Operating Systems
BCSC 0004

Process Management (1)

Books

Operating System Concepts
Ninth Edition
Avi Silberschatz Peter Baer Galvin Greg Gagne

Operating Systems: Internals and Design Principles
William Stallings

Outlines…

- Process Concept
- Process States
- Process State Transition Diagram
- Process Control Block
- Process Scheduling Concepts
- Threads and their management

Process & Program

- The term process was first used by the designers of the Multics in 1960’s
- The process has been given many definition for instance
  - A program in execution .
  - An asynchronous activity
  - The entity to which process are assigned
  - Sequence of states during the execution of a program
- The most frequently used def^{n} “Program in execution”
- What is the relationship between process & Program?
- It is the same beast with different name or when this beast is sleeping (not executing)called program and when it is executing becomes process.
- Process is not same as program. A program is only part of process.

Process Operations

- ➢Create/terminate process
- ➢Change program
- ➢Set/get process parameters
- ➢Block process
- ➢Awaken process
- ➢Switch process
- ➢Schedule process

- In process model all software on the computer is organised into a number of sequential processes.
- A process include PC, Register and variables.
- Conceptually each process has its own virtual CPU. In reality the CPU switches back and forth among process.

Process creation

- In general purpose system, some way is needed to create processes as needed during operation.
- There are four principle events led to process creation.
    - ➢System initialization
    - ➢Execution of a process creation system calls by a running process.
    - ➢A user request to create a new process
    - ➢Initialization of a batch job
- Fore ground process interact with users
- Background processes are processes that stay in background sleeping but suddenly springing to life to handle activity such as email ,webpage, printing and so on.

Process Termination

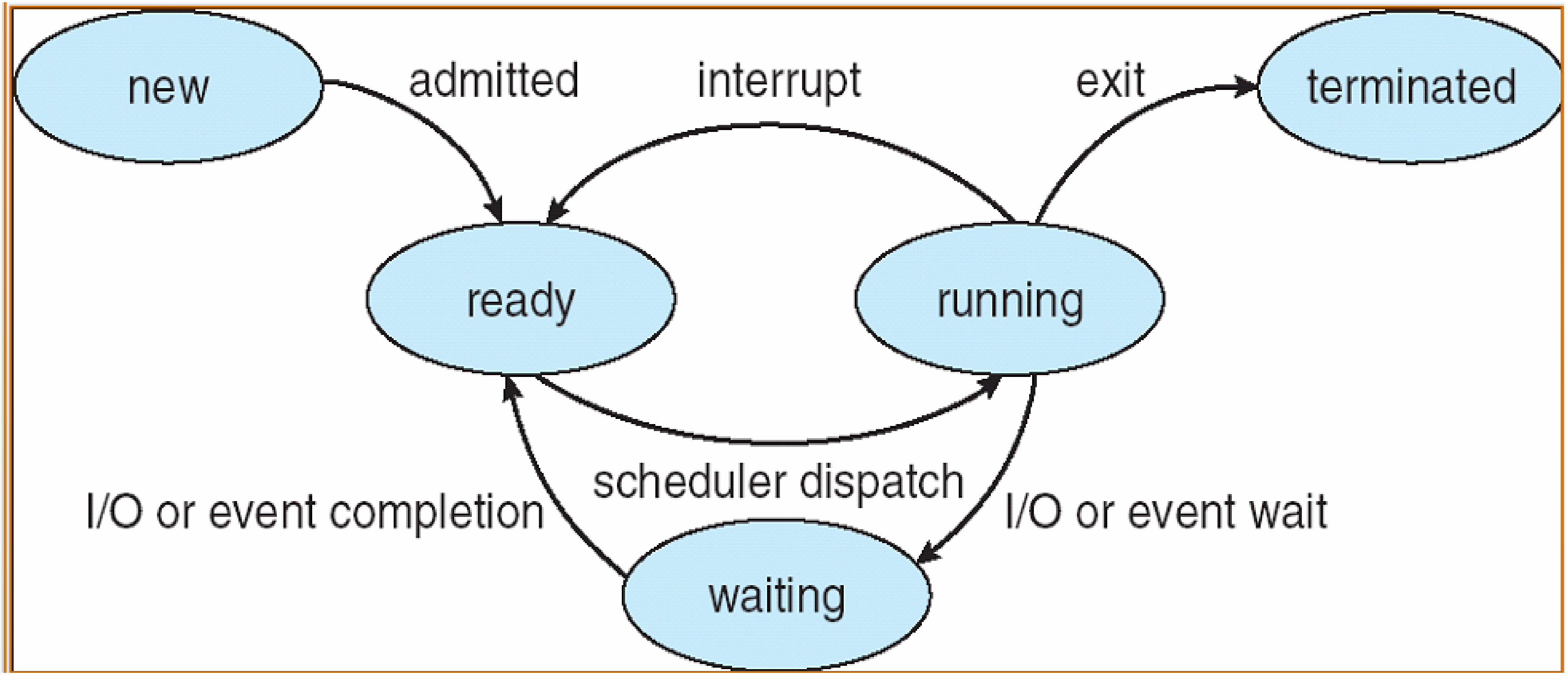
- ❑A process terminates when it finishes executing its last statements. Its resources are returned to the system and its process control block is erased and memory space is returned to a free memory pool.
- ❑The new process terminates existing process, usually due to following reasons.
- ➢Normal exit – process terminates bcz they have done their job.
- ➢Error exit – when process discover a fatal error.
- ➢Killed by another process – a process executes a system call telling the operating system to terminate some other process. In UNIX this call is kill.

Process State

- As a process executes, it changes state

- New: The process is being created
- Running: Instructions are being executed
- Waiting: The process is waiting for some event to occur
- Ready: The process is waiting to be assigned to a process
- Terminated: The process has finished execution

Process state transition Diagram



Process Control Block (PCB)

❑A process in an operating system is represented by a data structure known as a process control block.

❑The PCB contains important information about the specific process including

➢Process state

➢Program counter

➢CPU registers

➢CPU scheduling information

➢Memory-management information

➢Accounting information

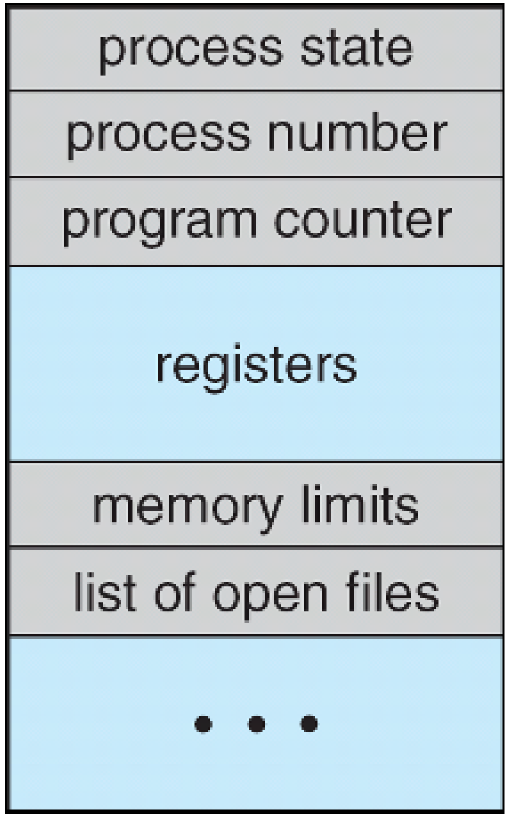
➢I/O status information

Process Control Block (PCB)

