Process Scheduling

Basic Concepts

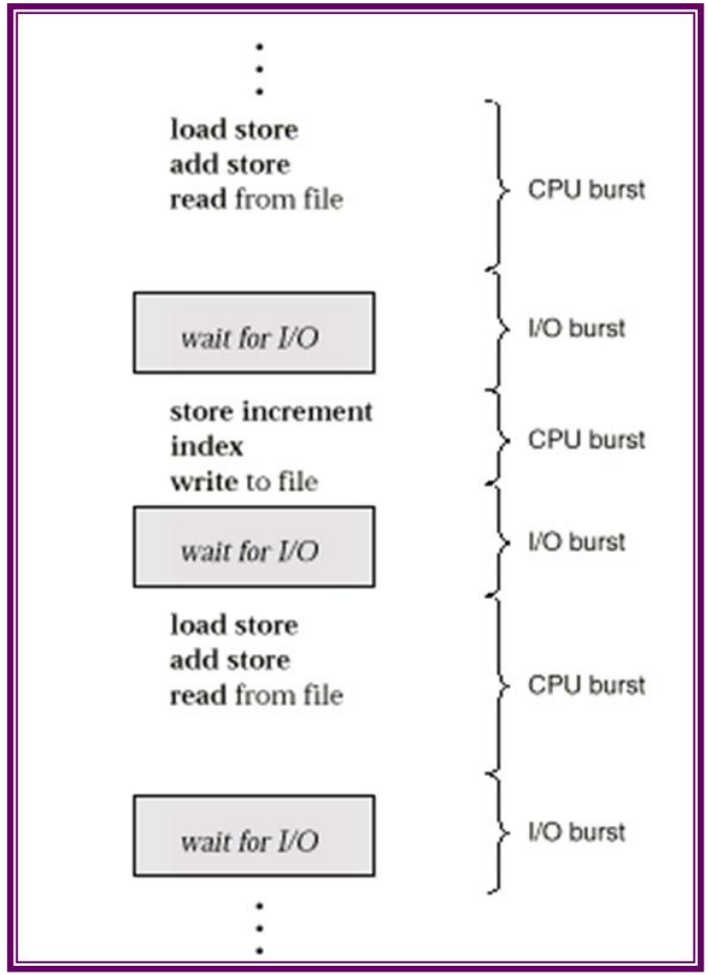
- CPU scheduling is the basis of multi-programmed operating systems.
- By switching the CPU among processes, the operating system can make the computer more productive.
- In a system with a single CPU core, only one process can run at a time.
- Others must wait until the CPU's core is free and can be rescheduled.
- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
  - Several processes are kept in memory at one time.
  - The number of processes currently in memory is known as the degree of multiprogramming.

Basic Concepts

- When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues.
- Every time one process has to wait, another process can take over use of the CPU.
- On a multicore system, this concept of keeping the CPU busy is extended to all processing cores on the system.
- Scheduling of this kind is a fundamental operating-system function.
- Almost all computer resources are scheduled before use.
- The CPU is, of course, one of the primary computer resources.
- Thus, its scheduling is central to operating-system design.

CPU–I/O Burst Cycle

- Balancing the objectives of multiprogramming and time sharing also requires taking the general behavior of a process into account.
- In general, most processes can be described as either I/O bound or CPU bound.
- An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations.
- A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.



CPU–I/O Burst Cycle

- Processes alternate between these two states.
- Process execution begins with a CPU burst.
- That is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on.
- Eventually, the final CPU burst ends with a system request to terminate execution
- An I/O-bound program typically has many short CPU bursts.
- A CPU-bound program might have a few long CPU bursts.
- This distribution can be important when implementing a CPU-scheduling algorithm.

