and the heap start at opposite ends of the process's free space and grow towards each other.
- If they should ever meet, then either a stack overflow error will occur, or else a call to new or malloc will fail due to insufficient memory available.



Process States

- A process can be in any of the following states:
  - **New**
    - process is being created
  - **Ready**
    - the process is ready to be assigned a processor
  - **Running**
    - instructions are being executed
  - **Waiting**
    - the process is waiting for some event to occur (such as an I/O completion or reception of a signal)
  - **Terminated**
    - the process has finished execution
- This is called 5 state process model

Process States

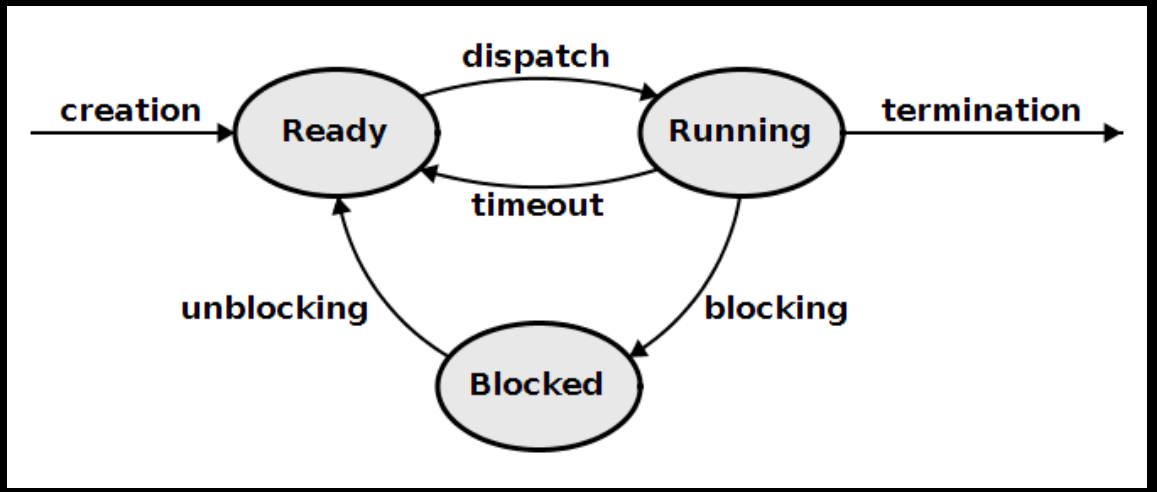


Process States Transitions

- **Null to New** : a new process is created
- **New to Ready** : when OS is ready to take an additional process
- **Ready to Running** : when scheduler/dispatcher selects the process for execution
- **Running to exit** : process completes its operation or if it aborts
- **Running to ready** : process runs out of time slice provided to it, interrupt from a process at high priority in the blocked state, a process may voluntary release control of processor
- **Running to blocked** : if process has to wait for an event like I/O or a resource request
- **Blocked to ready** : whatever the process was waiting for has happened
- **Ready to exit or Blocked to exit** : parent kills child process, or parent terminates causing all of its child to terminate as well
- Note that transition directly from waiting to running is not possible

Process States Models

- We have already seen 5 state model of process
- 2 State Model
  - Running
  - Not Running
- 3 state Model
  - Ready
  - Running
  - Blocked
- 7 state Model (do some research)
  - 5 state model with two suspend states:
    - Ready Suspended
    - Blocked Suspended

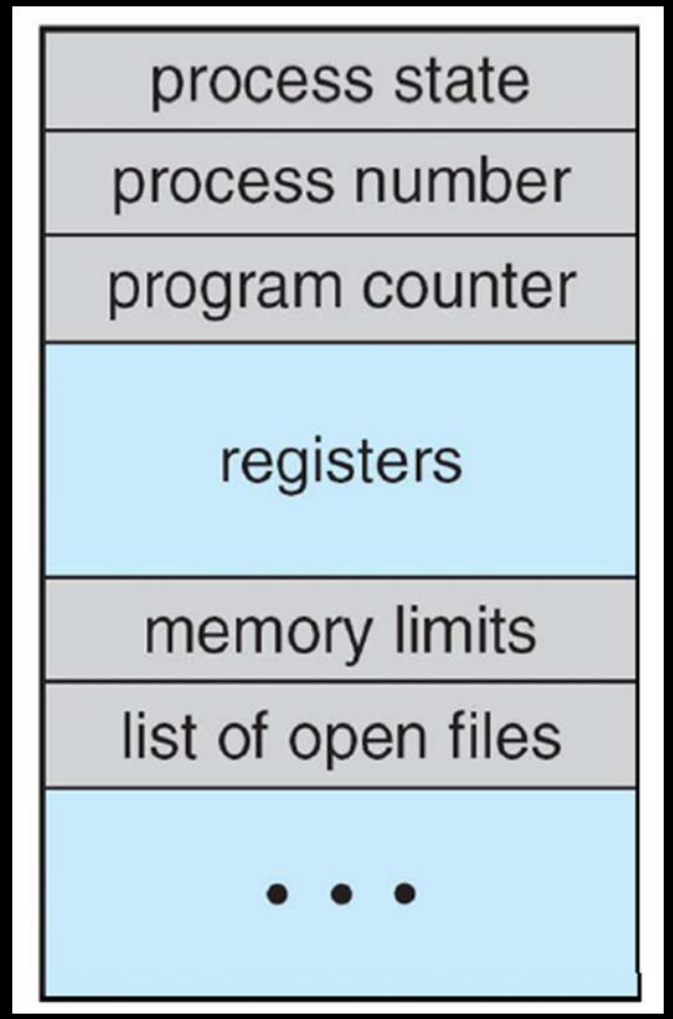


Process Control Block (PCB)

- Each process in an OS is represented by a Process Control Block (PCB) – also called a task control block
- When process is created, the OS creates a corresponding PCB and when it terminates, its PCB is released to the pool of free memory
- A PCB contains many information associated with a process, such as
  - **Process State**
    - indicates the current state of the process
    - could be new, ready, running, waiting etc.
  - **Process ID**
    - It is a unique process number or identifier that identifies each process uniquely
  - **Program counter**
    - contains the address of next instruction to be executed for the process

Process Control Block

- **CPU Registers**
  - vary in number and type depending on computer architecture
  - they include accumulators, index register, stack pointer, and general purpose registers
  - When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out.
  - When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers
- **CPU scheduling Information**
  - it contains process priority, pointers to scheduling queue and any other scheduling parameters



Process Control Block

- **Memory management Information**
  - may include the information as the value of base and limit registers, page tables, or the segment table etc.
- **Accounting Information**
  - includes information about CPU and real time used, time limits, etc.
- **I/O status Information**
  - includes list of I/O devices allocated to the process, list of open files etc.

Operation on process

- Principal operations that can be done on process are
  - Creation
  - Execution
  - Termination
  - Block
  - Wakeup
  - Change Priority

- We shall see the two important operation in detail here, the creation and termination of processes

Process Creation

- Four principal events that causes process to be created

1. **System initialization**
   - Processes are created when OS boots up
   - They may be foreground process or background (daemon process)
   - Foreground processes interact with users
   - Daemons run in background and remains dormant until some event occurs (e.g.. Email, printing etc.)
2. **A user request to create new process**
   - In interactive system, user can create a process by double clicking an icon or by typing a command
   - In UNIX, fork() system call creates new process while in windows it is CreateProcess()

Process Creation

3. **Initiation of a batch job**
   - In batch systems, when OS decides that It has all the necessary resources to run another job in the input queue, it creates a new process and runs the next job in the queue
4. **Execution of a process creation system call, by a running process**
   - A running process can issue system calls to create more processes to help it do its job
   - For example, a process fetching and processing data over a network can create another process so that the old process only does the task of fetching data, keep it on shared buffer and the new process removes the data and processes it, if large amount of data is being fetched

Process Termination

- Sooner or later, a process is terminated, usually from one of the following

1. **Normal exit (voluntary)**
   - Process terminate after they have completed their work
   - Process calls exit() in UNIX and ExitProcess() in Windows to exit normally
2. **Error Exit (voluntary)**
   - process discovers a fatal error and terminates
   - For example, gcc foo.c and the file foo.c does not exist, compiler exits.
3. **Fatal Error (involuntary)**
   - Processes terminate in this case when process causes some error, often due to a program bug
   - For example, referencing non existing memory, dividing by zero etc.

