

Operating Systems: CSE 3204

Chapter Two Process Management

Lecture 4: Process Scheduling

11/6/24

1

Basic Concepts

- In a single process/single processor system, only one process can run at a time
- If there are multiple processes in the system, some processes must wait until the CPU is free and can be allocated to them.
- At an instance of time, multiple processes may be willing to occupy the CPU (just after the currently running process yields the control of CPU).
- Now, which process is going to get the CPU next to the current process finish its turn is very important question from the point of view of not only keeping system functioning, but we will see that the decision regarding selection of next process can be made in many ways and can affect performance of the system.

Process scheduling

- The particular way in which a process is selected from a queue of processes in order to assign the CPU is called a **CPU scheduling algorithm**.
- While a CPU scheduling algorithm assigns the processor(s) to process(es), the general problem of **process scheduling** is little bit broader, in which we study about keeping processes in different queues, in memory, in swap area or at CPU, keeping eye on system performance.

CPU Scheduling

- Almost all resources are scheduled before use
- CPU is one of the primary resource of a computer
- CPU scheduling is the basis of multiprogrammed operating systems
- By switching the CPU among processes, the operating systems can be made more productive
- **Scheduling** refers to the way processes are assigned to run on the available CPUs. since there are typically many more processes running than there are available CPUs.
- This assignment is carried out by software known as a **scheduler** and **dispatcher**.

CPU - I/O burst

Each process is assigned a fixed number of time slices to perform its execution at the processor or to complete its I/O operation. There are two types of “Bursts” on the timeline:

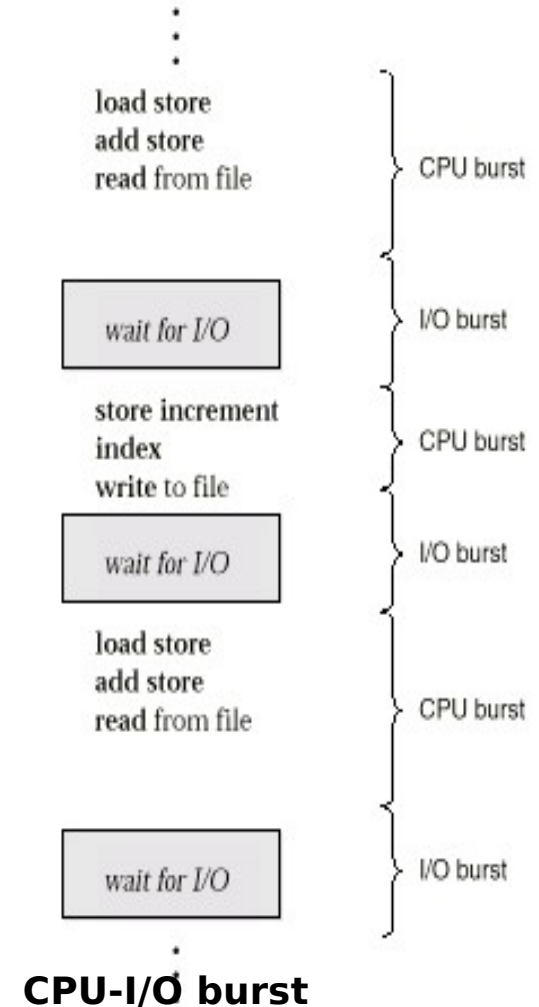
CPU burst

- CPU burst is the time allocated to a process or required by a process to execute on CPU.

I/O Burst

I/O burst is the time allocated or required by a process to perform its I/O operation.

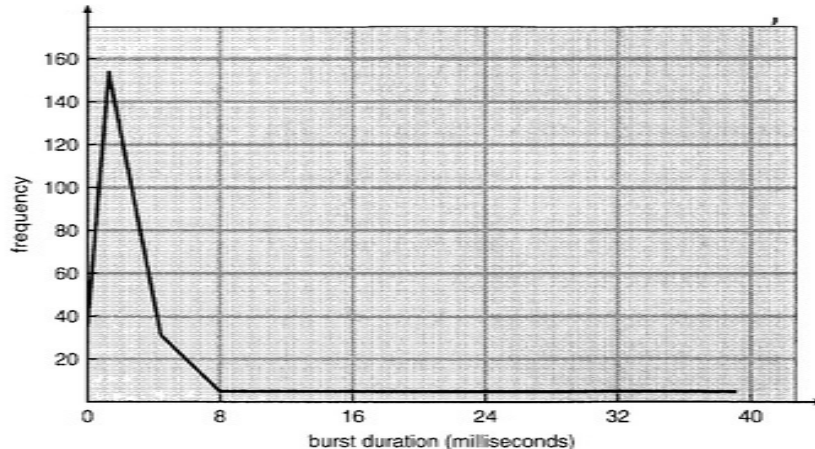
CPU - I/O burst cycle: If we carefully see the timeline of execution of all processes in the system, most processes will be alternating between CPU and I/O operations. i.e. on timeline we can observe CPU burst followed by I/O burst. In this alternating burst sequence, CPU intensive process have larger CPU burst while I/O intensive processes have larger I/O



operation requirement.