Scheduling Queues

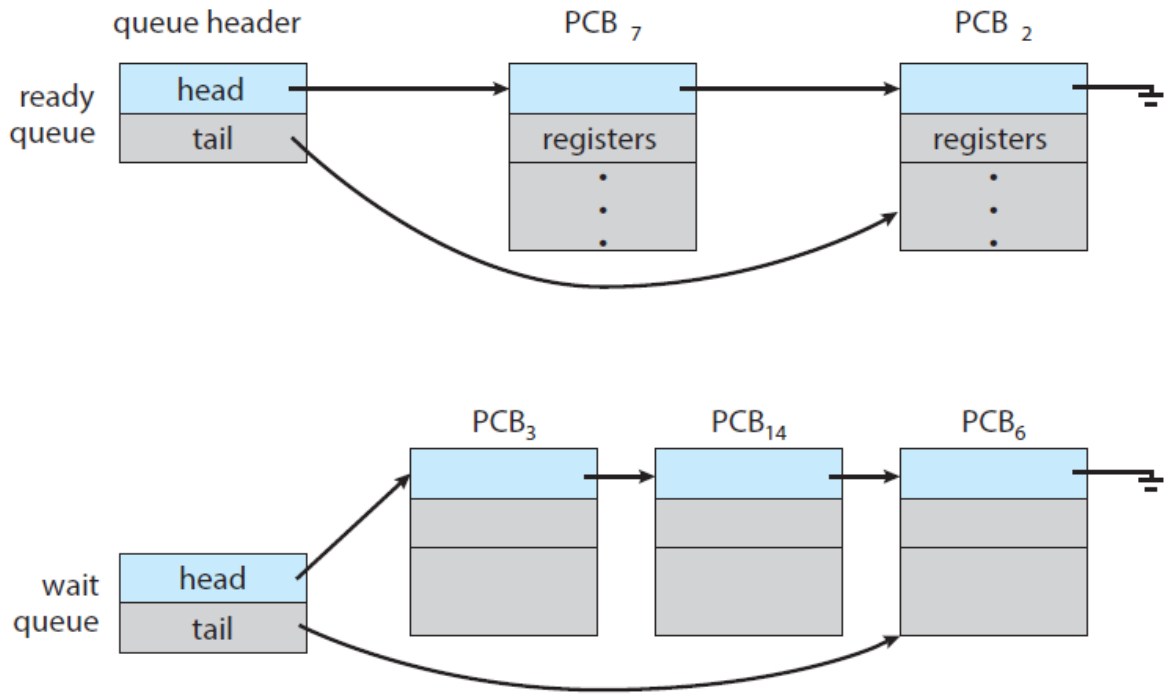
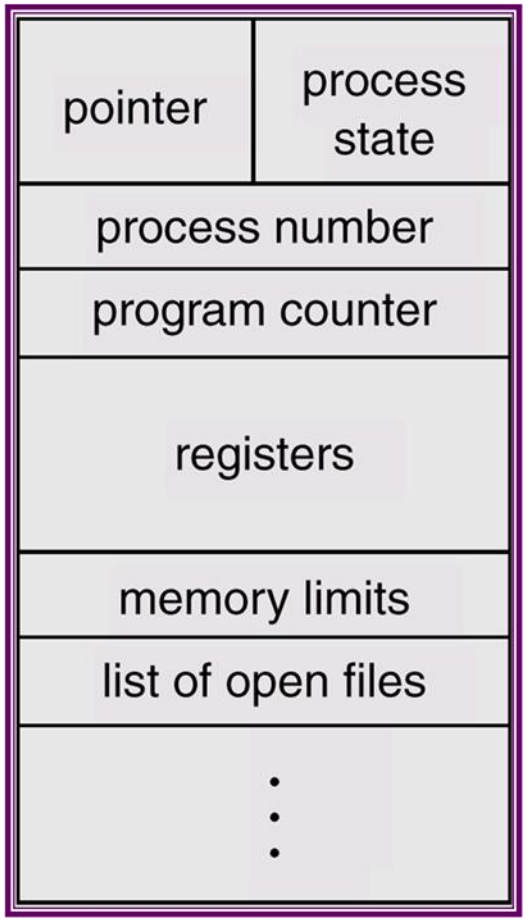


Figure 3.4 The ready queue and wait queues.



PCB

Scheduling Queues

- As processes enter the system, they are put into a ready queue, where they are ready and waiting to execute on a CPU's core.
- This queue is generally stored as a linked list; a ready-queue header contains pointers to the first PCB in the list, and each PCB includes a pointer field that points to the next PCB in the ready queue.
- The system also includes other queues. When a process is allocated a CPU core, it executes for a while and eventually terminates, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request.

Scheduling Queues

- Suppose the process makes an I/O request to a device such as a disk.
- Since devices run significantly slower than processors, the process will have to wait for the I/O to become available.
- Processes that are waiting for a certain event to occur — such as completion of I/O — are placed in a wait queue