Information associated with each process

(also called task control block)

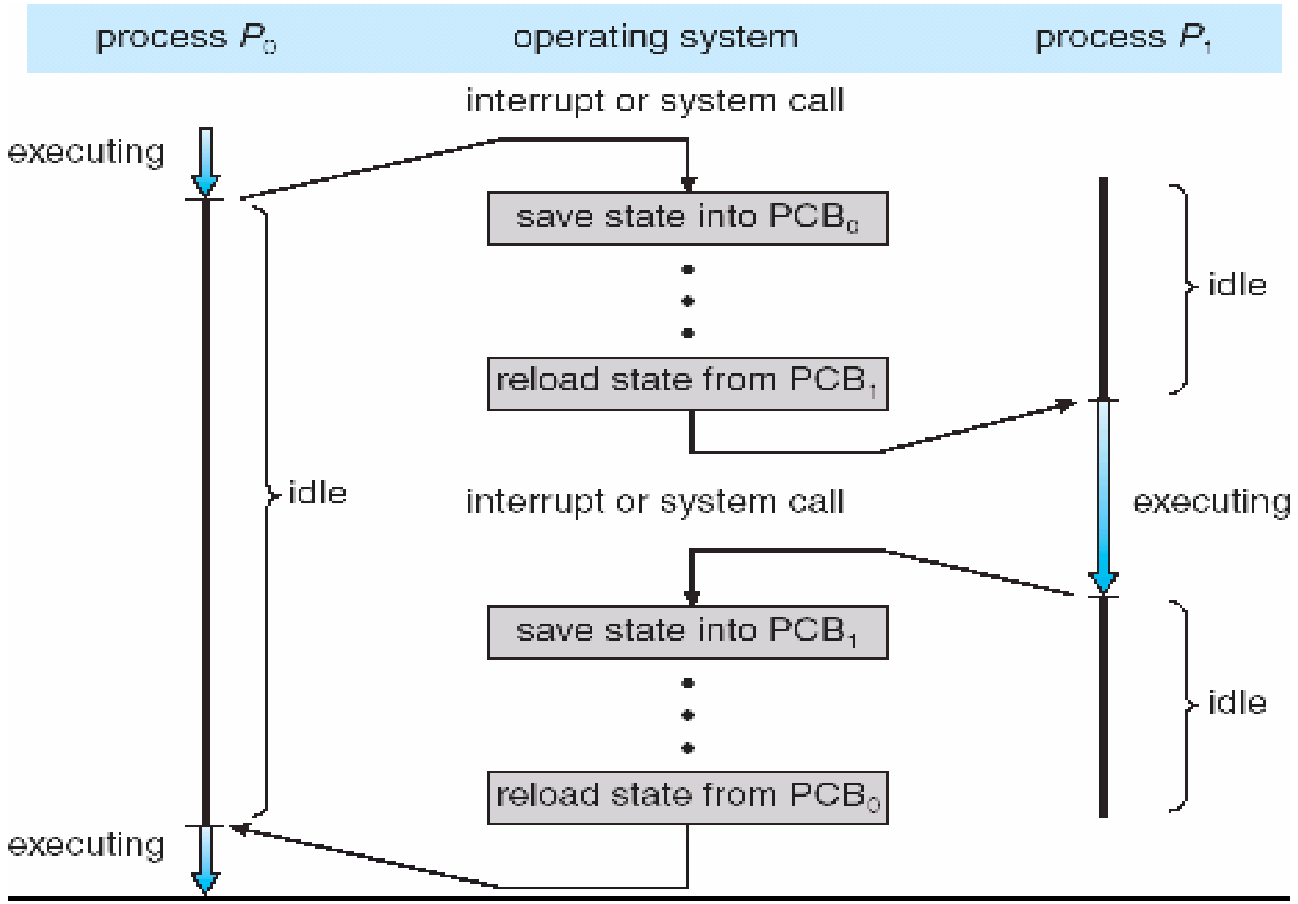
- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointer
- Memory-management information – memory allocated to the pr
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



Process Table

- ❑One entry for each process
- ❑Some typical data:
    - ➢Saved registers
    - ➢Process state
    - ➢Process ID
    - ➢Owner/group ID
    - ➢Priority
    - ➢Memory usage
    - ➢Open files

CPU Switch From Process to Process



Threads (1)

- ❑Despite of fact that a thread must execute in processes, the processes and its associated threads are different concept.
- ❑Process are used to group resources together and threads are the entities scheduled for execution on the CPU.
- ❑A thread is a single sequence stream within in a process. Bcz threads have some of the properties of processes, they are sometime called light weight processes.
- ❑In a process, threads allows multiple executions of streams. In many respect, threads are popular way to improve application through parallelism.
- ❑The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel.

Threads (2)

- ❑Process with one thread, a thread can be in any of several states(running, blocked, ready or terminated).
- ❑Each thread has its own stack.
- ❑An O.S. that has threads facility the basic unit of CPU utilization is thread.
- ❑A thread has a program counter , a register set and a stack space.
- ❑Thread are not independent of one another like processes: as a result threads share with other thread their code section, OS resources also known as task such as open files and signals.

Why threads

❑ Following are some reason why we use thread in designing operating system.

- ➢ A process with multiple threads make a great server for example printer server
- ➢ Bcz threads can share common data they do not use inter- process communication.
- ➢ Bcz of the vary nature, threads can take advantage of multiprocessors.
- ➢ Threads are very little resources of an O.S in which they are working. i.e. thread do not need new address space, global data, program code, or O.S resources.
- ➢ Context switching are fast when working with threads . The reason is that we only have to save and/or restore PC, SP & register.

Context switch

- To give each process on a multi programmed machine a fair share of the CPU, a hardware clock generates interrupts periodically.
- This allows the O.S to schedule all processes in main memory (using scheduling algorithm) to run on the CPU at equal intervals.
- Each time a clock interrupt occurs, the interrupts handler checks how much time the current running process has used.
- If it has used up its entire time slice the CPU scheduling algorithm
- In kernel picks a different process to run. Each switch of the CPU from one process to another it called a context switch.

Major step of context switching

- The value of the CPU registers are saved in the process table of the process that was running just before the clock interrupt occurred.
- The register are loaded from the process picked up by the CPU scheduler to run next.
- In a multi programmed processor computing system, context switches occur frequently enough that all processes appear to be running concurrently.
- If a process has more than one thread, the O.S can use the context switching technique to schedule the threads so that they appear to execute parallel.