Process Termination

4. **Killed by another process (Involuntary)**
   - A process may execute a system call telling the OS to kill some other process
   - In UNIX this call is kill() and in Windows, TerminateProcess()
   - In any case, the killer needs to have sufficient authorization to perform the kill
   - In some systems, when a process terminates, either voluntary or otherwise, all processes it created are immediately killed as well.

Cooperating Processes

- Two processes are said to be **serial** if execution of one must be complete before the execution of another process can start
- Two processes are said to be **concurrent** if their execution can overlap in time
- These concurrent processes executing in the OS may be independent or cooperating
- A process is **independent** if it cannot affect or be affected by other processes executing in the system
- In other hand, a process is **cooperating** if it can affect or be affected by the other processes executing in the system
- Clearly, any process that shares data with other processes is a cooperating process and that does not is an independent one

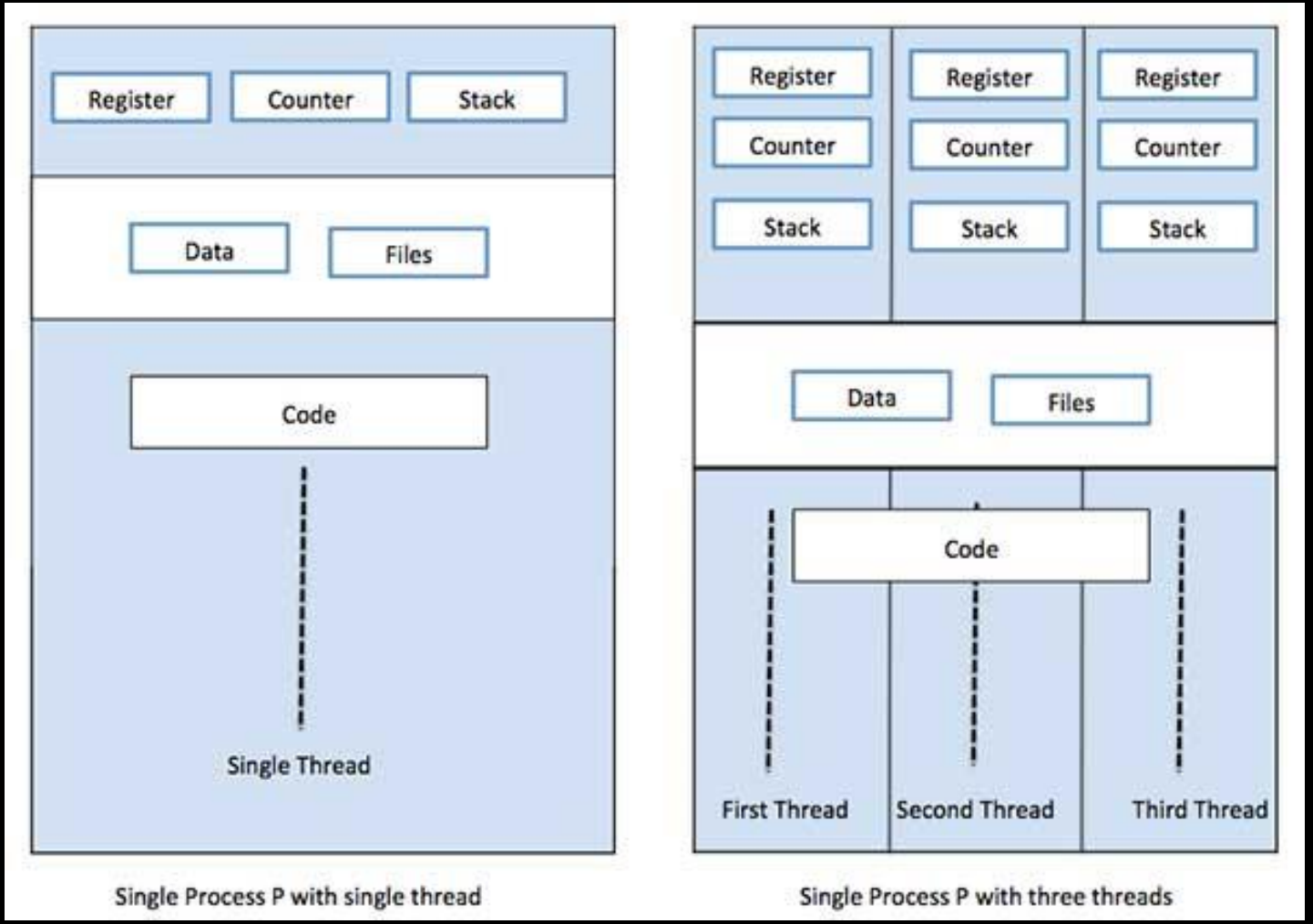
Threads

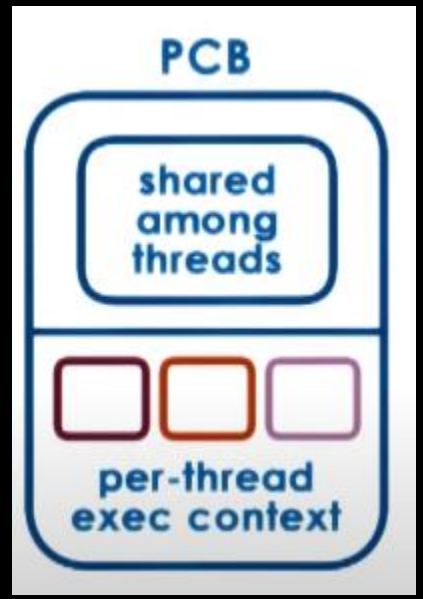
- A thread is a sequential flow of control within a process
- A thread is the smallest sequence of programmed instructions that can be managed independently by a scheduler
- Because threads have some of the properties of processes, a thread is also called Light Weight Process (LWP) or mini process
- A traditional/heavyweight Processes are used to group resources together and threads are the entities scheduled for execution on the CPU
- Threads run within a process utilizing the resources allocated for a process
- process has a single thread of control
- A single process can have multiple threads (multithreaded process)
- If process have multiple thread of control*, it can do more than one task at a time*

Threads

- For example in a **word processor**, a background thread may check spelling and grammar while a foreground thread processes user input (keystrokes), while yet a third thread loads images from the hard drive, and a fourth does periodic automatic backups of the file being edited.
- Another example is **a web server** - Multiple threads allow for multiple requests to be satisfied simultaneously, without having to service requests sequentially or to fork off separate processes for every incoming request.
- Another example is a **web browser** can have a thread to display images or text while another thread retrieves data from the network
- Each individual thread comprises of thread ID, a program counter, register set and a stack
- Multiple thread in a same process share code, data, and certain structures such as open files.

Threads


Register
Counter
Stack
Data
Files
Code
Single Thread
Single Process P with single thread
Register
Counter
Stack
Register
Counter
Stack
Register
Counter
Stack
Data
Files
Code
First Thread
Second Thread
Third Thread
Single Process P with three threads


PCB
shared among threads
per-thread exec context

Benefits of Threads

- Threads are easy and inexpensive to create(only need a stack and storage for registers.) Threads are 10 times as faster to create than process (studies done by Mach developers)
- It takes less time to terminate a thread than a process
- Thread minimizes context switching time (i.e. switching between two threads within the same process is faster). The reason is that we only have to save and/or restore PC, SP and registers
- Use of threads provides concurrency within a process
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency
- Threads use very little resources of an operating system in which they are working. That is, threads do not need new address space, global data, program code or operating system resources.
- Because threads can share common data, they do not need to use inter process communication.