Scheduling Queues

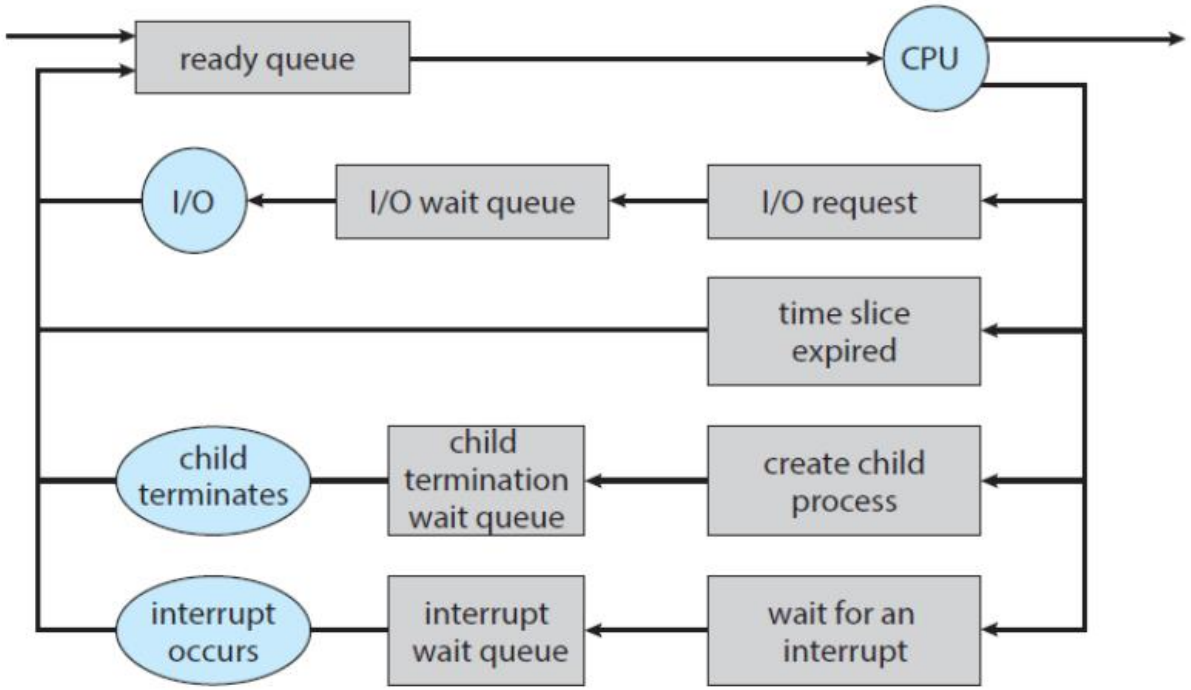


Figure 3.5 Queueing-diagram representation of process scheduling.

Scheduling Queues

- A common representation of process scheduling is a queueing diagram, such as that in Figure 3.5.
- Two types of queues are present:
  - the ready queue and
  - a set of wait queues.
- The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.
- A new process is initially put in the ready queue.
- It waits there until it is selected for execution, or dispatched.

Scheduling Queues

- Once the process is allocated a CPU core and is executing, one of several events could occur:
  - The process could issue an I/O request and then be placed in an I/O wait queue.
  - The process could create a new child process and then be placed in a wait queue while it awaits the child's termination.
  - The process could be removed forcibly from the core, as a result of an interrupt or having its time slice expire, and be put back in the ready queue.
- In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue.
- A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Types of Scheduling

Table 9.1 Types of Scheduling

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

Types of Scheduling – Short-term scheduling

- Also known as the **CPU Scheduler**, decides which process to run after **clock** or **I/O interrupt** or **system calls**.
- Whenever the CPU becomes idle, the OS must select one of the processes in the ready queue to be executed.
- This selection process is done by the **CPU/Short-term scheduler**.
- This scheduler selects a process from the processes in memory that are ready to execute and allocates CPU to that process.
- Ready queue need not follow FIFO order(priority or random order also allowed)
- The records in the queues are generally PCBs of the processes.

Types of Scheduling – Long-term scheduling

- Determines which programs are admitted to the system for processing
  - May be first-come-first-served
  - Or according to criteria such as priority, I/O requirements or expected execution time
- Controls the degree of multiprogramming
- More processes, smaller percentage of time each process is executed

Types of Scheduling – Medium-term scheduling

- The **medium-term scheduler** is executed somewhat more frequently.
- Medium-term scheduling is part of the swapping function.
- Typically, the swapping-in decision is based on the need to manage the degree of multiprogramming.
- On a system that does not use virtual memory, memory management is also an issue.
- Thus, the swapping-in decision will consider the memory requirements of the swapped-out processes.

Dispatcher

- Another component involved in the CPU-scheduling function is the dispatcher.
- The dispatcher is the module that gives control of the CPU's core to the process selected by the CPU scheduler.
- This function involves the following:
  - Switching context from one process to another
  - Switching to user mode
  - Jumping to the proper location in the user program to resume that program
- The dispatcher should be as fast as possible, since it is invoked during every context switch.
- The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency

Preemptive and Non-preemptive Scheduling

- CPU-scheduling decisions may take place under the following four circumstances:
  1. When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process) – non-preemptive
  2. When a process switches from the running state to the ready state (for example, when an interrupt occurs) - preemptive
  3. When a process switches from the waiting state to the ready state (for example, at completion of I/O) - preemptive
  4. When a process terminates – non-preemptive

Preemptive and Non-preemptive Scheduling

- For situations 1 and 4, there is no choice in terms of scheduling.
- A new process (if one exists in the ready queue) must be selected for execution.
- There is a choice, however, for situations 2 and 3.
- When scheduling takes place only under circumstances 1 and 4,we say that the scheduling scheme is non-preemptive or cooperative.
- Otherwise, it is preemptive.
- Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state.