CPU Process scheduling

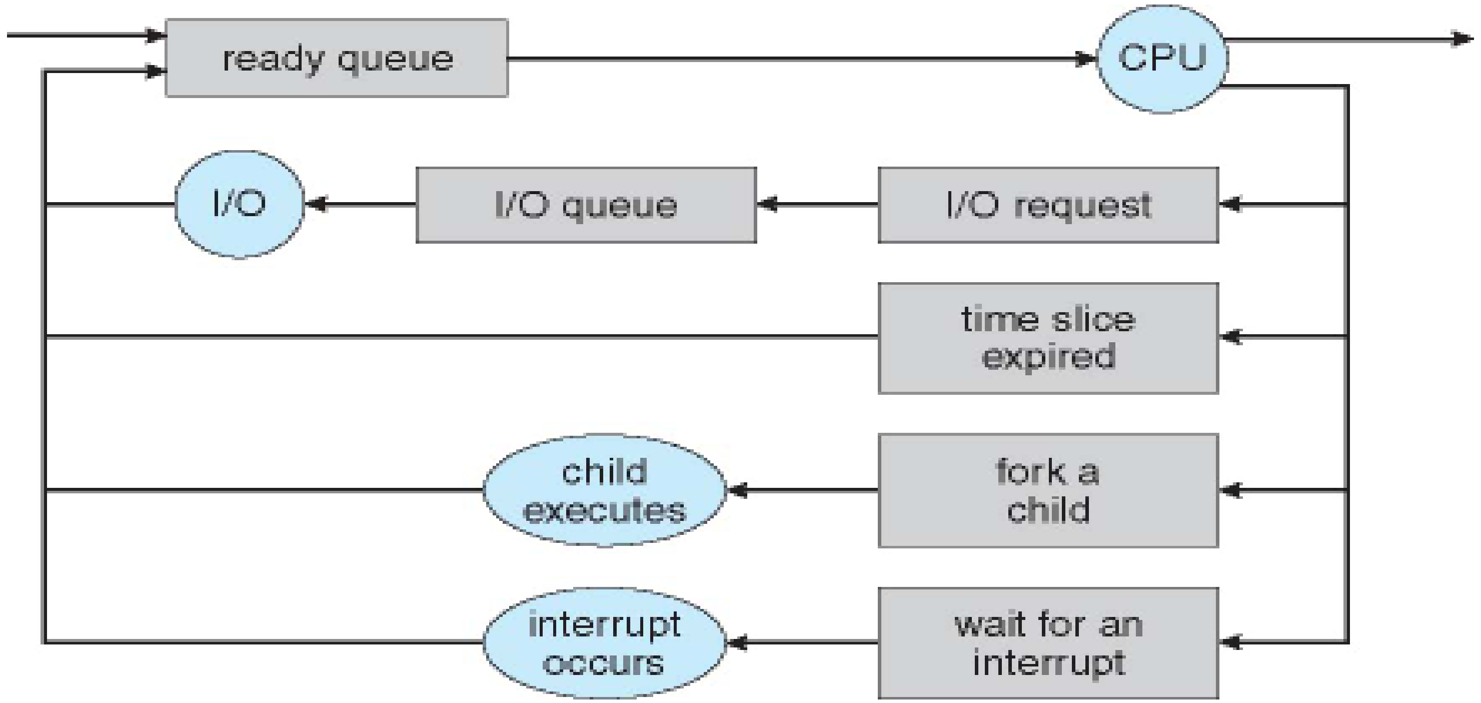
- The problem of determining when processor should be assigned and to which processes is called processor scheduling or CPU scheduling.
- When more than one process is run able the O.S must decide which one first.
- The part of O.S. concerned with this decision is called the scheduler and algorithm it uses the called scheduling algorithms.
- Select the next process to run
- Consider processes in the ready state
- Approaches different for batch systems and for time-sharing systems

Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - ➢**Job queue** – set of all processes in the system
  - ➢**Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - ➢**Device queues** – set of processes waiting for an I/O device
  - ➢Processes migrate among the various queues

Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows

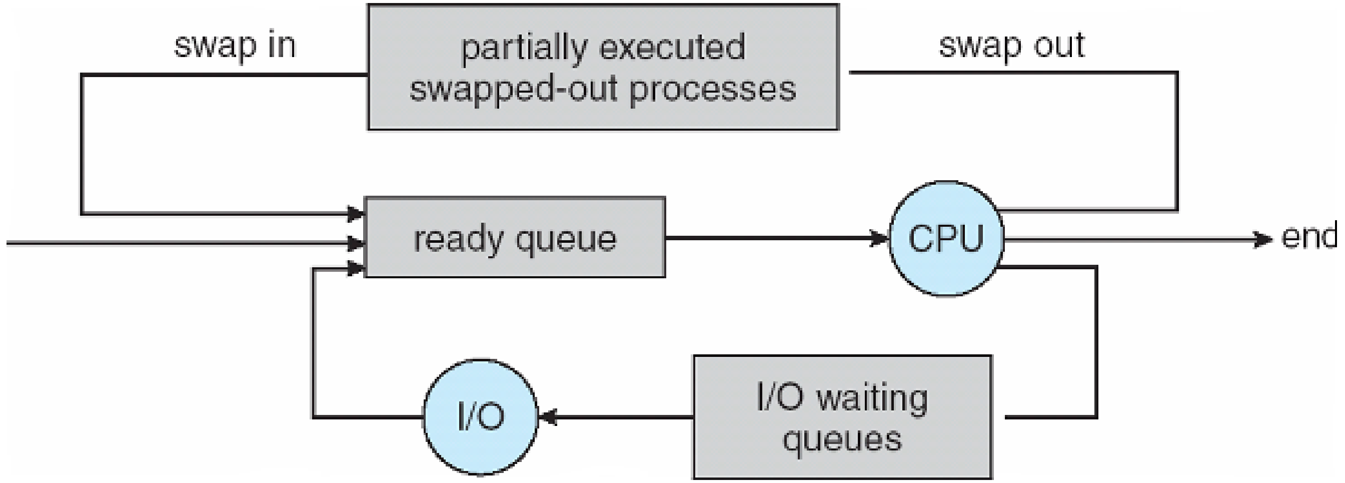


Schedulers

- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - The long-term scheduler controls the degree of multiprogramming
- Processes can be described as either:
  - I/O-bound process – spends more time doing I/O than computations, many short CPU bursts
  - CPU-bound process – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

Addition of Medium Term Scheduling

- Medium-term scheduler can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: swapping