Thread states

- As with processes, the key states for the threads are:

1. Running
2. Ready
3. Blocked

Thread Operation

- **Spawn**: when a process is spawned, a thread within the process is also spawned. Also, a thread may spawn another thread within the same process
- **Block** : when a thread needs to wait for an event, it will block saving its registers, PC & SP
- **Unblock** : the event for which the thread is blocked occurs, thread is moved to the ready queue
- **Finish** : when a thread completes its operation, its register and stacks are de-allocated

Scheduling

- Scheduling is the method by which threads or processes are given access to system resources
- The primary goal of scheduling (*done by schedulers*) is to allocate resources in a way that system meets objectives such as response time, throughput and processor efficiency
- Types of Scheduling:
  1. **Processor Scheduling**
     - refers to determining the order of execution of the processes (i.e., the order in which processes are going to receive the processor and for how long)
     - ready queue is maintained for this
  2. **I/O Scheduling**
     - refers to determining the order in which processes receive the resources
     - a device queue is maintained for this

Types of schedulers

❖**Long term scheduler (Job Scheduler)**

- A long-term scheduler determines which programs are admitted to the system (ready queue) for processing

- decide whether to admit a new process to ready state or not

-invoked very infrequently

❖**Short term scheduler (CPU Scheduler)**

- decides which process gets the processor

-decides which process from ready state is to be moved to running state

- calls the dispatcher after decision is made. Dispatcher gives control of the CPU to the process selected by short term scheduler.

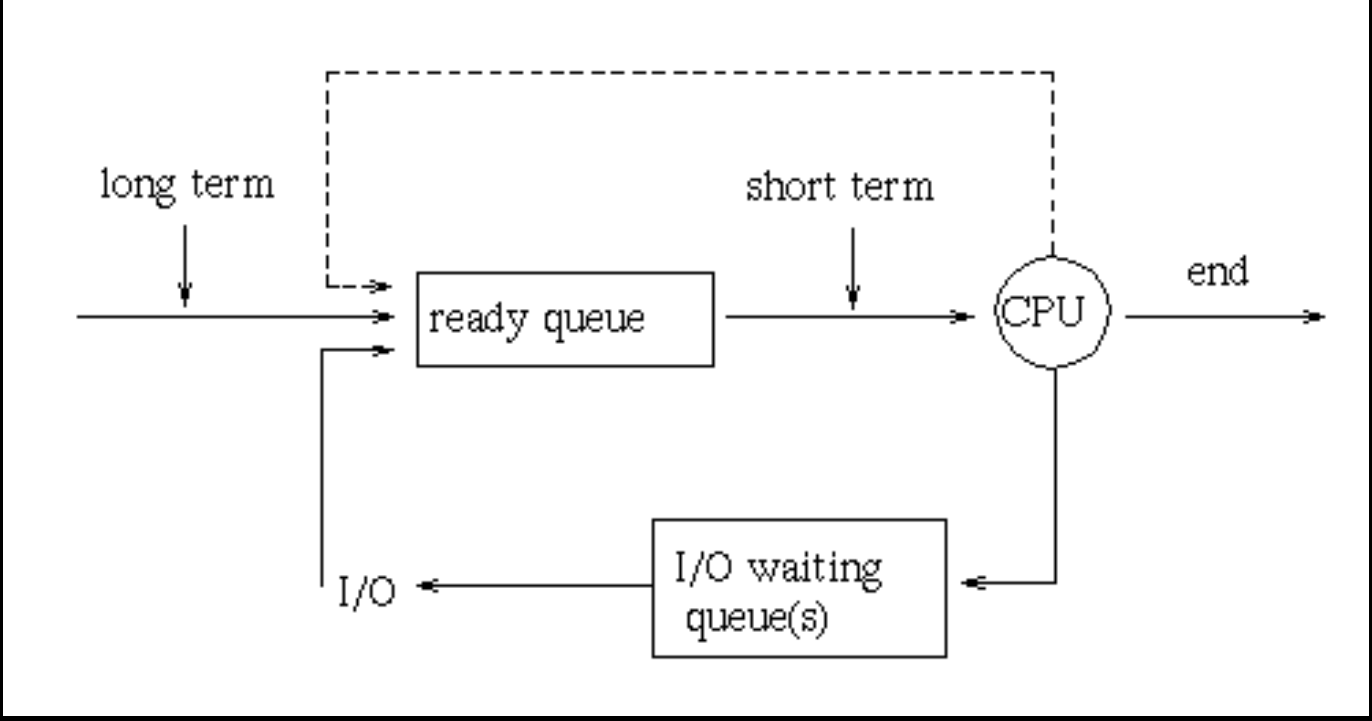
- invoked very frequently

Types of Schedulers

❖**Medium term scheduler**

- Medium-term scheduling is a part of swapping.

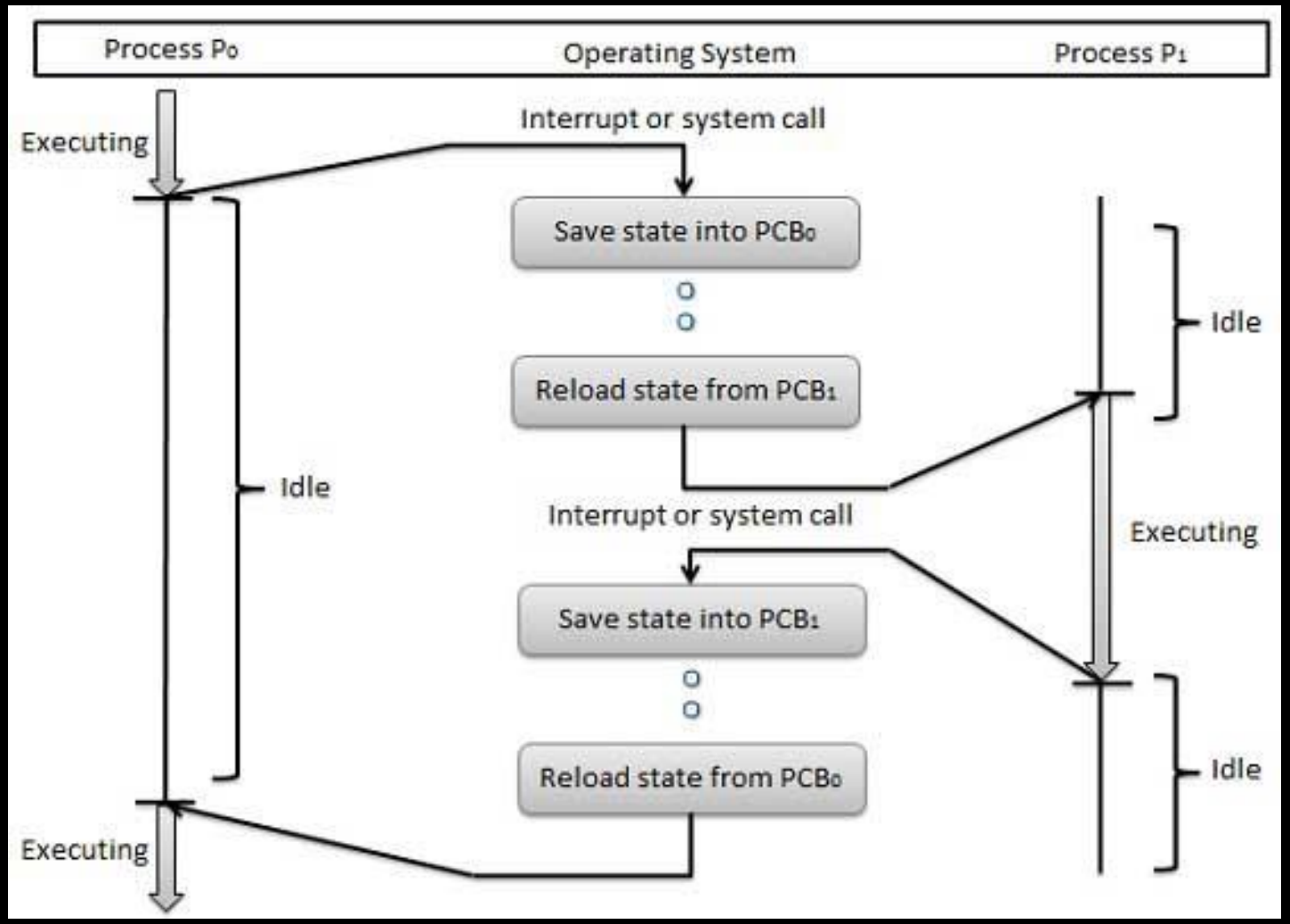
-decide to suspend a process from main memory to secondary memory due to an I/O or event request



Context Switching

- When CPU switches from one process to another,
  - the system must save the state of old process and
  - load the saved state of new process
- Context of a process is represented in the PCB
- Context switching time is overhead, CPU does not do any useful time during this.