CPU Scheduling

- Whenever the CPU finishes executing a process, the operating system must select another process in the ready queue (**A process must be in ready/waiting queue and not in any other state as per state transition diagram to be scheduled next on the CPU**)
- This selection of next process from ready queue is done by the **scheduler**
- The selection process is carried out by the short-term scheduler (CPU scheduler)
 - The scheduler selects a process from the list of processes in memory ready for execution and allocates the CPU
 - Although CPU bursts differ from computer to computer & from process to process, they tend to have a frequency curve shown in the diagram below with a large number of short CPU bursts and a small



CPU burst durations

Types of CPU Schedulers

There are three types of **process schedulers** based on the **source and destination location** of the process being scheduled

- **Short term Scheduler(CPU scheduler)**
- **Medium term Scheduler**
- **Long term Scheduler**

Short term scheduler(CPU scheduler)

- A short term scheduler, also called CPU scheduler is responsible for selecting the jobs from ready queue and dispatch the selected job for execution at CPU.
- This scheduler is invoked frequently and should be implemented in a very efficient manner with minimum scheduling overhead.
- How much time will be allowed to a process on CPU is dependent on the CPU scheduling algorithm used.

Types of CPU schedulers

There are two types of CPU schedulers:

a)Preemptive Scheduler:

Preemptive scheduling is used when a process switches from **running state to ready state** or from **waiting state to ready state**. The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in ready queue till it gets next chance to execute.

b)Non-preemptive Scheduler: Non-preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state. In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state. In case of non-preemptive scheduling does not interrupt a process running CPU in middle of the execution. Instead, it waits till the process complete its CPU burst time and then it can allocate the CPU to another process. .

When preemptive and non-preemptive are used

- 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/exits
- When scheduling occurs in either 1st or 4th way, then the scheduling scheme is called **non-preemptive** or **cooperative**, all other scheduling scheme is termed as **preemptive**(eg. scheduling 2 and 3)
- In **non-preemptive scheduling**, once the CPU is allocated to a process, the process keeps using the CPU until it either finishes its execution or it enters in to a waiting state
 - It is used on certain/ most hardware since it does not require special hardware needed by preemptive scheduling
- In **preemptive scheduling**, An interrupt causes currently running process to give up the CPU and be replaced by another process(the situations 2nd and 3rd)
 - The design of the operating system kernel is affected
 - it **incur** cost associated with access to shared data

Dispatcher

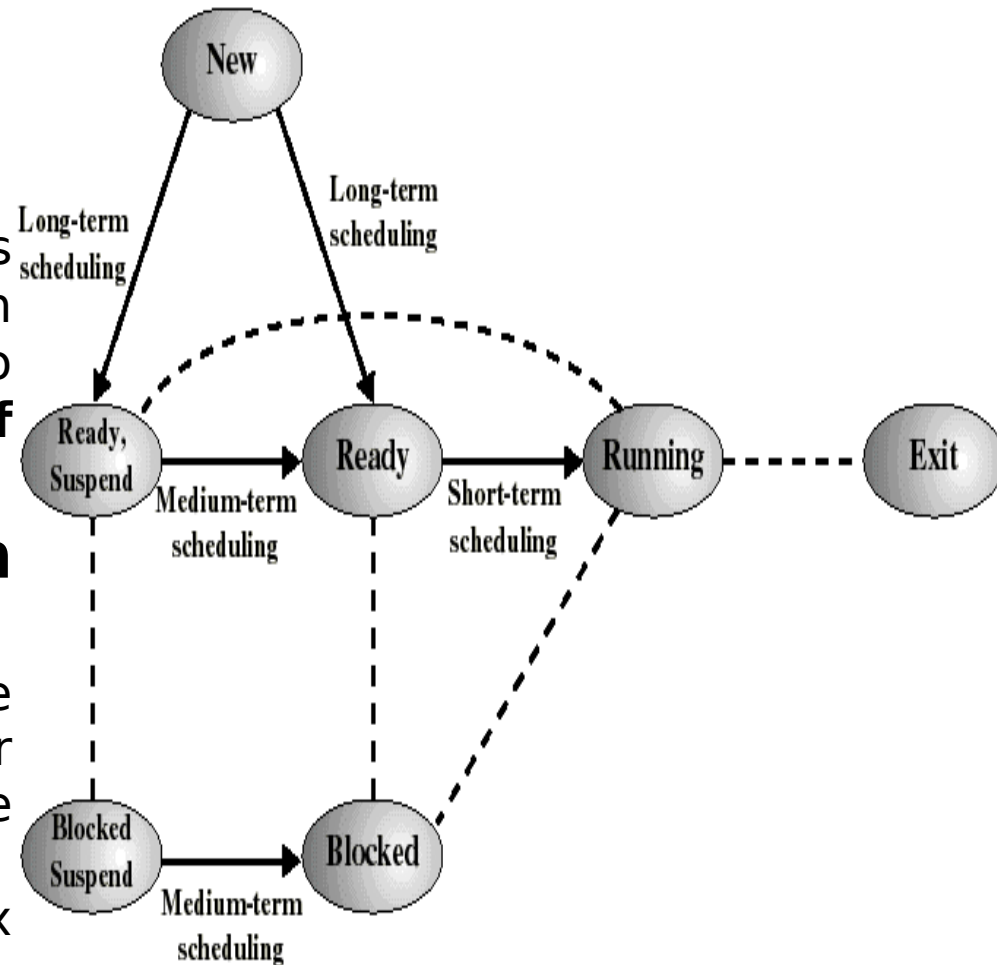
- The **dispatcher** is the module that gives control of the CPU to the process selected by the **short-term scheduler**.
- This function involves the following:
 - switching context
 - switching to user mode
 - jumping to the proper location in the user program to restart that program
- The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.