CPU Scheduling or Process scheduling

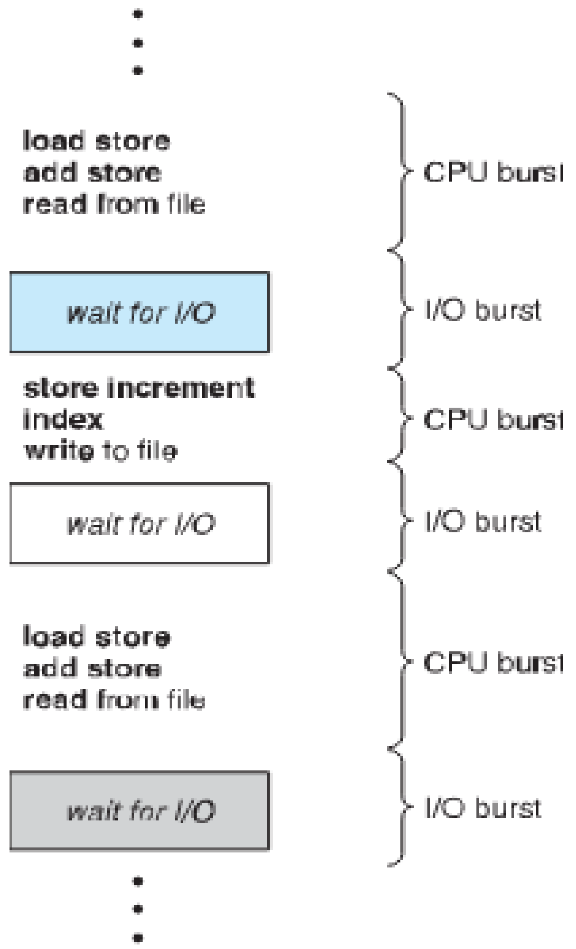
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation

Objectives

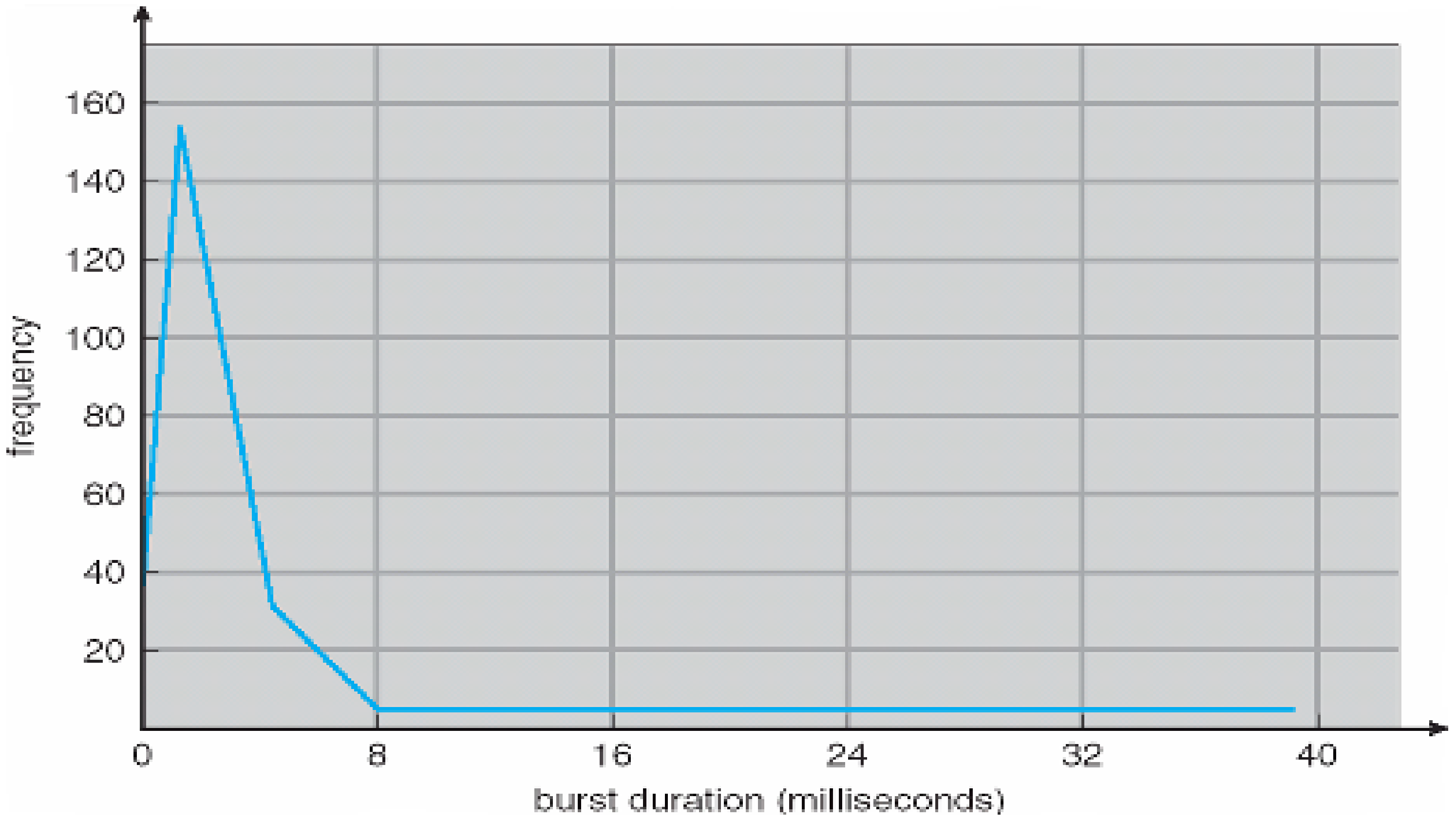
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait
- CPU burst followed by I/O burst
- CPU burst distribution is of main concern



Histogram of CPU-burst Times



Type of Scheduling

- ❑Scheduling algorithm can be divided into two categories with respect to how they deal with clock interrupt.
- ❑Non Preemptive scheduling
  - ➢Short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
  - ➢Response time are more predictable bcz incoming high priority jobs can not displace waiting jobs.
- ❑Preemptive scheduling
- ❑A scheduling discipline is preemptive if once a process has been given the CPU, it can be taken away.

CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- 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
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

CPU scheduling

- A process execution consist of a cycle of CPU execution and I/O execution.
- Normally every process begin with CPU burst that may be followed by I/O burst, then another CPU burst then I/O burst and so on. Eventually in the last will end up on CPU burst.
- CPU bound process: These are those processes which require most of time on CPU.
- I/O bound process: These are those processes which require most of time on I/O devices.
- A good scheduling idea should choose the mixture of both so that both I/O device and CPU can be utilized efficiently.

Scheduling Criteria CPU scheduling terminology

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Burst time/ execution time/ running time:** is the time process require for running on CPU
- **Waiting time**: time spend by a process in ready state waiting for CPU
- **Arrival time**: when a process enter the ready state
- **Exit time**: when process completes execution and exit from system
- **Turn around time**: total time spend by a process in the system
- TAT = ET – AT or BT + WT
- **Response time**: Time between a process enter ready queue and get scheduled on the CPU for the first time

Scheduling Criteria for algorithms

- ➢Average waiting time
- ➢Average response time
- ➢CPU Utilization
- ➢Throughput

Scheduling Algorithm Optimization Criteria

- ➢Max CPU utilization
- ➢Max throughput
- ➢Min turnaround time
- ➢Min waiting time
- ➢Min response time

First- Come, First-Served (FCFS) Scheduling

- Simplest scheduling algorithm, it assign CPU to the process which arrives first.
- Easy to understand and can easily be implemented using queue data structure.
- Always non-pre-emptive in nature.