Context switching



Process Types

- **I/O Bound Processes**
  - Spends more time doing I/O than computations
- **CPU Bound Processes**
  - Spends more time ding computations
- Long term scheduler strives for good process mix
- The long-term and medium term scheduler control the degree of multiprogramming

Preemptive VS Non preemptive Scheduling

❖**Non preemptive scheduling**

- Once a process is allocated the CPU, it does not leave unless

  a. It terminates

  b. It has to wait for I/O

-Advantage: easy to implement

❖**Preemptive scheduling**

- OS can force/preempt a process to leave the CPU at any time for any reason like interrupt

CPU Scheduling

- Who ?
  - short term scheduler
- Where?
  - ready state to running
- When?
  - when a process moves from
    1. Running to terminated
       Running to Wait
       Running to Ready
    2. New to ready
    3. Wait to ready

Scheduling criteria

- CPU Utilization (keep the CPU as busy as possible): Maximize
- Throughput (no of process complete per unit time): Maximize
- Turn around time (amount of time to execute a particular process) : Minimize
- Waiting Time (amount of time for which a process is in ready queue): Minimize
- Response Time (amount of time it takes from when a request was submitted until the first response (not output) is produced): Minimize

Scheduling Terms

- **Arrival time (AT)**

  -time at which process gets into the ready queue

- **Burst time (BT)**

  - amount of CPU time required by process to be finished

- **Waiting Time (WT)**

  -time for which process has to wait for CPU

- **Completion time (CT)**

  -time at which process finishes

  CT-AT=BT+WT

Scheduling Terms

- **Turn around time (TAT)**

  -difference between Completion time and Arrival time

  TAT=CT-AT=BT+WT

- **Response time (RT)**

  -time at which process gets a processor for the first time

Scheduling Algorithms

❖**Scheduling in Batch systems**

1. First-Come First-served (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time Next (SRT)

❖**Scheduling in interactive systems**

1.Round Robin Scheduling
2. Priority Scheduling
3. Multiple queues
4. Shortest Process Next
5. Guaranteed Scheduling
6. Lottery Scheduling
7. Fair-share Scheduling

Scheduling Algorithms

- **Scheduling in Real Time systems**
  1. Static Scheduling
  2. Dynamic Scheduling
- **Thread Scheduling**
  1. User level thread scheduling
  2. Kernel level thread scheduling
- **Multiprocessor Scheduling**

First Come First Serve (FCFS)

Criteria : AT (select process which arrived first in ready queue)

Mode : Non-preemptive

Process	AT	BT	CT	TAT = CT - AT	WT = TAT-BT	RT
P1	0	20	2	2	0	0
P2	1	3	5	4	1	1
P3	2	5	10	8	3	3
P4	3	4	14	11	7	7
P5	4	1	15	11	10	10
				ΣTAT = 36	Σ WT = 21	Σ RT = 21

Calculations

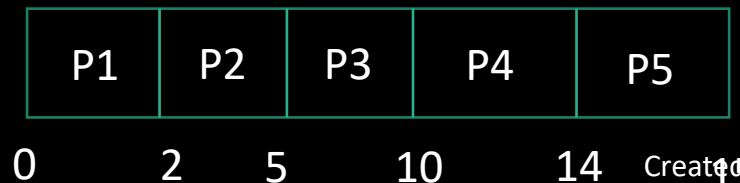
Total processes (n) : 5

$(TAT)_{avg}$ = Σ TAT / n = 36/5 = 7.2

$(WT)_{avg}$ = Σ WT/ n = 21/5 = 4.2

$(RT)_{avg}$ = Σ WT/ n = 21/5 = 4.2

Gantt Chart



First Come First Serve (FCFS)

Process	AT	BT	CT	TAT = CT - AT	WT = TAT - BT
P1	0	4	4	4	0
P2	1	3	7	6	3
P3	2	1	8	6	5
P4	3	2	10	7	5
P5	4	5	15	11	6
				ΣTAT = 34	ΣWT = 19

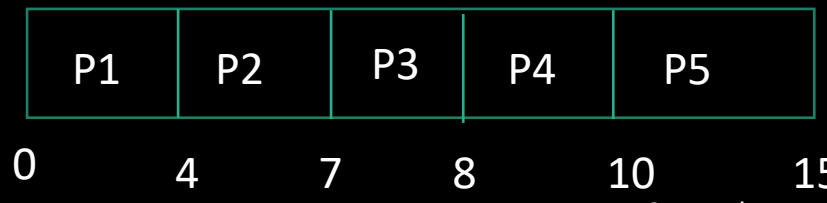
Calculations

Total processes (n) : 5

$(TAT)_{av\ g}$ = Σ TAT / n = 34 /5 = 6.8

$(WT)_{avg}$ = Σ WT/ n = 19/5 = 3.8

Gantt Chart



First Come First Serve (FCFS)

Process	AT	BT	CT	TAT	WT
P1	0	2	2	2	0
P2	3	1	4	1	0
P3	5	6	11	6	0
				ΣTAT = 9	Σ WT = 0

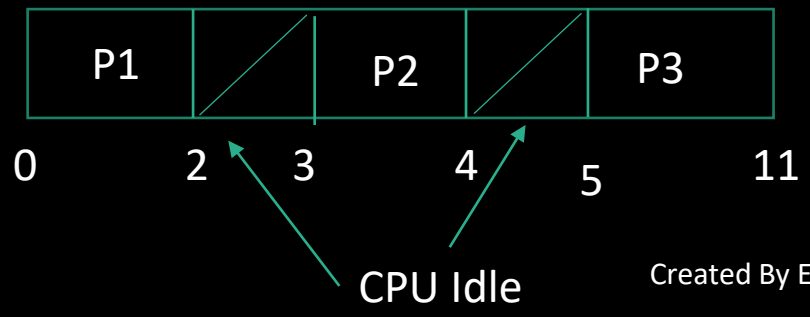
Calculations

Total processes (n) : 3

$(TAT)_{av\ g}$ = Σ TAT / n = 9/3 = 3

$(WT)_{avg}$ = Σ WT/ n = 0/3 = 0

Gantt Chart



Shortest Job First (SJF)

Criteria : BT (select process with least Burst Time in ready queue)

Mode : Non-preemptive

Process	AT	BT	CT	TAT = CT-AT	WT =TAT-BT
P1	0	3	3	3	0
P2	1	4	13	12	8
P3	2	2	5	3	1
P4	4	1	6	2	1
P5	6	2	8	2	0
P6	8	1	9	1	0
				ΣTAT = 23	Σ WT = 10

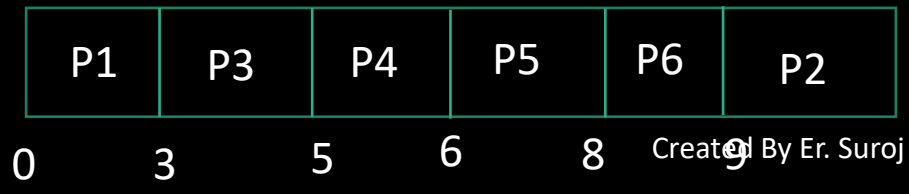
Calculations

Total processes (n) : 6

$(TAT)_{av\ g}$ = Σ TAT / n = 23/6 =3.83

$(WT)_{avg}$ = Σ WT/ n = 10/6 = 1.68

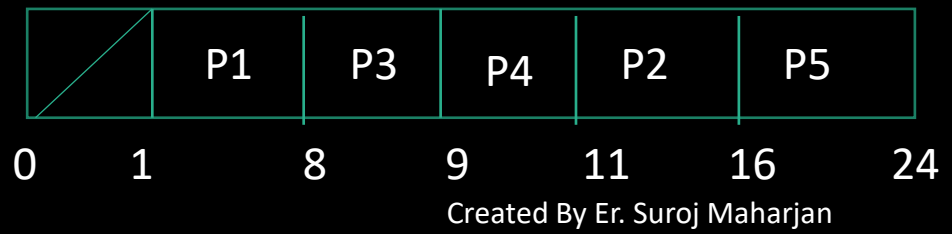
Gantt Chart



Shortest Job First (SJF)