✓ **Medium-term: which process to swap in or out?**

- Controls the process remains resident in memory and which jobs must be swapped out to **reduce degree of multiprogramming**

✓ **Long-term: which process to admit?**

Determines which programs are admitted to the system for processing & it Controls the degree of multiprogramming
Attempts to keep a balanced mix of processor-bound and I/O-bound processes



CPU Scheduling Criteria

The most common criteria used to compare scheduling algorithms:

✓ **CPU Utilization**

- The fraction of time a device is in use. (ratio of in-use time / total observation time)

✓ **Throughput**

- The number of job completions in a period of time. (jobs / second)

✓ **Turnaround time**

- The interval between the submission of a process to its execution
- It is the sum of the periods spent waiting to get the memory, waiting in the ready queue, executing on the CPU and doing I/O

✓ **Waiting time**

- The sum of the periods spent waiting in the ready queue

✓ **Service time**

- The time required by a device to handle a request.

✓ **Response time**

- Amount of time it takes from the submission of a request till the first response is produced

CPU Scheduling Optimization

criteria

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

Note that:

- ☑ It's desirable to maximize CPU utilization and throughput and minimize turnaround time, waiting time and response time.

CPU Scheduling Algorithms

- Scheduling deals with the problem of deciding which of the outstanding requests is to be allocated resources.
- Scheduling algorithms are used for distributing resources among parties which simultaneously and asynchronously request them. In **OS** (to share CPU time among both threads and processes)
- The main purposes of scheduling algorithms are to minimize **resource starvation** and to ensure fairness amongst the parties utilizing the resources.
- **There are many different scheduling algorithms:**
 1. First-Come, First Served(FCFS)
 2. Shortest Job First(SJF)
 3. Priority Based Scheduling
 4. Round Robin Scheduling
 5. Multi-Level Queues
 6. Multi-Level Feedback Queues

1.First-Come, First Served(FCFS)

- In this algorithm, that process the requests the CPU first is allocated the CPU first
- The implementation of this algorithm is handled by FIFO queue
 - Arriving jobs are inserted in to the tail(rear) of the ready queue and the process to be executed next is removed from the front (head) of the ready queue
- Relative importance of jobs is measured by arrival time
- The average waiting time is quite too long
- **Throughput** can be low, since long processes can hog the CPU
- **Turnaround time, waiting time** and **response time** can be high
- A long CPU-bound process may hog the CPU and may force shorter processes to wait for a prolonged period.
- This may lead to a long queue of ready jobs in the ready queue

(convoy effect)

1.First-Come, First Served(FCFS) (cont.)

- The convoy effect results in a lower CPU and device utilization
- It's a non-preemptive algorithm
 - A process runs until it blocks for an I/O or it terminates
- Favors CPU-bound processes
 - A CPU-bound process monopolizes the processor
 - I/O-bound processes have to wait until completion of CPU-bound process
 - I/O-bound processes may have to wait even after their I/Os are completed (poor device utilization)
 - Better I/O device utilization could be achieved if I/O bound processes had higher priority

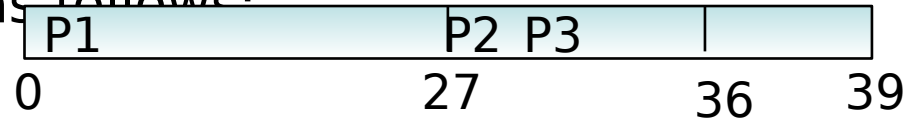
1.First-Come, First Served(FCFS) (cont.)

Example 1:

Consider the following processes that arrive at time zero, with the length of the CPU burst given in milliseconds

Proces s	Burst Time
P1	27
P2	9
P3	3

- If the processes arrive in the order of P1, P2 and P3 and are served in the FCFS order, then the waiting time for each of the processes will be as follows:



- Waiting time for P1 is 0 ms, meaning it starts immediately
- Waiting time for P2 is 27 ms, before starting
- Waiting time for P3 is 36 ms
- Average waiting time = **$(0+27+36)/3=21$ ms**

1.First-Come, First Served (FCFS) (cont.)