Preemptive and Non-preemptive Scheduling

- Unfortunately, preemptive scheduling can result in race conditions when data are shared among several processes.
  - Consider the case of two processes that share data.
  - While one process is updating the data, it is preempted so that the second process can run.
  - The second process then tries to read the data, which are in an inconsistent state.

Preemptive and Non-preemptive Scheduling

- Preemption also affects the design of the operating-system kernel.
  - During the processing of a system call, the kernel may be busy with an activity on behalf of a process.
  - Such activities may involve changing important kernel data (for instance, I/O queues).
  - What happens if the process is preempted in the middle of these changes and the kernel (or the device driver) needs to read or modify the same structure?
  - Chaos ensues.
- Solution : Process Synchronization using locks

CPU Scheduling Algorithms

- Deals with the problem of deciding which of the process in ready queue is to be allocated the CPU
- Different scheduling algorithms
  - First Come, First Served
  - Shortest Job First
  - Priority
  - Round Robin

Scheduling Criteria

- Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another.
- In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.
- Many criteria have been suggested for comparing CPU-scheduling algorithms.
- Which characteristics are used for comparison can make a substantial difference in which algorithm is judged to be best.

Scheduling Criteria

- The criteria include the following:
  - CPU utilization. We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).
  - Throughput. If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput. For long processes, this rate may be one process over several seconds; for short transactions, it may be tens of processes per second.

Scheduling Criteria

- Turnaround time. From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. **Turnaround time is the sum of the periods spent waiting in the ready queue, executing on the CPU, and doing I/O**.
- Waiting time. The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O. It affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.

Scheduling Criteria

- Response time. In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. **This measure, called response time, is the time it takes to start responding, not the time it takes to output the response**.
- It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time, and response time.

- Consider the following set of processes with the indicated arrival time. The length of the CPU-burst time is given in ms.

Process	Arrival Time	CPU-Burst Time	Priority
P1	0	10	7
P2	1	1	2
P3	3	4	4
P4	5	2	3
P5	11	3	5

For each of the following scheduling algorithms, draw the Gantt chart and compute the avg. waiting time, avg. Response time, avg. turnaround time.

First Come First Served (FCFS)

- The process that requests CPU first is allocated the CPU first (done using FIFO queue).
- When a process enters the ready queue, its PCB is linked to the end of the queue.
- When CPU is free, it is allocated to the process at the head of the queue.
- Then the running process is removed from the queue.

Scheduling Criteria

- Turn Around Time = Completion time – Arrival Time
- Waiting Time = Turn Around Time – CPU Burst
- Response Time = First response – Arrival Time

FCFS

- Gantt Chart

- Calculation

Process	Waiting Time	Response Time	Turnaround Time
P1	10 -10=0	0	10-0=10
P2	10 -1=9	10-1=9	11-1=10
P3	12-4=8	11-3=8	15-3=12
P4	12-2=10	15-5=10	17-5=12
P5	9-3=6	17-11=6	20-11=9

Avg. W. T.= (0+9+8+10+6)/5= **6.6** ms.
Avg. R. T.= (0+9+8+10+6)/5= **6.6** ms.
Avg. T. T.= (10+10+12+12+9)/5= **10.6** ms.

First Come First Served (FCFS)

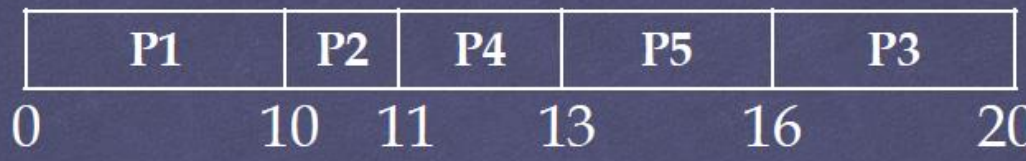
- If processes arrive in different order, like sorted based on CPU time, then average waiting time is reduced.
- FCFS is non-preemptive and so not used in Time Sharing Systems.
- Convoy effect
  - All the other processes wait for one big process to get off the CPU, results in lower CPU utilization

Shortest Job First (SJF)

- When the CPU is available, it is assigned to the process that has smallest next CPU burst
- If next CPU burst of 2 processes same, then FCFS is followed.
- Two kinds of SJF
  - Non-preemptive SJF
  - Preemptive SJF (Shortest Remaining Time First/Next)