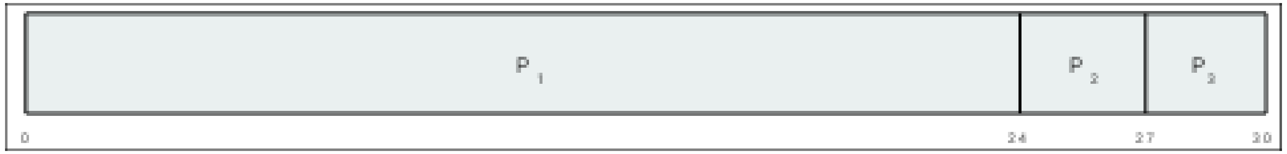
Process Id	Arrival time	Burst time
A	3	4
B	5	3
C	0	2
D	5	1
E	4	3

Process Id	Arrival time	Burst time	Turn around time	Waiting Time
A	3	4	4	0
B	5	3	8	5
C	0	2	2	0
D	5	1	9	8
E	4	3	6	3

ProcessBurst Time

P_1	24
P_2	3
P_3	3

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



- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

- Convoy effect: smaller process have to wait for long time for bigger process to release CPU.
- Advantage: simple, easy to use, easy to understand, easy to implement, must be used for background process where execution is not urgent.
- Disadvantage: Suffer from convoy effect, normally higher average waiting time, no consideration to priority or burst time, should not be used for interactive system.

PID	AT	BT
A	0	100
B	1	1

PID	AT	BT
A	1	100
B	0	1

Shortest Job First (SJF non-pre-emptive)/ Shor Remaining Time First (SRTF Pre-emptive)

- Out of all available process, CPU is assigned to the process having smallest burst time requirement (no priority, no seniority)
- If there is a tie, FCFS is used to break tie
- Can be used both with non-pre-emptive and pre-emptive approach
- Pre-emptive version (SRTF) is also called as optimal as it gurantee minimal average waiting time.

Pid	AT	BT	Non-pre-emptive		Pre-emptive	
			TAT	WT	TAT	WT
A	3	1	4	3	1	0
B	1	4	15	11	5	1
C	4	2	5	3	4	2
D	0	6	6	0	16	10
E	2	3	10	7	9	6

Shortest Job First (SJF non-pre-emptive)/ Shortest Remaining Time First (SRTF Pre-emptive)

- ❑Advantage: SRTF guarantees minimal average waiting time.
- ➢Provide a standard for other algorithm in terms of average waiting time.
- ➢Better average response time compare to FCFS
- ❑Disadvantage: Algorithm can not be implemented as there is no way to know the burst time of a process.
- ➢Process with longer CPU burst time requirement will go into starvation.
- ➢No idea of priority, process with large burst time have poor response time.

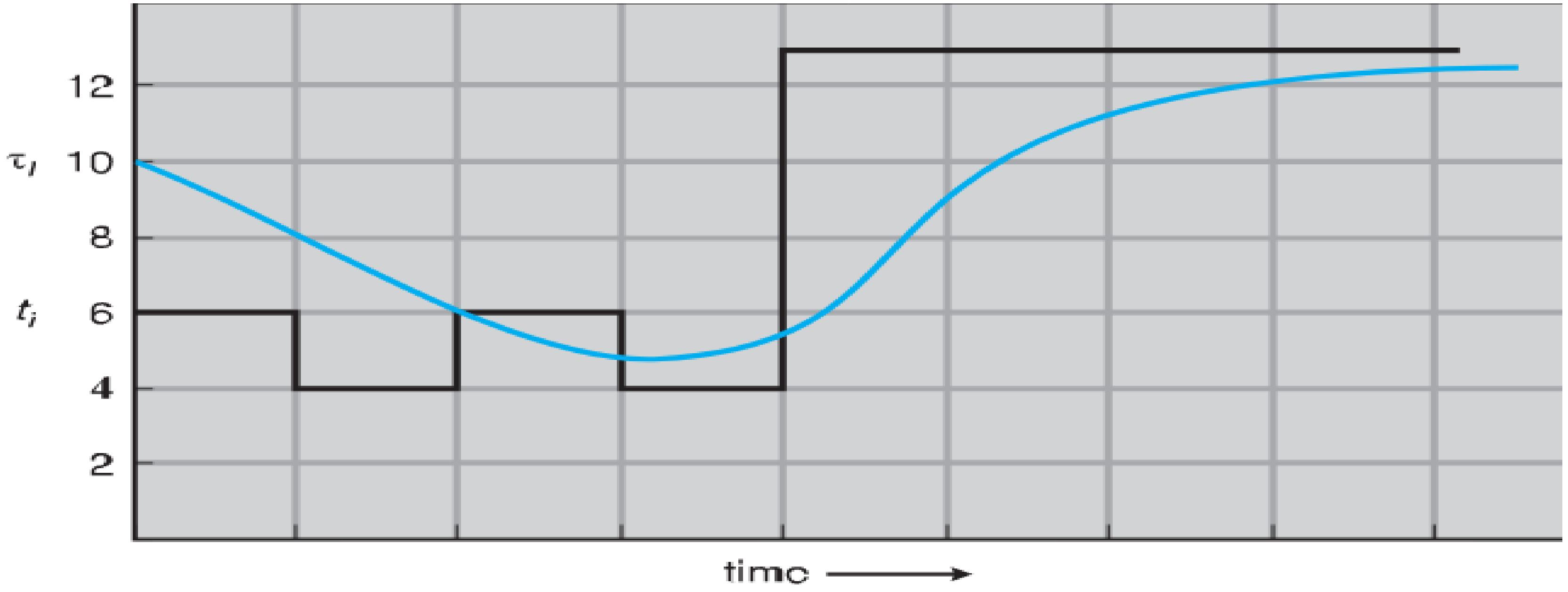
Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. α, $0 \leq \alpha \leq 1$
4. Define : $\tau_{n=1} = \alpha\, t_n + (1 - \alpha)\tau_n$.

- Commonly, α set to ½
- Preemptive version called shortest-remaining-time-first

Prediction of the Length of the Next CPU Bur



CPU burst (t_i)		6	4	6	4	13	13	13	...
'guess' (τ_i)	10	8	6	6	5	9	11	12	...

Priority Scheduling Algorithm

- Here a priority is associated with each process.
- At any instance of time out of all available process, CPU is allocated to the process which process the highest priority (Number may be higher or lower) .
- Tie is broken using FCFS order.
- No importance of arrival time or burst time.
- Support both non-preemptive and preemptive version.