* What if the order of the processes was P2, P3, P1? What will be the average waiting time? Check [avg. waiting time= **7 ms**] what do you notice from this?

Example 2:

Proces s	Arriva l time	Service time
1	0	8
2	1	4
3	2	9
4	3	5

Average wait = $((0) + (8-1) + (12-2) + (21-3)) / 4 = 35/4 = 8.75$

Waiting time for $P_1 = 0$; $P_2 = 8-1$; $P_3 = 12-2$; $P_4 = 21-3$

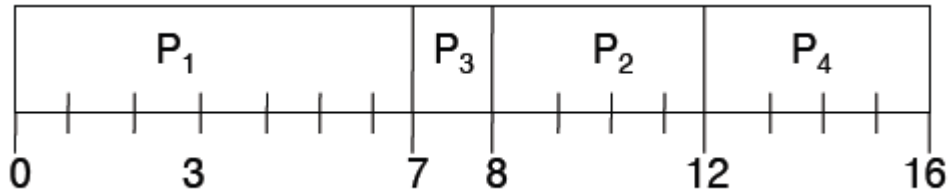


2. Shortest Job First (SJF), Shortest Job Next (SJN)

- SJF policy selects the job with the shortest (expected) processing time first.
- With this strategy the scheduler arranges processes with the least estimated processing time remaining to be next in the queue. This requires advance knowledge or estimations about the time required for a process to complete
- **Two schemes:**
 - **Non-preemptive** – once CPU is given to a process, it cannot be preempted in the current CPU burst
 - **Preemptive** – if a new process arrives with CPU burst length less than the remaining time of current process, preempt.
- One major difficulty with SJF is the need to know or estimate the processing time of each job (can only predict the future!)
- This scheme is known as the **Shortest-Remaining-Time-First (SRTF)**
- **SJF is optimal** – gives minimum average waiting time for a given set of processes
- **Starvation** is possible, especially in a busy system with many small processes being run.

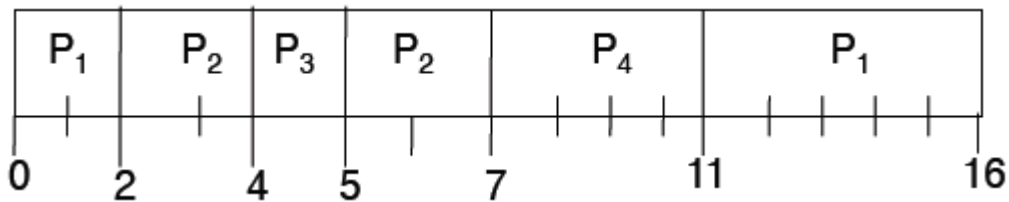
2. Shortest Job First (SJF), Shortest Job Next (SJN)

a. NonPreemptive



$$\text{Average waiting time} = (0 + (8 - 2) + (7 - 4) + (12 - 5)) / 4 =$$

4 b. preemptive



$$\text{Average waiting time} = ((11 - 2) + (5 - 4) + (4 - 4) + (7 - 5)) / 4 = 3$$

Example: SJF

Processes	Arrival time	Service time
1	0	7
2	2	4
3	4	1
4	5	4

3. Priority Based Scheduling

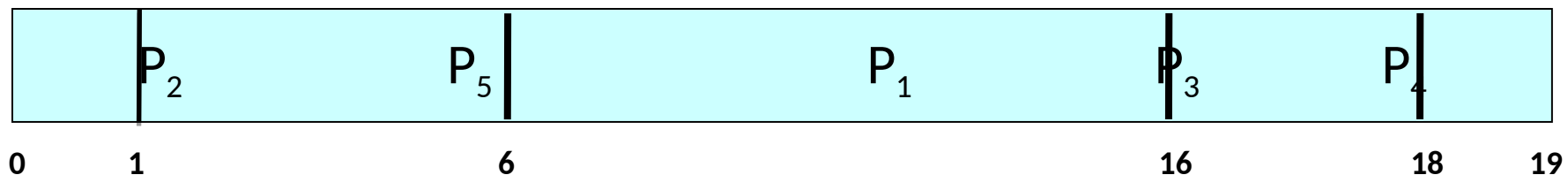
- Assign each process a priority. Schedule **highest priority first**. All processes within same priority are FCFS
- Priority may be determined by user or by some default mechanism
- The system may determine the priority based on memory requirements, time limits, or other resource usage
- CPU allocated to process with highest priority
- **Preemptive or non-preemptive**
- **Problem** $\equiv$ **Starvation** – low priority processes may never execute.
- **Solution** $\equiv$ **Aging** – as time progresses increase the priority of the process
- Delicate balance between giving favorable response for interactive jobs, but not starving batch jobs

3. Priority Based Scheduling(cont...)

- **EXAMPLE** :Consider the following processes that arrive at time zero, with the length of the CPU burst and priorities given.