Process	AT	BT	CT
P1	1	7	8
P2	2	5	16
P3	3	1	9
P4	4	2	11
P5	5	8	24

Gantt Chart



Created By Er. Suroj Maharjan

Shortest Remaining Time Next (SRTN)

Criteria : BT (select process with least Burst Time in ready queue)

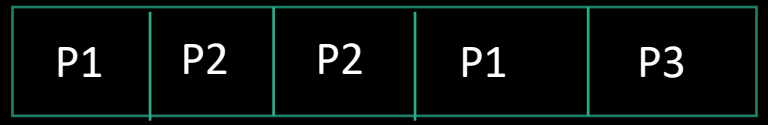
: preempt the process with larger BT

Mode : Preemptive (also called preemptive SJF)

Use FCFS in case of conflict (two or more processes with equal BT/ Remaining time)

Process	AT	BT	CT
P1	0	9	13
P2	1	4	5
P3	2	9	22

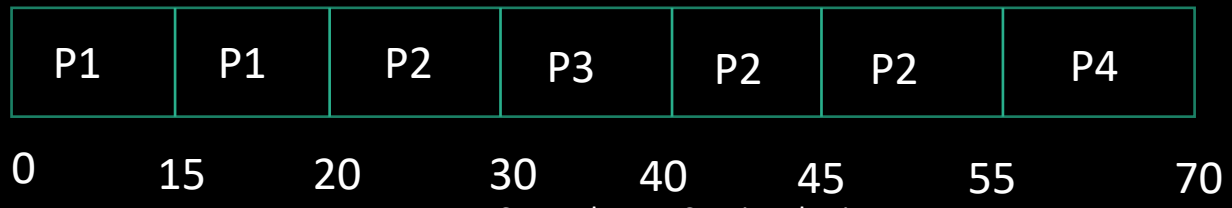
Gantt Chart



Shortest Remaining Time Next (SRTN)

Process	AT	BT	CT
P1	0	20	20
P2	15	25	55
P3	30	10	40
P4	45	15	70

Gantt Chart



Shortest Remaining Time Next (SRTN)

Process	AT	BT	CT
P1	8	1	9
P2	5	1	7
P3	2	7	21
P4	4	3	11
P5	2	8	29
P6	4	2	6
P7	3	5	15

Round Robin Scheduling (RR)

Criteria : AT + Time Quantum (TQ)

Mode : Preemptive

Algorithm

1. Select the process from front of the queue and execute it for fixed time quantum(TQ)
2. After the TQ expires,
   2.1 Enlist new processes (if any) at back of the queue with respect to their arrival time
   2.2 If the current process is not complete, put it at the back after the new processes
3. Repeat 1-2 until the queue is empty

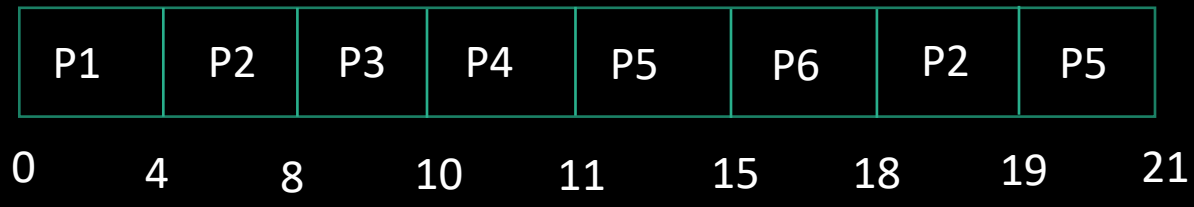
Special case

1. If a process's BT or remaining time is less than the TQ it should get, finish the process and start the new process (do not wait for the remaining TQ). Also, the next process wont get remaining TQ of old process

Round Robin Scheduling(RR)

Process	AT	BT	CT
P1	0	4	4
P2	1	5	19
P3	2	2	10
P4	3	1	11
P5	4	6	21
P6	6	3	18

Gantt Chart



Time Quantum : 4

Front

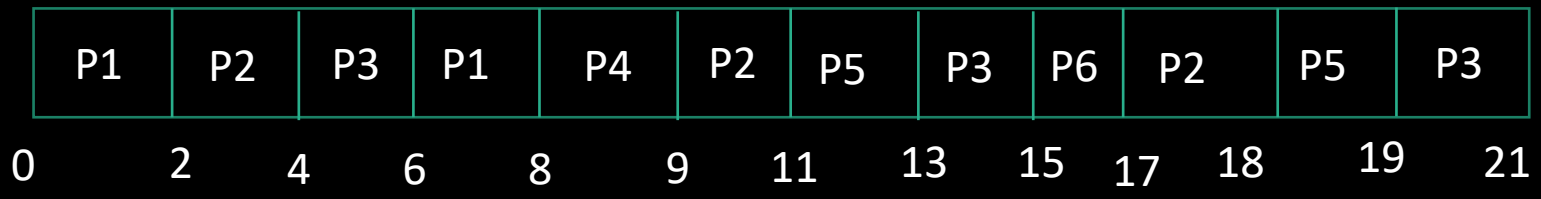
Ready Queue

Created By Er. Suroj Maharjan

Round Robin Scheduling(RR)

Process	AT	BT	CT
P1	0	4	8
P2	1	5	18
P3	2	6	21
P4	4	1	9
P5	6	3	19
P6	7	2	17

Gantt Chart



Time Quantum : 2

Front

Ready Queue

Round Robin Scheduling(RR)

Process	AT	BT	CT
P1	0	2	3
P2	1	5	11
P3	2	1	4
P4	3	8	16

Time Quantum : 1

Process	AT	BT	CT
P1	0	4	8
P2	1	5	18
P3	2	2	6
P4	3	1	9
P5	4	6	21
P6	6	3	19

Time Quantum : 2

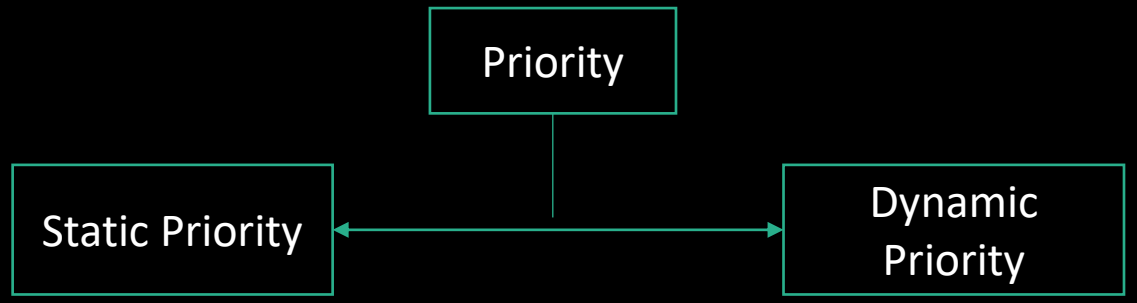
Process	AT	BT	CT
P1	5	5	32
P2	4	6	27
P3	3	7	33
P4	1	9	30
P5	2	2	6
P6	6	3	21

Time Quantum : 3

Created By Er. Suroj Maharjan

Priority Scheduling Algorithms

- Each process is assigned a priority when it arrives at ready queue
- Priority is simply a number associated with each process which can be decided on basis of memory requirement, time requirement etc.
- The process with highest priority in ready queue is allowed to run
- Processes with same priority is run on FCFS basis



- Priority does not change throughout the execution of process
- May cause starvation (low priority processes may never execute)