Priority Scheduling Algorithm

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

Pid	AT	BT	Priority	Non-pre-emptive		Pre-emptive	
				TAT	WT	TAT	WT
A	0	4	2	4	0	15	11
B	1	3	3	14	11	11	8
C	2	1	4	10	9	1	0
D	3	5	5	6	1	5	0
E	4	2	5	7	5	6	4

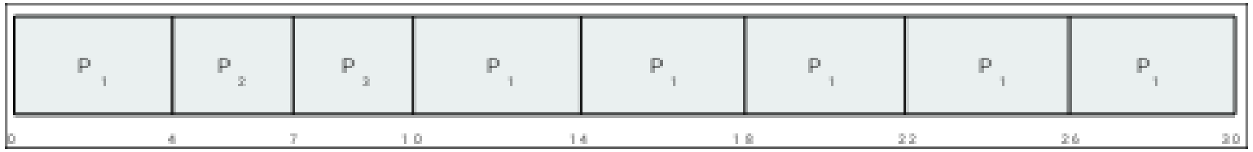
Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum *q*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- 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.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

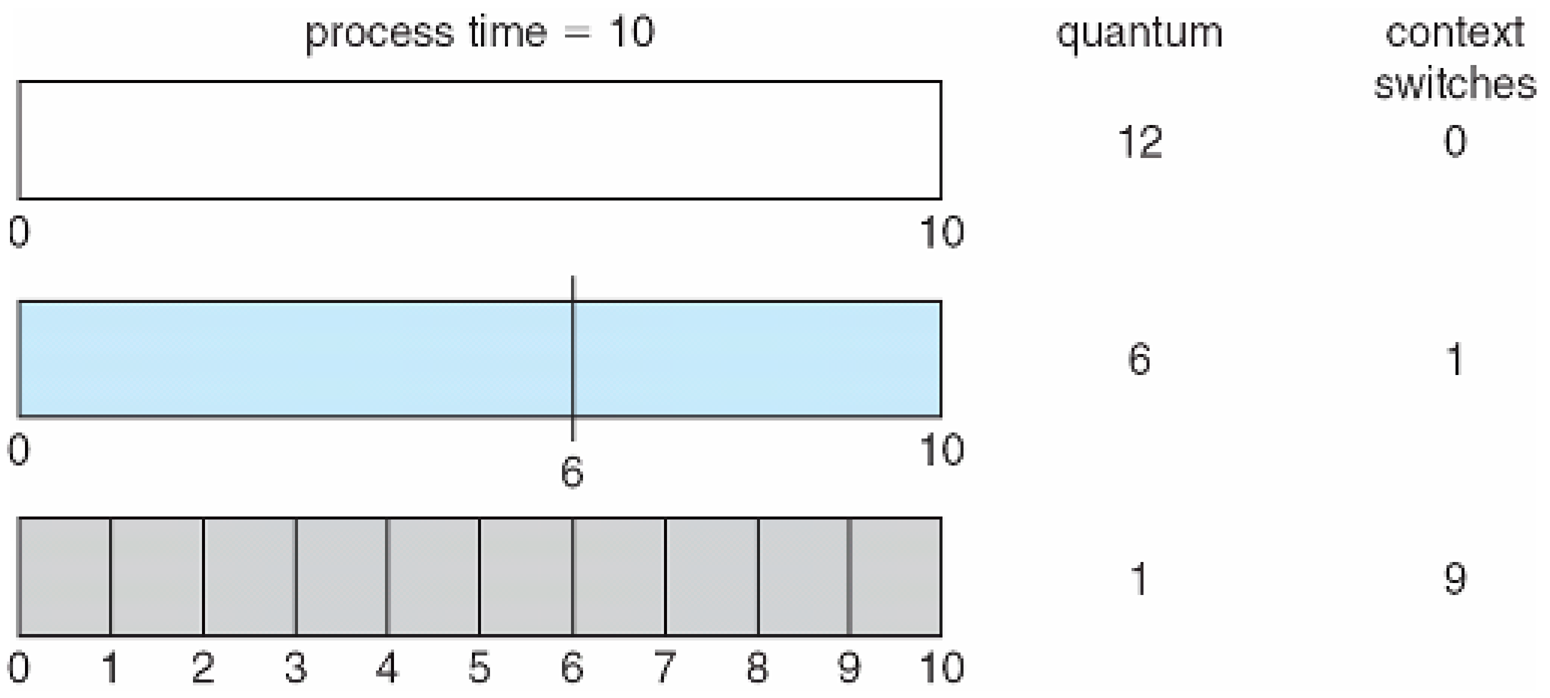
❑Advantage:

- ➢Perform best in terms of average response time.
- ➢Works will in case of time sharing system, client server architecture and interactive system.
- ➢Implemented with a circular queue
- ➢Kind of SJF implementation.

❑Disadvantage:

- ➢Longer process may starve
- ➢Performance depends heavily on time quantum
- ➢No idea of priority

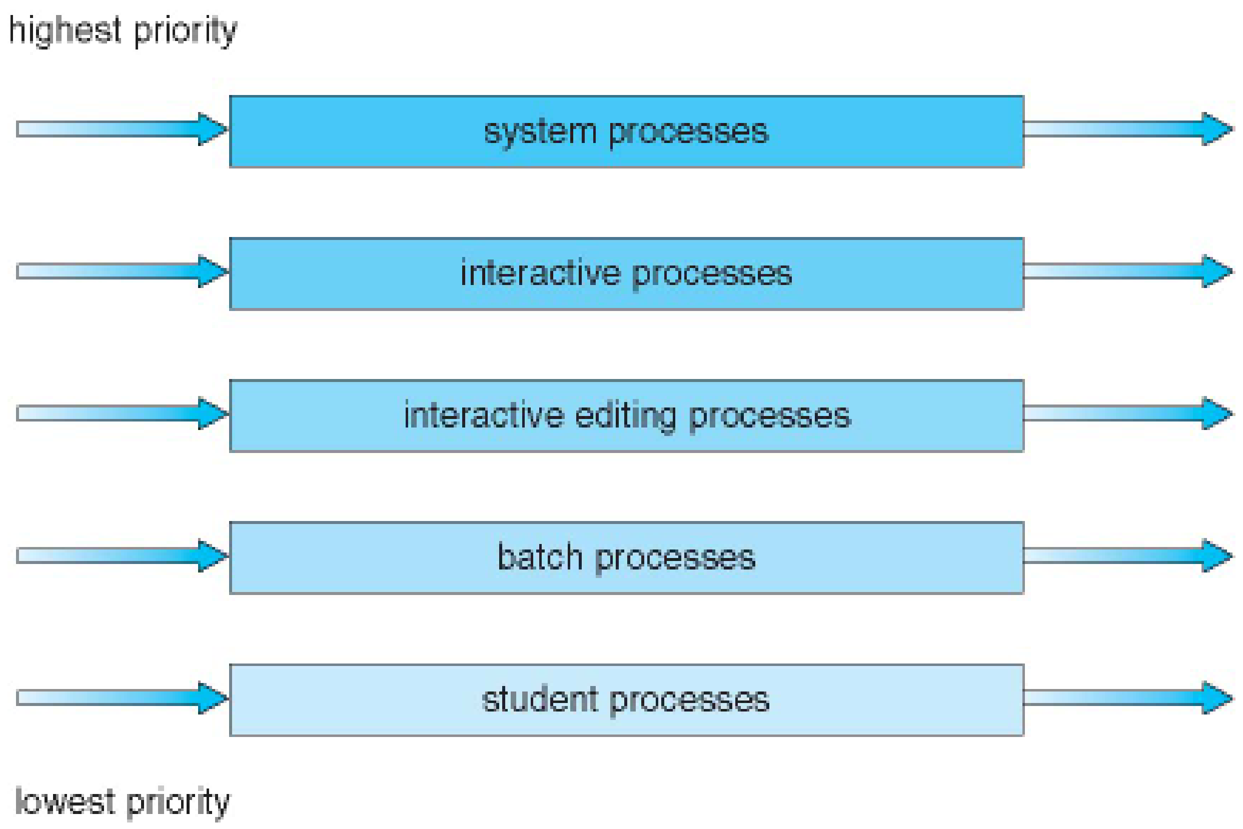
Time Quantum and Context Switch Time



Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- 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