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
1	10	3
2	1	1
3	2	4
4	1	5
5	5	2

Using Priority scheduling, we would schedule these processes according to the following Gant chart:



$$\text{Average waiting time} = (6+0+16+18+1)/5 = 8.2$$

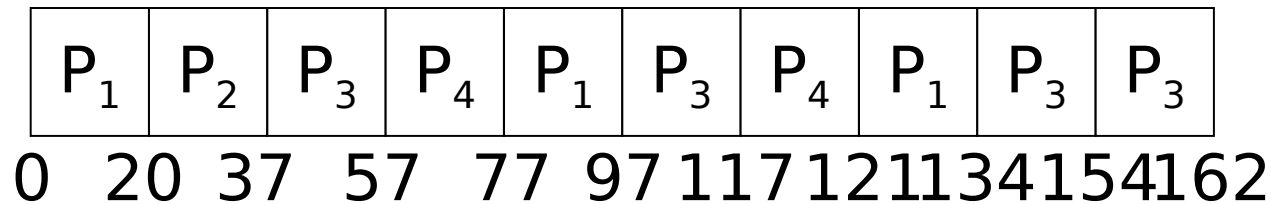
4. Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue
- The name of the algorithm comes from the round- robin principle known from other fields, where each person takes an equal share of something in turn
- If there are n processes in the ready queue and the time quantum is q , then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units
- **Performance: Choosing a time quantum q :**
 - **q large** $\Rightarrow$ FIFO
 - **q small** $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

4. Round Robin (RR) (cont.)

Example 1: RR with time quantum = 20

Processes	Burst time
P1	53
P2	17
P3	68
P4	24

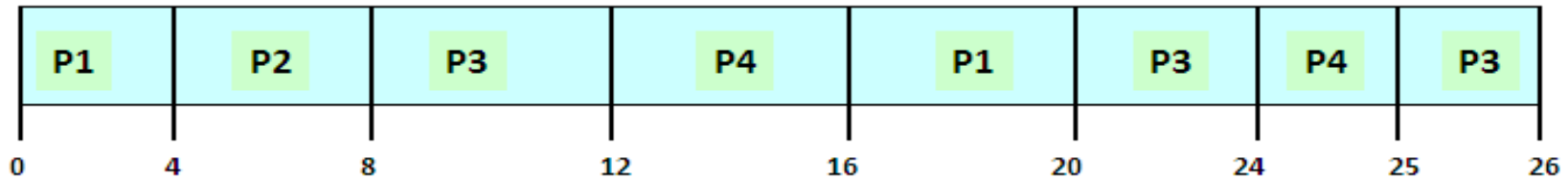


Typically, higher average **turnaround** than SJF, but better *response*

4. Round Robin (RR) (cont.)

Example 2: RR with time quantum = 4, no priority-based preemption

Process	Arrival Time	Service Time
1	0	8
2	1	4
3	2	9
4	3	5



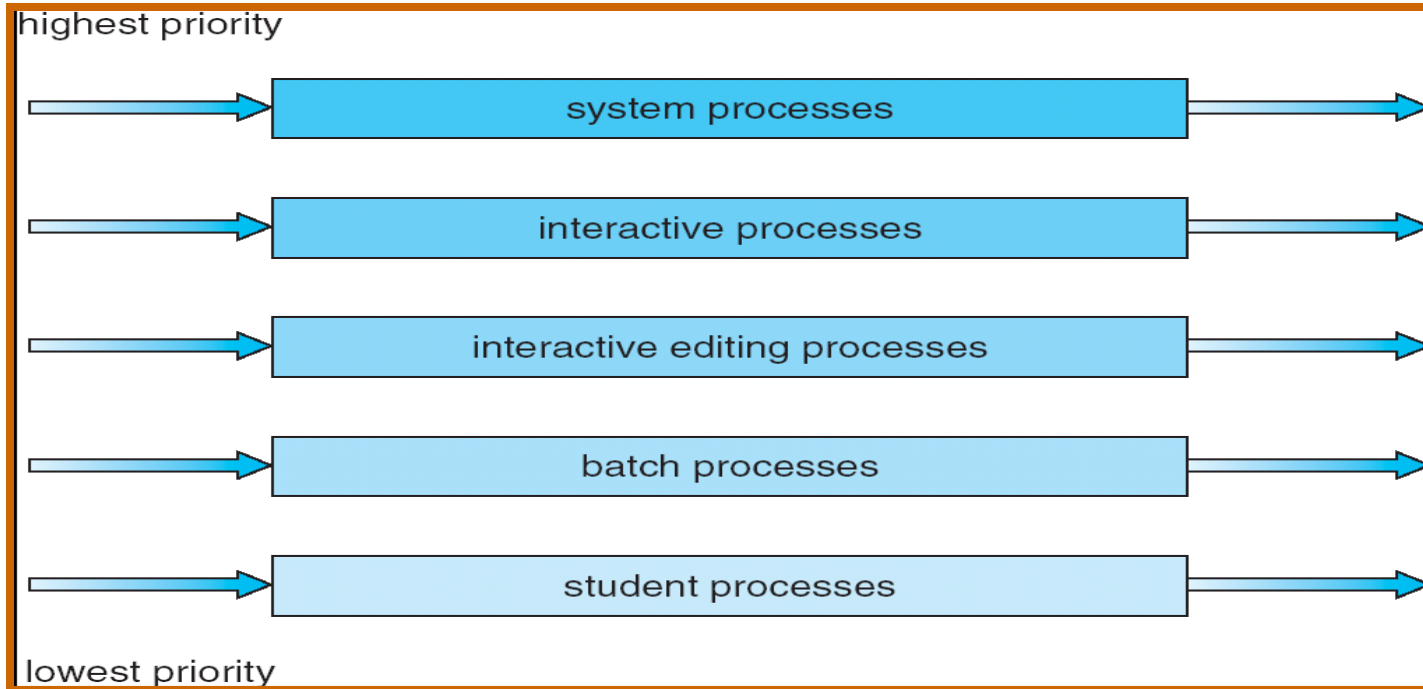
$$\text{Average wait} = ((20-0) + (8-1) + (26-2) + (25-3)) / 4 = 74/4 = 18.5$$