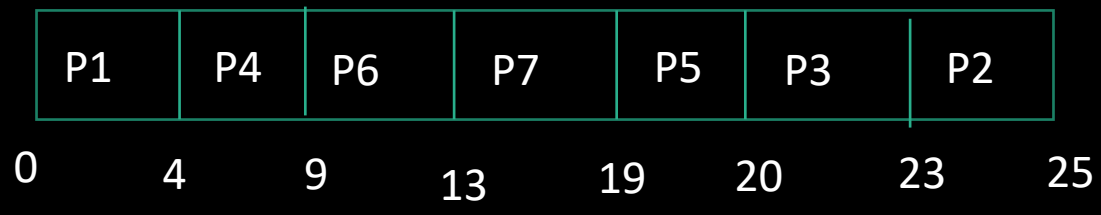
- Priority changes at regular interval (aging)
- decreases the problem of starvation

Priority Scheduling Algorithms

Non Preemptive static priority scheduling

Process	AT	BT	Priority
P1	0	4	2
P2	1	2	4
P3	2	3	6
P4	3	5	10
P5	4	1	8
P6	5	4	12
P7	6	6	9

Gantt Chart



Note: 12 is highest priority

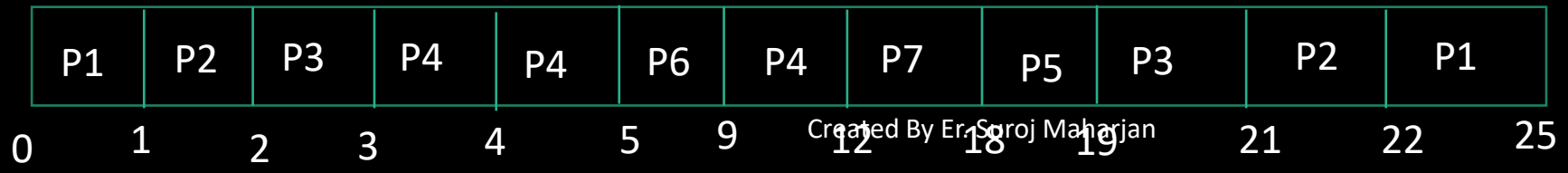
Priority Scheduling Algorithms

Preemptive static priority scheduling

Process	AT	BT	Priority
P1	0	4	2
P2	1	2	4
P3	2	3	6
P4	3	5	10
P5	4	1	8
P6	5	4	12
P7	6	6	9

Note: 12 is highest priority

Gantt Chart



Priority Scheduling Algorithms

- Assume the processes arrived in the order Pl, P2,P3, P4 and P5 all at time 0, priority 1 as highest and 4 as lowest.

Process	BT	Priority	CT
P1	10	3	13
P2	1	1	1
P3	2	3	15
P4	1	4	16
P5	2	2	3

Priority Scheduling Algorithms

Process	AT	BT	Priority	CT
A	0	12	1	55
B	2	8	2	39
C	5	7	4	14
D	3	9	3	23
E	4	6	2	45
F	8	5	1	60
G	7	7	3	32
H	3	4	4	7
I	4	2	3	25

Assume 4 is highest priority

Gantt Chart

Highest Response Ratio Next (HRRN)

Criteria : Response Ratio (RR)

$$RR = \frac{w+s}{s}$$

where

s : service time (Burst Time)

w: waited time so far

Mode : Non-Preemptive

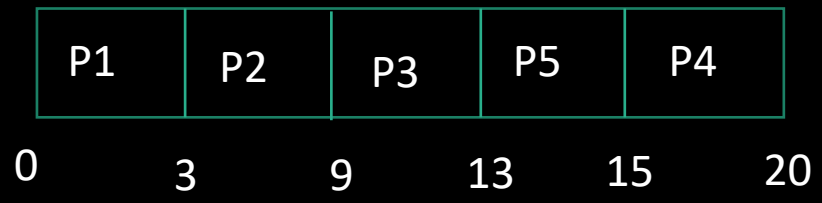
Algorithm : Select process with highest RR to execute

- HRRN not only favors shorter jobs but also limits the waiting time of longer jobs

Highest Response Ratio Next (HRRN)

Process	AT	BT	CT
P1	0	3	3
P2	2	6	9
P3	4	4	13
P4	6	5	20
P5	8	2	15

Gantt Chart



At 9, calculate RR of P3, P4 and P5

$RR_{p3} = \frac{w+s}{s}$ = (5+4)/4 = 2.25

$RR_{p4} = \frac{w+s}{s}$ = (3+5)/5 = 1.6

$RR_{p5} = \frac{w+s}{s}$ = (1+2)/2 = 1.5

At 13, calculate RR of P4 and P5

$RR_{p4} = \frac{w+s}{s}$ = (7+5)/5 = 2.4

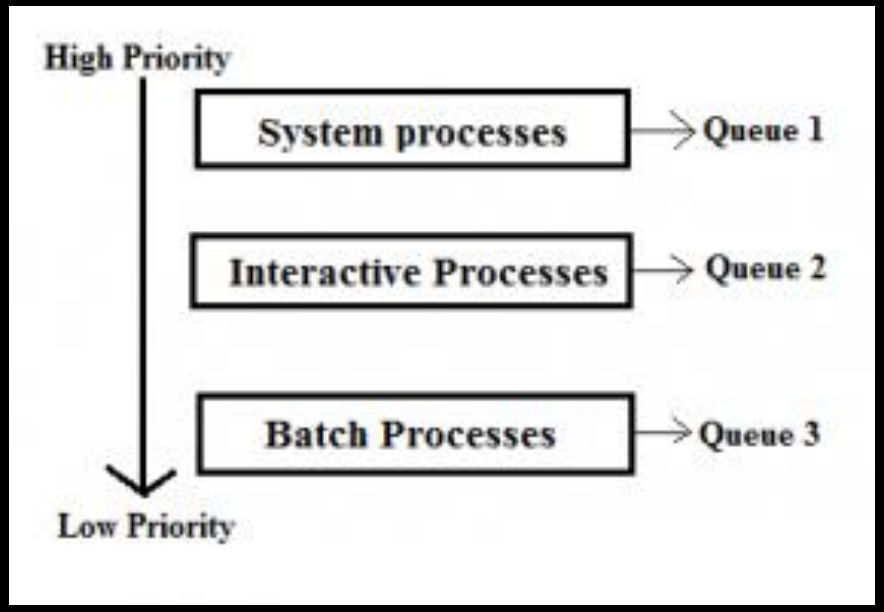
$RR_{p5} = \frac{w+s}{s}$ = (5+2)/2 = 3.5

Highest Response Ratio Next (HRRN)

Process	AT	BT	CT
A	0	12	12
B	2	8	20
C	5	7	27
D	10	9	36

Multi-Level Queue Scheduling (MLQ)

- Processes are divided into different groups resulting in partitioning the ready queue into several separate queues with one queue for a specific process group.
- The processes are permanently assigned to one queue, generally based on some property like process type, memory requirements etc.
- Each queue may have their own scheduling algorithms



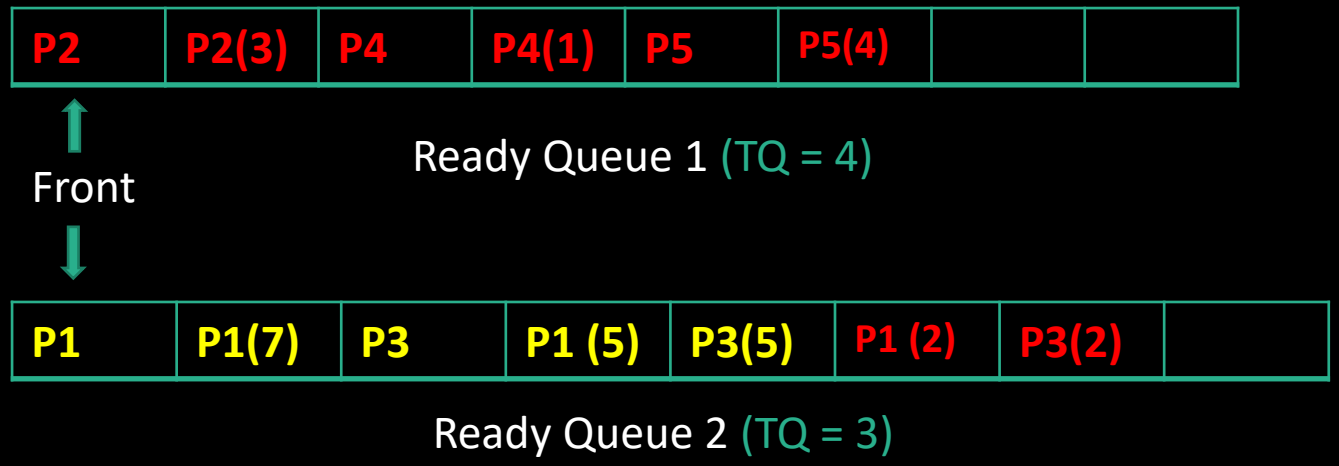
Multi-Level Queue Scheduling (MLQ)