Multilevel Queue Scheduling

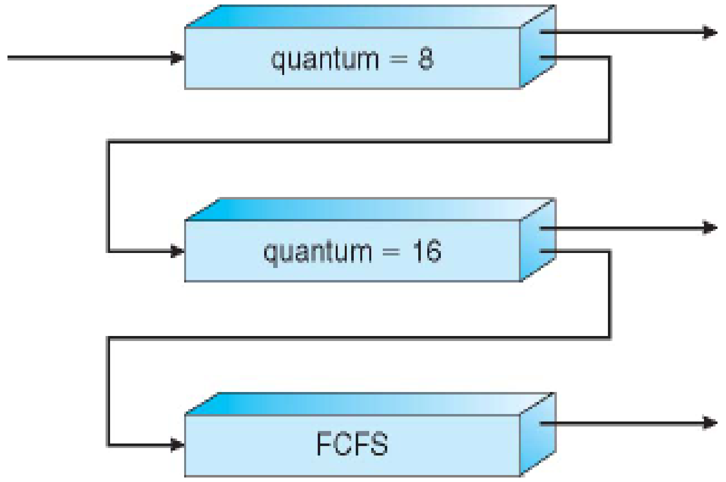


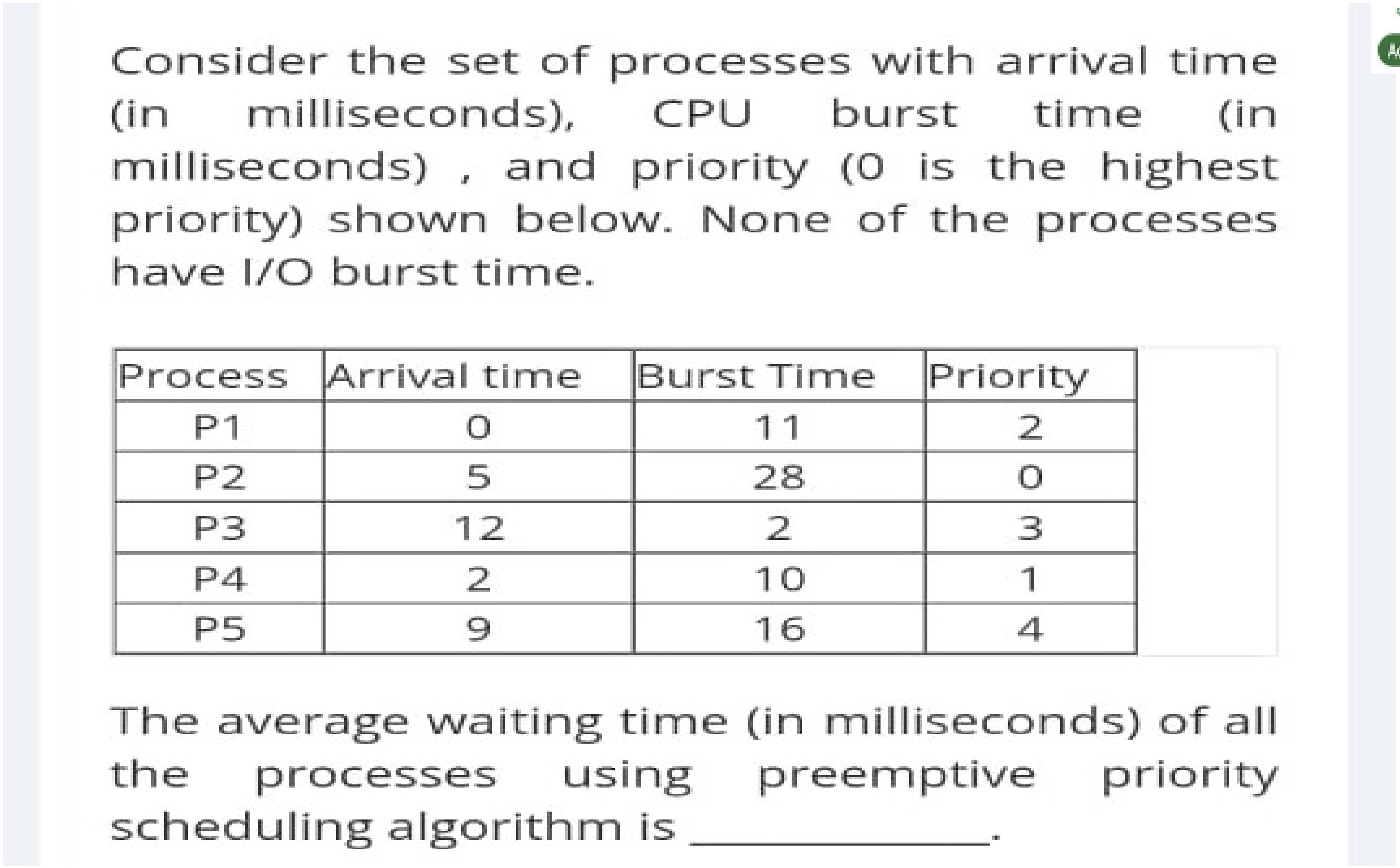
Multilevel Feedback Queue

- 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 used to determine when to upgrade a process
    - method used to determine when to demote a process
    - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - *A new job enters queue Q0 which is served FCFS*
  - *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 is again served FCFS and receives 16 additional milliseconds*
  - *If it still does not complete, it is preempted and moved to queue Q2*