Non-Preemptive SJF

- Gantt Chart



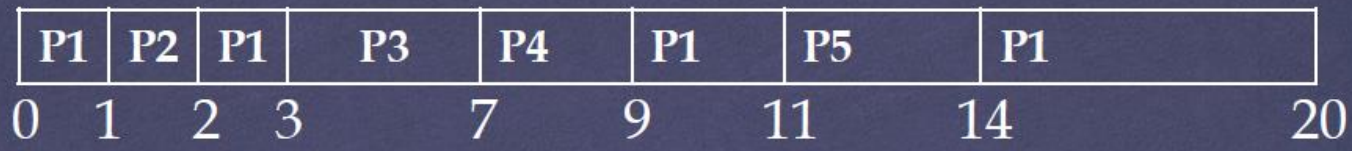
- Calculation

P	W.T.	R.T.	T.T.
P1	10 -10=0	0	10
P2	10 -1=9	10-1=9	11-1=10
P3	17-4=13	16-3=13	20-3=17
P4	8-2=6	11-5=6	13-5=8
P5	5-3=2	13-11=2	16-11=5

Avg. W. T.= (0+9+13+6+2)/5 = **6** ms.
Avg. R. T. = (0+9+13+6+2)/5 = **6** ms.
Avg. T. T. = (10+10+17+8+5)/5 = **10** ms.

Preemptive SJF

- Gantt Chart



- Calculation

P	W.T.	R.T.	T.T.
P1	20 -10=10	0	20-0=20
P2	1-1=0	1-1=0	2-1=1
P3	4-4=0	3-3=0	7-3=4
P4	4-2=2	7-5=2	9-5=4
P5	3-3=0	11-11=0	14-11=3

Avg. W. T.= (10+0+0+2+0)/5 = **2.4** ms.
Avg. R. T. = (0+0+0+2+0)/5 = **0.4** ms.
Avg. T. T. = (20+1+4+4+3)/5 = **6.4** ms.

SJF

- SJF is optimal, minimum average waiting time and decreases waiting time of short process
- But difficult to find the length of next CPU request time unless specified by user
- SJF used for long-term scheduling but not used for short-term scheduling, since cannot find next CPU burst time, only can predict.

Priority Scheduling

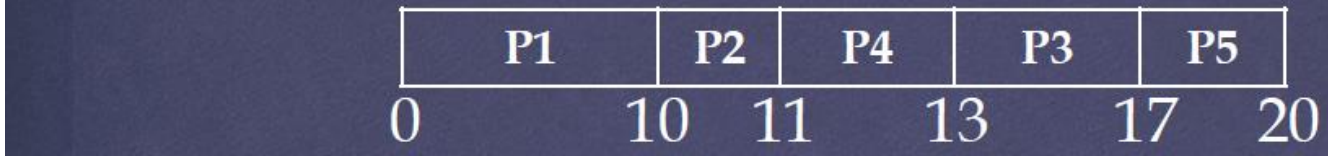
- Priority associated with each process and CPU allocated to process with the highest priority
- Process with equal priority can follow FCFS order
- The larger the CPU burst, the lower the priority, usually.
- Sometimes, priority values can be modified explicitly.
- Two kinds of priority scheduling
  - Non-preemptive priority
  - Preemptive priority

Non-preemptive priority

- A non-preemptive priority scheduling will simply put the new process at the head of the ready queue

Non-Preemptive Priority

- ***Note:*** Smaller value means a higher priority.
- Gantt Chart



- Calculation

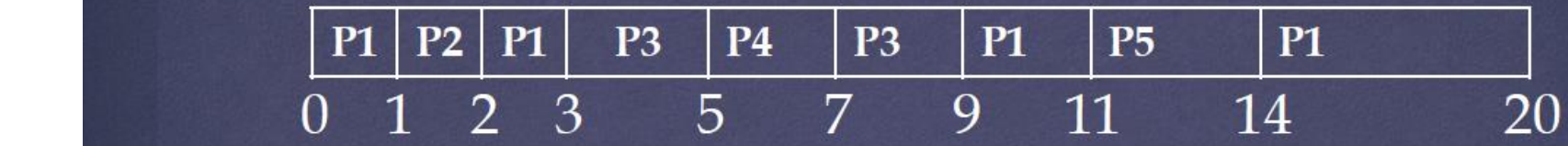
P	T.T.	W.T.	R.T.
P1	10-0=10	10-10=0	0
P2	11-1=10	10 -1=9	10-1=9
P3	17-3=14	14-4=10	13-3=10
P4	13-5=8	8-2=6	11-5=6
P5	20-11=9	9-3=6	17-11=6
Avg.	10.2	6.2	6.2

Preemptive priority

- Prremptive priority scheduling algorithm will pre-empt the CPU if the priority of the newly arrived process is higher than the priority of currently running processs