- Each queue has absolute priority over lower queues. For example, no process in the batch queue could run unless the queue for system processes and interactive processes were all empty.
- This means there may be problem of starvation among lower priority queues in this scheduling
- A possible solution may be to time slice among queues. Here each queue gets a certain portion of CPU time which it can schedule amongst its various processes

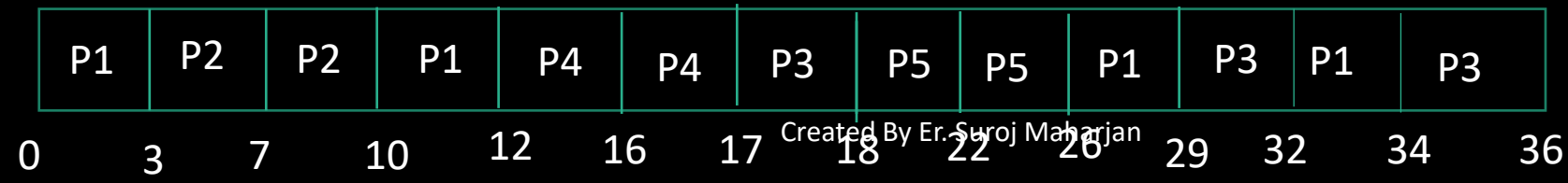
Multi-Level Queue Scheduling (MLQ)

- Apply MLQ scheduling for following set of processes of two queues Q1 and Q2. Both queue uses RR with TQ 4 and 3 respectively. Q1 has higher priority

Process	AT	BT	Queue
P1	0	10	2
P2	3	7	1
P3	4	6	2
P4	12	5	1
P5	18	8	1



Gantt Chart



Multi-Level Queue Scheduling (MLQ)

- Apply MLQ scheduling for following set of processes of two queues Q1 and Q2. Both queue uses RR with TQ 4 and 3 respectively. Q1 has higher priority

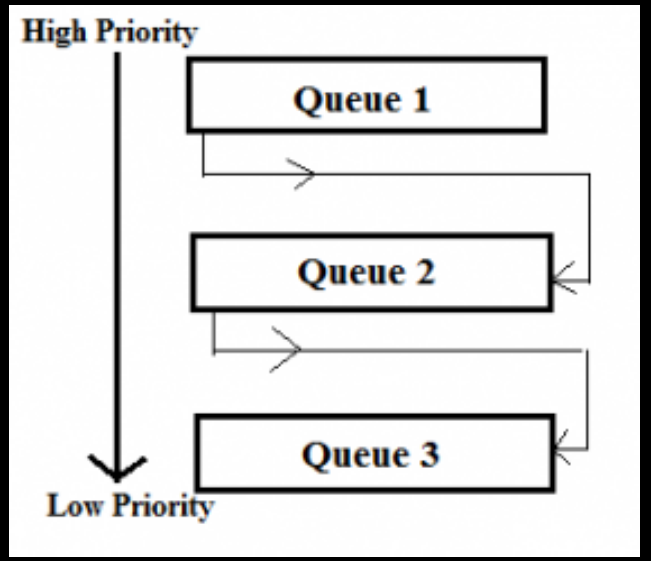
Process	AT	BT	Queue	CT
P1	0	9	2	33
P2	3	7	1	10
P3	4	7	2	34
P4	10	5	1	15
P5	17	6	1	23

Multi-Level Feedback Queue (MLFQ)

- Similar to MLQ but allows the processes to move between queues
- Idea is to separate processes according to characteristics of their CPU bursts
- If a process uses too much CPU time, it will be moved to a lower priority queue which leaves I/O bound and interactive processes in the higher priority queue
- In addition, a process that waits too long in the lower priority queue may be moved to a higher priority queue. This form of aging prevents starvation
- In general, a MLFQ scheduler is defined by following parameters
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade a process to a higher priority queue or demote a process to a lower priority queue
  - Method used to determine which queue a process will enter when it needs to run

Multi-Level Feedback Queue (MLFQ)

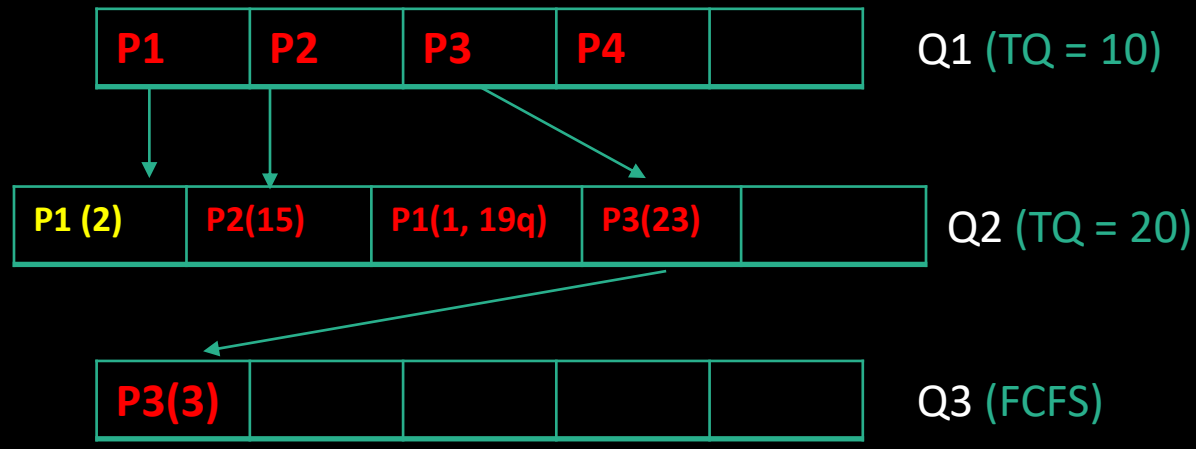
- Consider a MLFQ with queue numbered 1, 2 and 3
- The scheduler first executes processes in the Q1
- Processes in Q2 will be executed only if no processes are in Q1
- Processes in Q3 will be executed only if no processes are in Q1 & Q2
- A process entering ready queue is put in Q1.
- Say, Q1 employs RR with TQ 8, Q2 employs RR with TQ 16 and Q3 FCFS
- A process in Q1 is given time quantum of 8ms.
- If it does not finish within this time, it is moved to Q2
- If queue 1 is empty, the process at the head of Q2 is given a time quantum of 16ms.
- If it does not complete, it is preempted in Q3
- Processes in Q3 are run in FCFS basis when Q1 and Q2 are empty



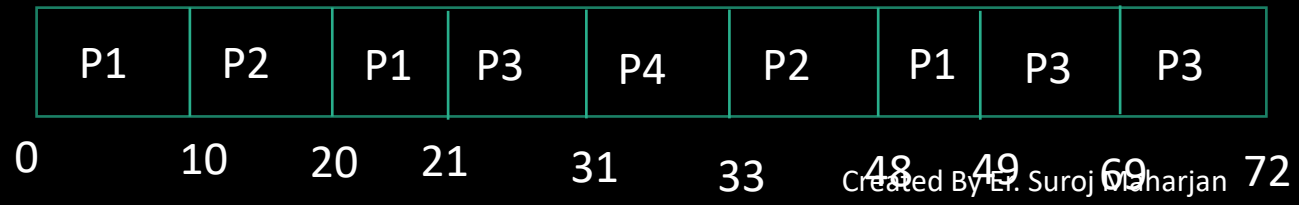
Multi-Level Feedback Queue (MLFQ)