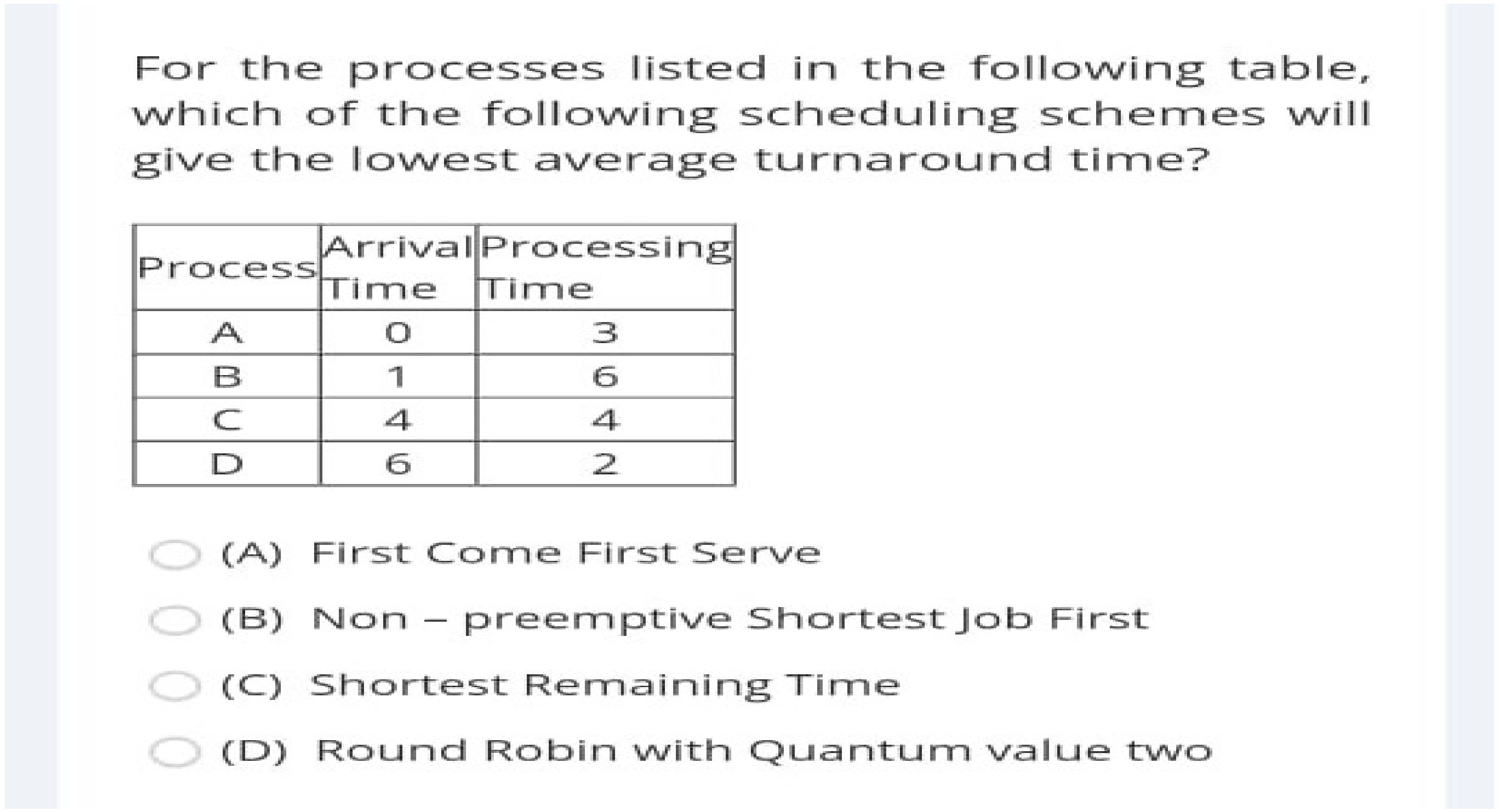




Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds) , and priority (0 is the highest priority) shown below. None of the processes have I/O burst time.

Process	Arrival time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is ______________.



For the processes listed in the following table, which of the following scheduling schemes will give the lowest average turnaround time?

Process	Arrival Time	Processing Time
A	0	3
B	1	6
C	4	4
D	6	2

○ (A) First Come First Serve

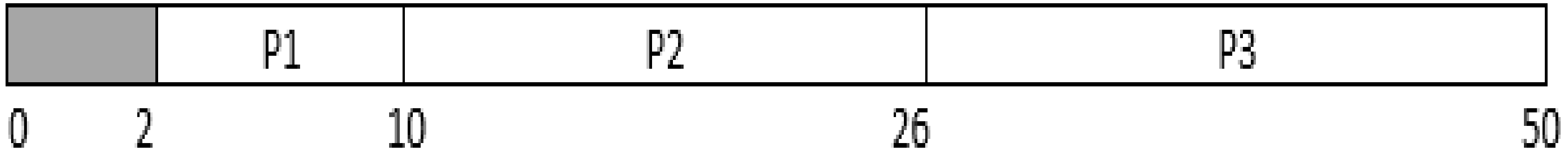
○ (B) Non – preemptive Shortest Job First

○ (C) Shortest Remaining Time

○ (D) Round Robin with Quantum value two

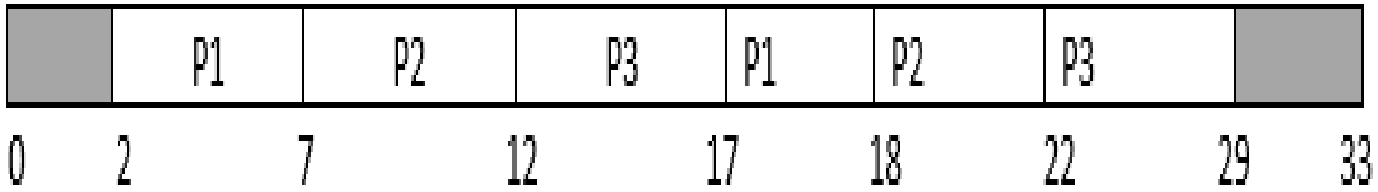
Q1. Three process P1, P2 and P3 arrive at time zero. The total time spent by the process in the system is 10ms, 20ms, and 30ms respectively. They spent first 20% of their execution time in doing I/O and the rest 80% in CPU processing. What is the percentage utilization of CPU using FCFS scheduling algorithm?

Process	Execution time	I/O time	CPU time
P1	10	2	8
P2	20	4	16
P3	30	6	24



Q2. Three process p1, P2 and P3 arrive at time zero. Their total execution time is 10ms, 15ms, and 20ms respectively. They spent first 20% of their execution time in doing I/O, next 60% in CPU processing and the last 20% again doing I/O. For what percentage of time was the CPU free? Use Round robin algorithm with time quantum 5ms.

Process	Execution time	I/O burst	CPU burst	I/O Burst
P1	10	2	6	2
P2	15	3	9	3
P3	20	4	12	4



Process Synchronization

- **Two types**
  - **Independent process**: Execution of the process does not affect the execution of other.
  - **Cooperative process:** Execution of one process affects the execution of other process.
- **Critical Section:** When more than one processes access a same code segment that segment is known as critical section. Critical section contains shared variables or resources which are needed to be synchronized to maintain consistency of data variable.

Race condition

❑In O.S. the processes that are working together shared some common storage (main memory file) that each process can read and write when two or more processes are reading or writing some shared data and the final result depend on who run precisely when, are called race conditions.

❑Concurrently executing threads that share data need to synchronize their operations and processing in order to thread at a time should be allowed to examine and update the shared variables.

❑Race condition are also possible in O.S. if the ready queue is implemented as a linked list and if the ready queue is manipulated during the handling of other interrupt before the first one completes. If the interrupts are not disable then the linked list could become corrupted.

How to avoid race condition?

- ❑The key to preempting trouble involving shared storage is to find some way to prohibit more than one process from reading and writing shared data simultaneously.
- ❑That part of the program where the shared memory is accessed is called the critical section.
- ❑To avoid race conditions and flowed results one must identify codes in critical section in each thread.