Preemptive Priority

- Gantt Chart



- Calculation

P	T.T.	W.T.	R.T.
P1	20-0=20	20-10=10	0
P2	2-1=1	1-1=0	1-1=0
P3	9-3=6	6-4=2	3-3=0
P4	7-5=2	2-2=0	5-5=0
P5	14-11=3	3-3=0	11-11=0
Avg.	**6.4**	**2.4**	**0**

Priority Scheduling Issue - Solution

- Issue – Starvation
  - Indefinite blocking/starvation of low priority processes
- Solution
  - Aging – involves gradually increasing the priority of processes that wait in the system for a long time

Round Robin Scheduling

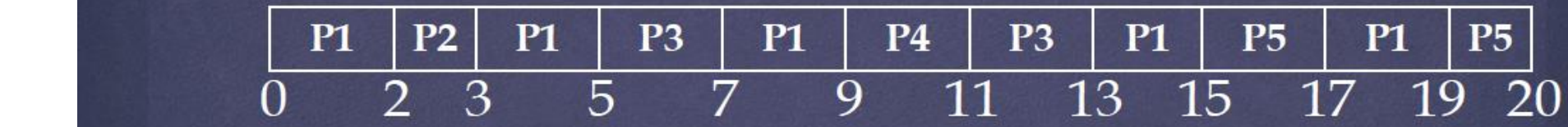
- Designed essentially for time sharing systems
- Similar to FCFS, but pre-emption is added to enable the system to switch between processes
- A small unit of time called a time quantum or time slice is defined (10-100ms in length)
- Ready queue treated as circular queue
- The CPU scheduler goes around the ready queue allocating the CPU to each process for a time interval of up to 1 time quantum
- Here ready queue maintained as FIFO queue – new processes added to the tail of the ready queue

Round Robin Scheduling

- The CPU Scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum and dispatches the process
- The process may have a CPU burst of less than 1 time quantum
- Then the process itself will release the CPU else interrupted and context switch occurs

RR (q=2 ms)

- **Note:** If two process need to enter the ready queue at the same time, use the FCFS to choose between them.
- Gantt Chart



- Calculation

P	T.T.	W.T.	R.T.
P1	19-0=19	19-10=9	0
P2	3-1=2	2-1=1	2-1=1
P3	13-3=10	10-4=6	5-3=2
P4	11-5=6	6-2=4	9-5=4
P5	20-11=9	9-3=6	15-11=4
Avg.	9.2	5.2	2.2

RR (q=2 ms)

- **Note:** If two process need to enter the ready queue at the same time, the new process will be served first.
- Gantt Chart

- Calculation

P	T.T.	W.T.	R.T.
P1	20-0=20	20-10=10	0
P2	3-1=2	2-1=1	2-1=1
P3	13-3=10	10-4=6	5 - 3=2
P4	9-5=4	4-2=2	7 - 5=2
P5	18-11=7	7-3=4	13 -11=2
Avg.	**8.6**	**4.6**	**1.4**

Round Robin

- If time quantum so large, then Round robin becomes same as FCFS
- If time quantum so small, then it leads to increase in context switching
- So time quantum should not be large or small, but must be medium

Multi-Level Queue Scheduling

- Processes are classified into different groups
  - System processes -> usually ordered by priority
  - Interactive processes or foreground processes
  - Interactive editing processes
  - Background/batch processes
  - Student processes
- They have different response time requirement, so may require different scheduling needs and foreground processes can have priority over background processes

Multi-Level Queue Scheduling

- A multi-level queue scheduling algorithm partitions ready queue into several separate queues
- Processes are permanently assigned to one queue based on some property of the process such as memory size, priority and process time.
- Each queue has its own scheduling algorithm
  - Eg:
    - Foreground process by Round Robin
    - Background process by FCFS
- Additionally scheduling among queues based on priority.
  - Eg:
    - Foreground queue -> high priority
    - Background queue -> low priority

Multi-Level Queue Scheduling

- Similarly below is the five scheduling queues in order of priority
  - System processes
  - Interactive processes
  - Interactive editing processes
  - Batch processes
  - Student processes
- If interactive editing processes enters ready queue,a batch process which is running can be pre-empted
- Time slice is also focused among the queues
- 80% for foreground process with RR
- 20% for background process with FCFS