- Apply MLFQ scheduling to following set of processes consisting of three queues Q1, Q2 and Q3. Q1 uses RR with TQ 10 , Q2 uses RR with TQ 20 and Q3 uses FCFS. Q1 has higher priority and Q3 has lowest priority

Process	AT	BT	CT
P1	0	12	49
P2	8	25	48
P3	21	33	72
P4	30	2	33



Gantt Chart



Multi-Level Feedback Queue (MLFQ)

- Apply MLFQ scheduling to following set of processes consisting of three queues Q1, Q2 and Q3. Q1 uses RR with TQ 5ms , Q2 uses RR with TQ 10ms and Q3 uses FCFS. Q1 has higher priority and Q3 has lowest priority

Process	AT	BT	CT
P1	0	25	54
P2	12	18	57
P3	25	4	29
P4	32	10	44

Lottery Scheduling

- The basic idea in lottery scheduling is to give lottery tickets for various system resources such as CPU time
- When scheduling decision has to be made, a lottery ticket is chosen at random and process holding that ticket gets the resource
- When applied to CPU scheduling, system might hold a lottery 50 times a second with each winner getting 20 millisecond of CPU time as a prize
- More important process can be given extra tickets to increase their odds of winning (priority)

Fair Share Scheduling

- Till this point, we have only considered multiple processes but not multiple users (i.e. owner of the processes)
- If user 1 starts 9 processes and user 2 starts 1 process, then with RR (Round Robin), user 1 gets 90% of CPU and user 2 will get only 10 % of CPU. This does not seem fair
- To avoid this, system must take into account who owns the process before scheduling it
- In this model, each user is allocated some fraction of CPU and scheduler picks processes in such a way to enforce it

Fair Share Scheduling

- For example:
  - User 1 creates processes A, B, C, D
  - User 2 creates process E
  - Deal: 50 % of CPU to both
- If RR schedule is used a possible scheduling sequence could be

  A E B E C E D E A E B E ……

- If user 2 is promised twice as much CPU as User 1, in RR, the scheduling sequence could be

  A B E C D E A B E C D E…….

Thread Scheduling

- When several processes have multiple threads, we have two levels of parallelism present: process and threads
- Scheduling in such system differs substantially on whether user level threads or kernel level threads are supported

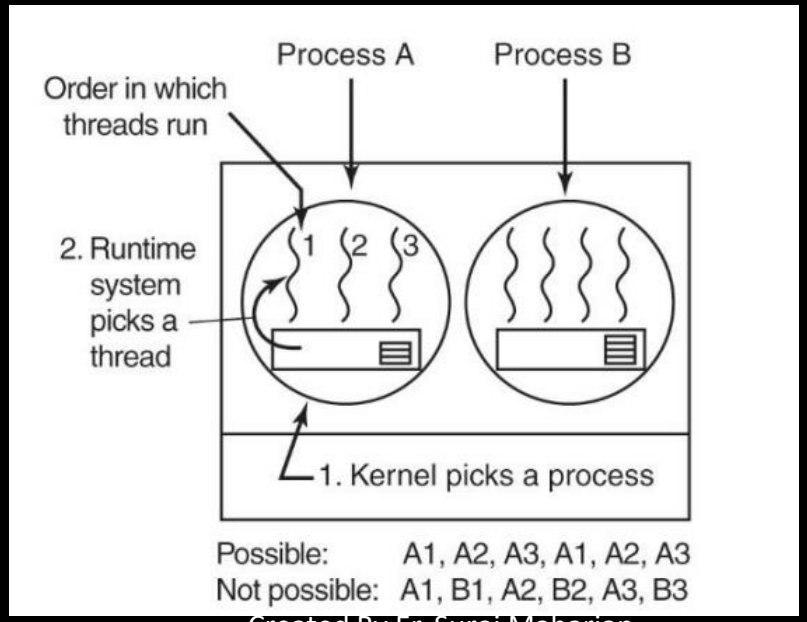
Types:

1. User level Thread Scheduling
2. Kernel Level Thread Scheduling

Thread Scheduling

User Level Thread Scheduling

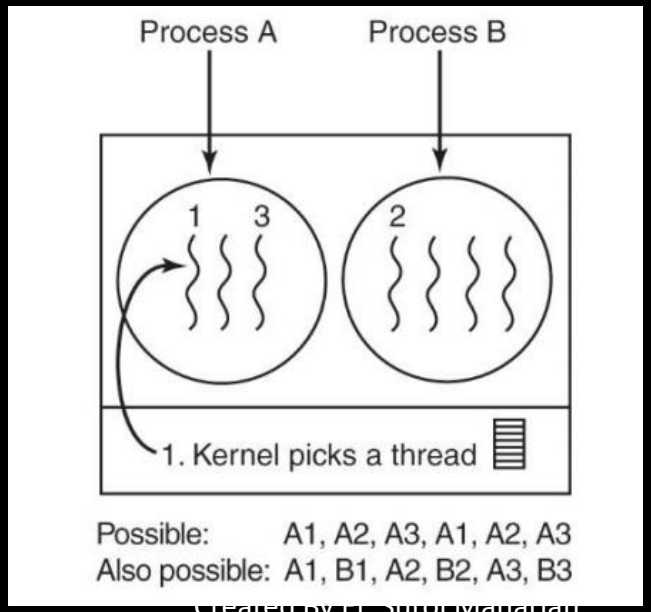
- Kernel is not aware of existence of threads so its operates as usual giving each process only a quantum
- Thread scheduler inside the process decides which thread to run



Thread Scheduling

Kernel Level Thread Scheduling

- Kernel is aware of existence of threads so it is responsible to schedule the threads
- Kernel picks a particular thread to run and is given a quantum to run
- Kernel does not have to take into account which process the thread belongs to but can if it wants to



Scheduling in Real Time System

1. Static Scheduling

- Make scheduling decisions before system starts running
- Only works when there is perfect information available in advance about the work to be done and deadlines to be met

2. Dynamic Scheduling

- Makes scheduling decisions at run time
- Works well if the work to be done and deadlines are not known prior to starting system

Scheduling in Real Time System

1. Rate Monotonic (RM) Scheduling

- Uses two parameters for process
  - Burst Time / Completion Time
  - Period of process (The time after which a process needs to be re-executed)
- Criteria : Highest priority ready process runs
- Mode: Preemptive
- RM Priority:
  - static/ fixed
  - Process with shortest period gets highest priority (i.e. priority inversely proportional to period)
  - Break ties arbitrarily

Scheduling in Real Time System

- Schedule the following processes using RM algorithm

Process	AT	BT	Period
P1	0	2	10
P2	0	1	5
P3	0	5	30
P4	0	2	15

LCM (10, 5, 30, 15) = 30
HCF (10, 5, 30, 15) = 5

Gantt Chart

0 1 3 5 6 10 11 13 14 15 16 18 20 21 23 25 26 30

P1 P2 P3 P4
P2 P3
P1 P2 P3(1)
P2 P4
P1 P2
P2
P1 P2 P3 P4

Scheduling in Real Time System

- Schedule the following processes using RM algorithm

Process	AT	BT	Period
P1	0	3	20
P2	0	2	5
P3	0	2	10

Scheduling in Real Time System

2. Earliest Deadline First (EDF)

- Criteria : Priority
- Mode: Preemptive
- Uses Dynamic priority

- EDF priorities are determined by distance to deadline (i.e. task closest to deadline has the highest priority)

Scheduling in Real Time System

- Schedule the following processes using EDF algorithm

Process	BT	Period	Deadline
P1	3	20	7
P2	2	5	4
P3	2	10	8

Gantt Chart

0 2 5 7 9 10 12 14 15 17 20

P1-7
P2-4
P3-8

P2-9
P3-8

P2-14
P3-18

P2-19

P1
P2
P3

End of Chapter 2