5.Multilevel Queue Scheduling

- Ready queue is partitioned into separate queues:
 - foreground (interactive)
 - background (batch)
- Each queue has its own scheduling algorithm
 - foreground – RR
 - background – FCFS
- Scheduling must be done between the queues
 - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
 - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
 - 20% to background in FCFS

5.Multilevel Queue Scheduling

- For example, could separate system processes, interactive, batch, favored, unfavored processes.



6. Multilevel Feedback Queue Scheduling

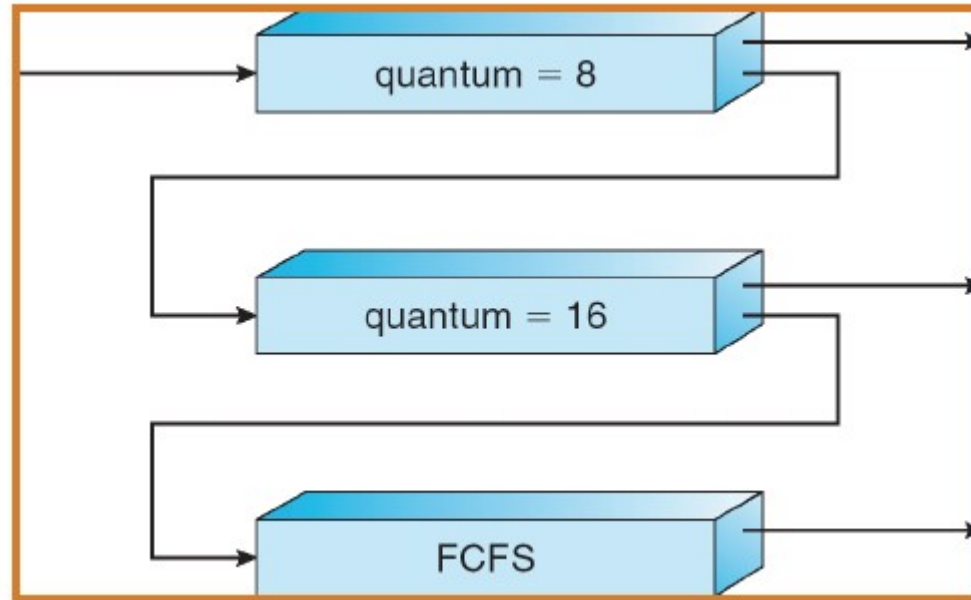
- A process can move between the various queues
 - aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - number of queues
 - scheduling algorithms for each queue
 - method to determine when to upgrade a process
 - method to determine when to demote a process
 - method used to determine which queue a process will enter when that process needs service

6. Multilevel Feedback Queue

Example:

- Three queues:
 - Q0 – RR with time quantum 8 milliseconds
 - Q1 – RR time quantum 16 milliseconds
 - Q2 – FCFS
- Scheduling
 - A new job enters queue Q0: when it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q1
 - At Q1 job: it receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q2

6. Multilevel Feedback Queue (cont.)



CPU Scheduling: using priorities

Here's how the priorities are used in Windows systems

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Scheduling Algorithms

- **Real-time systems**

- Hard real-time systems - required to complete a critical task within a guaranteed amount of time
- Soft real-time computing - requires that critical processes receive priority over less fortunate ones