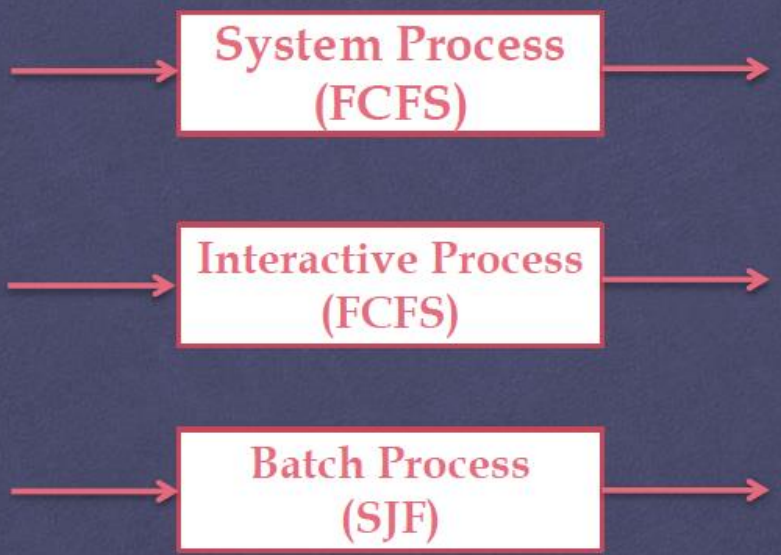
- Consider the following set of processes with the indicated arrival time. The length of the CPU-burst time is given in ms.

Process	Arrival Time	CPU-Burst Time	Priority
P1	0	10	7
P2	1	1	2
P3	3	4	4
P4	5	2	3
P5	11	3	5

For each of the following scheduling algorithms, draw the Gantt chart and compute the avg. waiting time, avg. Response time, avg. turnaround time.

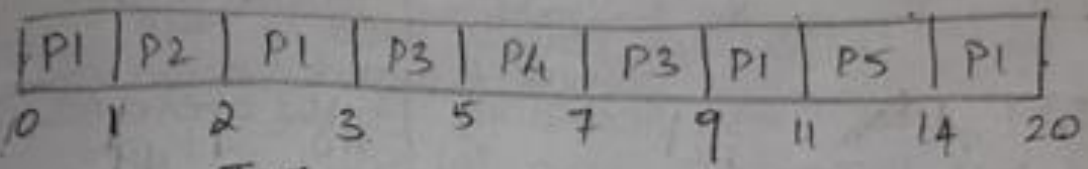
Multi-level Queue

- Assume both of P1 and P3 are batch processes, P2 is interactive processes, and both of P4 and P5 are system processes.
- Assume also that the batch processes queue is scheduled using the preemptive SJF and the other queues use FCFS.



FCFS

Gantt chart



	TAT	WT	RT
P1	20-0=20	20-10=10	0
P2	2-1=1	1-1=0	1-1=0
P3	9-3=6	6-4=2	3-3=0
P4	7-5=2	2-2=0	5-5=0
P5	14-11=3	3-3=0	11-11=0
	6.4	24	0

CPU Utilization

- CPU utilization = (CPU time / Total time) * 100 %
- In the above example, CPU time = 20 (to complete process p1 to p5)
- Assume context switch time = 0.5ms
- Total time = (20 + (0.5 * 10)) = 25 ms
- CPU Utilization = (20/25) * 100 = 80%

Multi-Level Feedback Queue Scheduling

- In multi-level queue scheduling, processes remain in same queue, not move from one queue to another.
- But in multi-level feedback queue scheduling, it allows processes to move between queues.
- The idea is to separate processes according to the characteristics of their CPU bursts.
- If processes use too much CPU time, they will move to lower priority queue
- This leave I/O bound and interactive process in high priority queues.
- If process waits too long in a lower priority queue, it may be moved to a higher priority queue to prevent starvation(Aging technique).

Multi-Level Feedback Queue Scheduling

- Multi-Level Feedback Queue Scheduling is defined by following parameters
  - Number of queues
  - Scheduling algorithms
  - Method used to determine when to upgrade a process to a higher priority queue
  - Method used to determine when to demote a process to a lower priority queue
  - Method used to determine which queue a process will enter when that process needs service.

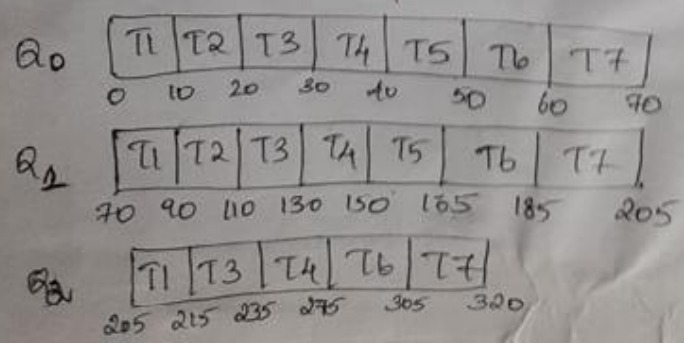
Multilevel Feedback Queue Scheduling.