- **Multiple Processor Scheduling**

- ¬_λ Different rules for homogeneous or heterogeneous processors
- ¬_λ Load sharing in the distribution of work, such that all processors have an equal amount to do.
- ¬_λ Each processor can schedule from a common ready queue (equal machines) OR can use a master slave arrangement

- **Thread Scheduling**

- Local Scheduling - How the user threads library decides which thread to put onto an available LWP (Light Weight Process)--- **process contention scope**
- Global Scheduling - How the kernel decides which kernel thread to run next

Linux Scheduling

- **Two algorithms:**
 - **time-sharing and real-time**
- **Time-sharing**
 - Prioritized credit-based – process with most credits is scheduled next
 - Credit subtracted when timer interrupt occurs
 - When credit = 0, another process chosen
 - When all processes have credit = 0, recrediting occurs
 - Based on factors including priority
- **Real-time**
 - Soft real-time
 - Posix.1b compliant – two classes
 - FCFS and RR
 - Highest priority process runs first

numeric priority	relative priority		time quantum
0	highest	real-time tasks	200 ms
•			
•			
•			
99		other tasks	10 ms
100			
•			
•			
•			
140	lowest		

Algorithm Evaluation Summary

- **Which algorithm is the best?**
- The answer depends on many factors:
 - the system workload (extremely variable)
 - hardware support for the dispatcher
 - relative importance of performance criteria (response time, CPU utilization, throughput...)
 - The evaluation method used (each has its limitations...)
- Which one works the best is application dependent
 - General purpose OS will use priority based, round robin, preemptive
 - Real Time OS will use priority, no preemption

Terminology for Examples

- AT: Arrival time of a process
- BT: Burst time of a process
- CT: completion time of a process
- WT: waiting time of a process
- TAT: turn around time
- ST: Scheduled time of a process

Formulas

- **Turn around time** is the total time in which a process is present in the system, irrespective to the fact that process was waiting, doing I/O or executing.

$$\mathbf{TAT=CT-AT=BT+WT}$$

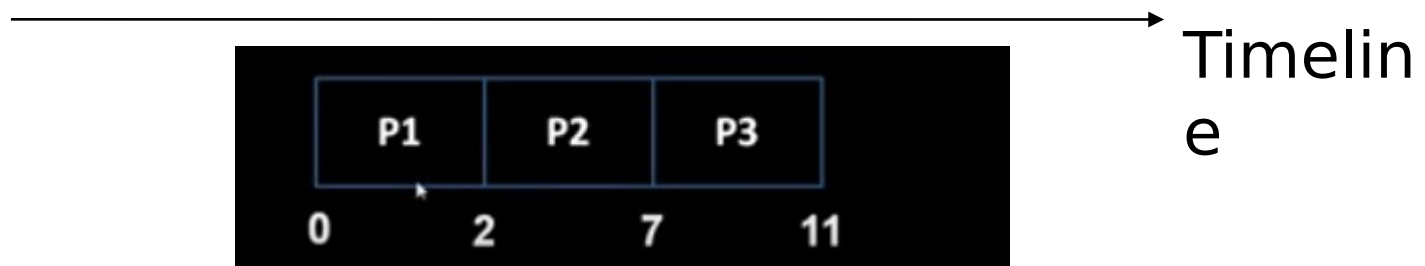
$$\text{Weighted TAT} = (CT-AT)/BT$$

- **Wait Time of a process**

$$\mathbf{WT=TAT-BT=CT-AT-BT}$$

Gantt Chart

- A graphical representation of process scheduling information
- A timeline represented by rectangular blocks, each block has a process Id,
- At time 0 P1 was scheduled for execution, at time 2 P2 was scheduled
- Schedule length is 11 unit of time



- **Schedule length** is the difference between maximum completion time of any process and the minimum arrival time of any process.

**(3 number of schedules are possible with 3 processes)

$$\begin{aligned}SL &= \text{Max}(\text{CT}) - \text{Min}(\text{AT}) \\ &= 11 - 0 \\ &= 11\end{aligned}$$

Throughput is the number of processes completed per unit time

$\text{Th} = \text{No. of process completed} / \text{Schedule length}$

FCFS example

- **Selection Criteria:** Basic criteria of selection of process is AT(Arrival Time)
- **Mode:** It is Non-preemptive mode
- **Assumptions:** context switch time is negligible, each process have only CPU burst time and zero I/O burst time.
- **Example:** Given following processes, compute TAT, WT for each process using FCFS

SN	Process	AT	BT
1	P1	0	2
2	P2	1	3
3	P3	2	5
4	P4	3	4
5	P5	4	1

solution

- First compute sum of all Burst times

$$\begin{aligned}\Sigma BT &= 2+3+5+4+1 \\ &= 15\end{aligned}$$

So, on Gantt chart we need to make a timeline of length 15, because in this amount of time all the processes will finish their burst time.

So let us make Gantt chart for schedule using FCFS:

Step 1: Select the process whose AT is smallest:
here P1 is arrived on time=0, so we select the P1 and schedule it on the processor.