1) Let us consider an example of MLFQ where ready queue is divided into 3 queues Q_0, Q_1 & Q_2.
Q_0 has high priority than Q_1 and then Q_2.
Q_0 and Q_1 follows Round Robin (RR) and Q_2 follows FCFS. If the following shows the processes, their order of arrival and burst time, draw the Gantt chart

	AT	BT
T1	0	40
T2	0	30
T3	0	50
T4	2	70
T5	4	25
T6	6	60
T7	7	45

TQ = 10 for Q_0 (RR)
TQ = 20 for Q_1 (RR)
Q_2 = FCFS

When a task enters the system, firstly it is added at the tail end of Q_0 and then system allots a fixed single Time Quantum. This scheduling algorithm provides the facility to move the tasks from one queue to another queue. If the task consumes more CPU time, the task is moved to the lower priority queue Q_1 & alloted double Time Quantum.



Predicting Next CPU Burst Length in SJF

- **Exponential Averaging or Aging**
- Let, Tn be the actual CPU burst time of nth process. T(n) be the predicted CPU burst time for nth process then the CPU burst time for the next process (n+1) will be calculated as,

$$T(n+1) = \alpha * Tn + (1-\alpha) * T(n)$$

- Where, α is the smoothing. Its value lies between 0 and 1.

Next CPU Burst Length

Q: If the length of the next CPU burst was predicted to be 8ms, the actual length used in this prediction was 5ms. According to the average exponential formula, what was the previous predicted value of CPU burst if α=0.4.

Solution

ι_{n+1}= 8ms.

t_n= 5ms.

ι_n= ?

α= 0.4

$\iota_{n+1} = \alpha * t_n + (1- \alpha) * \iota_n$

$8 = 0.4*5+0.6* \iota_n$

$\iota_n = (8-2)/0.6 = 10$ ms.

Algorithm Evaluation

- How do we select a CPU scheduling algorithm for a particular system?
- There are many scheduling algorithms, each with its own parameters.
- As a result, selecting an algorithm can be difficult.
- The first problem is defining the criteria to be used in selecting an algorithm.
- Criteria are often defined in terms of CPU utilization, response time, or throughput.
- To select an algorithm, we must first define the relative importance of these measures.
- Our criteria may include several measures, such as:
  - Maximizing CPU utilization under the constraint that the maximum response time is 1 second
  - Maximizing throughput such that turnaround time is (on average) linearly proportional to total execution time.

Algorithm Evaluation

- Once the selection criteria have been defined, we want to evaluate the algorithms under consideration.
- Various evaluation methods can be used like:
  - Deterministic modeling
  - Queuing Models
  - Simulations
  - Implementation

Deterministic Modeling

- This evaluation method takes a predetermined workload and evaluates each algorithm using that workload.
- The advantages of deterministic modeling is that it is exact and fast to compute.
- The disadvantage is that it is only applicable to the workload that we use to test.
- Of course, the workload might be typical and scale up but generally deterministic modeling is too specific and requires too much knowledge about the workload.

Queuing Models

- Another method of evaluating scheduling algorithms is to use queuing theory.
- Using data from real processes we can arrive at a probability distribution for the length of a burst time and the I/O times for a process.
- We can now generate these times with a certain distribution.
- We can also generate arrival times for processes (arrival time distribution).
- If we define a queue for the CPU and a queue for each I/O device we can test the various scheduling algorithms using queuing theory.
- Knowing the arrival rates and the service rates we can calculate various figures such as average queue length, average wait time, CPU utilization etc.

Queuing Models

- One useful formula is Little's Formula.
  - $n = \lambda w$
- Where
  - n is the average queue length
  - λ is the average arrival rate for new processes (e.g. five a second)
  - w is the average waiting time in the queue
- Knowing two of these values we can, obviously, calculate the third.
  - For example, if we know that eight processes arrive every second and there are normally sixteen processes in the queue we can compute that the average waiting time per process is two seconds.
- The main disadvantage of using queuing models is that it is not always easy to define realistic distribution times and we have to make assumptions.
- This results in the model only being an approximation of what actually happens.

Simulations

- Rather than using queuing models we simulate a computer.
- A Variable, representing a clock is incremented. At each increment the state of the simulation is updated.
- Statistics are gathered at each clock tick so that the system performance can be analyzed.
- The data to drive the simulation can be generated in the same way as the queuing model, although this leads to similar problems.
- Alternatively, we can use trace data. This is data collected from real processes on real machines and is fed into the simulation.
- This can often provide good results and good comparisons over a range of scheduling algorithms.
- However, simulations can take a long time to run, can take a long time to implement and the trace data may be difficult to collect and require large amounts of storage.

Implementation

- The best way to compare algorithms is to implement them on real machines.
- This will give the best results but does have a number of disadvantages.
  - It is expensive as the algorithm has to be written and then implemented on real hardware.
  - If typical workloads are to be monitored, the scheduling algorithm must be used in a live situation. Users may not be happy with an environment that is constantly changing.
  - If we find a scheduling algorithm that performs well there is no guarantee that this state will continue if the workload or environment changes.