Gantt chart for P1



- For process P1, Scheduled time is 0, because it was given processor at time 0, its Burst time is 2 units, so after completing its BT, it was completed, so CT of P1 is 2. its WT is zero because it was immediately scheduled. Also $TAT = CT - AT = 2 - 0 = 2$

Gantt Chart for P2



- P2 was arrived at $AT=1$, but at that time CPU was held by P1. In, non-preemptive settings P2 will wait for P1. On time 2, it is Schedule time for P2. its BT is 3 so its completion time is $CT=BT+ST=3+2=5$
- $TAT(P2)=CT(P2)-AT(P2)$
 $=5-1=4$

$$WT(P2)=TAT(P2)-BT(P2)$$

$$=4-3=1$$

Complete Gantt Chart



- Important to see AT and Scheduled times of processes in the Gantt chart.
- Here schedule length is $SL=15-0=15$

Assignment question:FCFS

P.no	AT	BT	$\Sigma BT = 17$
1	4	2	
2	10	5	
3	6	4	
4	2	3	
5	5	2	
6	3	1	

- Compute ST, CT, TAT and WT for each process
- Compute SL
- Compute TH

Non preemptive Shortest Job First algorithm

Criteria

- The next process will be selected for execution whose Burst time is least.
- Non preemptive SJF allow a process to finish its BT once scheduled and do not allow forceful yield of processor
- In case of preemptive version of SJF, if another process of shorter BT arrives during schedule of a process, we prefer to force the previous process to leave the CPU and execute shorter BT process first.

Example: SJF-NP

Question:

P.no	AT	BT
1	0	3
2	1	4
3	2	2
4	4	1
5	6	2
6	8	1
		$\Sigma BT = 13$

Compute:

- WT, TAT, ST etc.
- TH
- SL

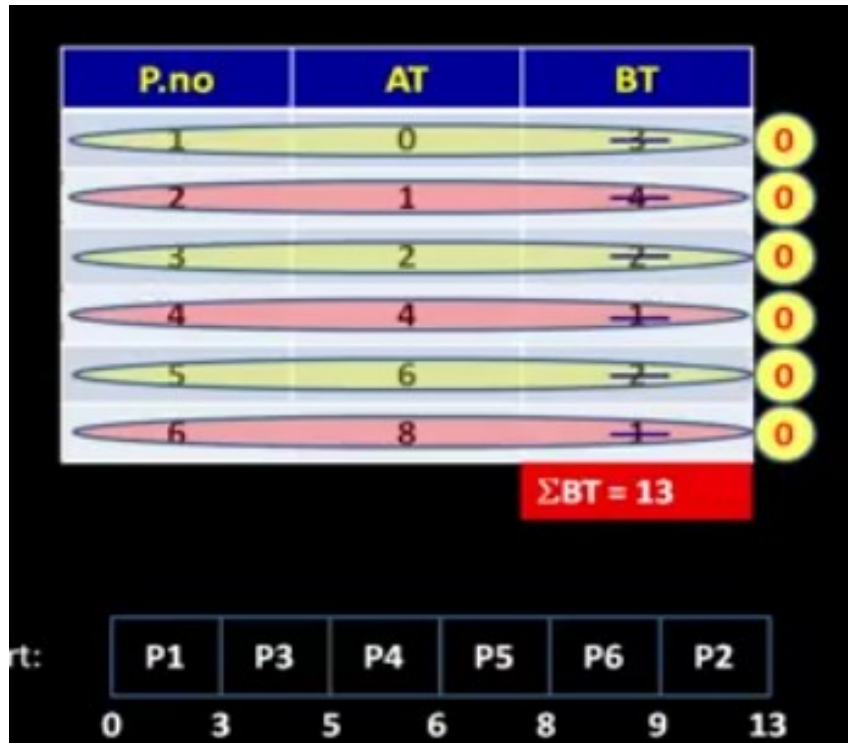
Schedule First process

P.no	AT	BT
1	0	4
2	1	4
3	2	2
4	4	1
5	6	2
6	8	1
$\Sigma BT = 13$		

t: P1
0 3

- Since at time 0, only P1 was in system so we have to schedule it.
 - However process P4 and P6 have shorter burst time but they have not arrived at time 0,
 - P1 will execute upto time = 3, in the mean time P2 and P3 will arrive.
 - However $AT(P2) < AT(P3)$
 - But $BT(P3) < BT(P2)$
- So after completion of P1, P3 will be selected.

Full solution



- Here the SL is 13
- Each process is selected and scheduled based on BT
- You can compute individual TAT, WT and ST, CT etc.

Assignment: SJF-NP

Question

- Solve following using NP-SJF and compute TAT, CT, WT, ST for each process
- Use preemptive SJF

P. no	AT	BT
1	3	2
2	8	1
3	2	6
4	5	5
5	4	3
6	2	4
		$\Sigma BT = 21$