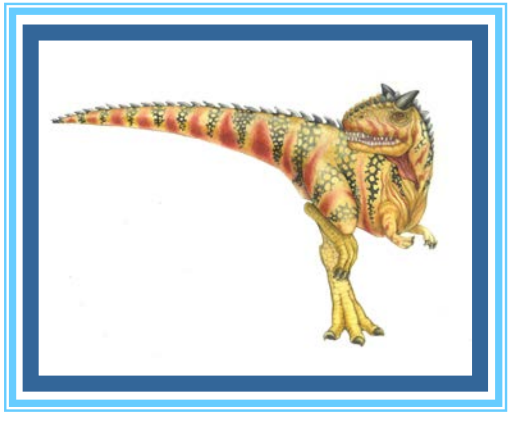
Chapter 6: CPU Scheduling



Chapter 6: CPU Scheduling

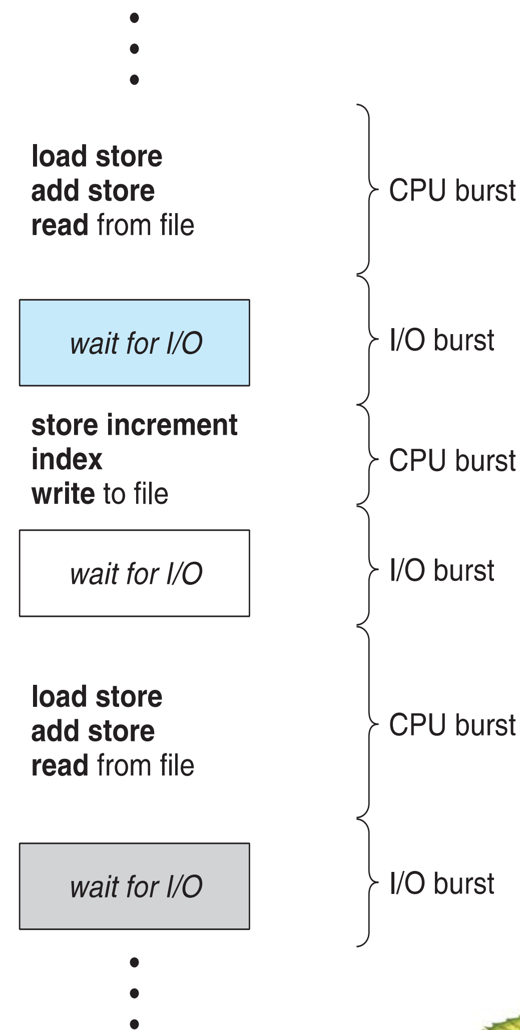
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling
- Algorithm Evaluation

Objectives

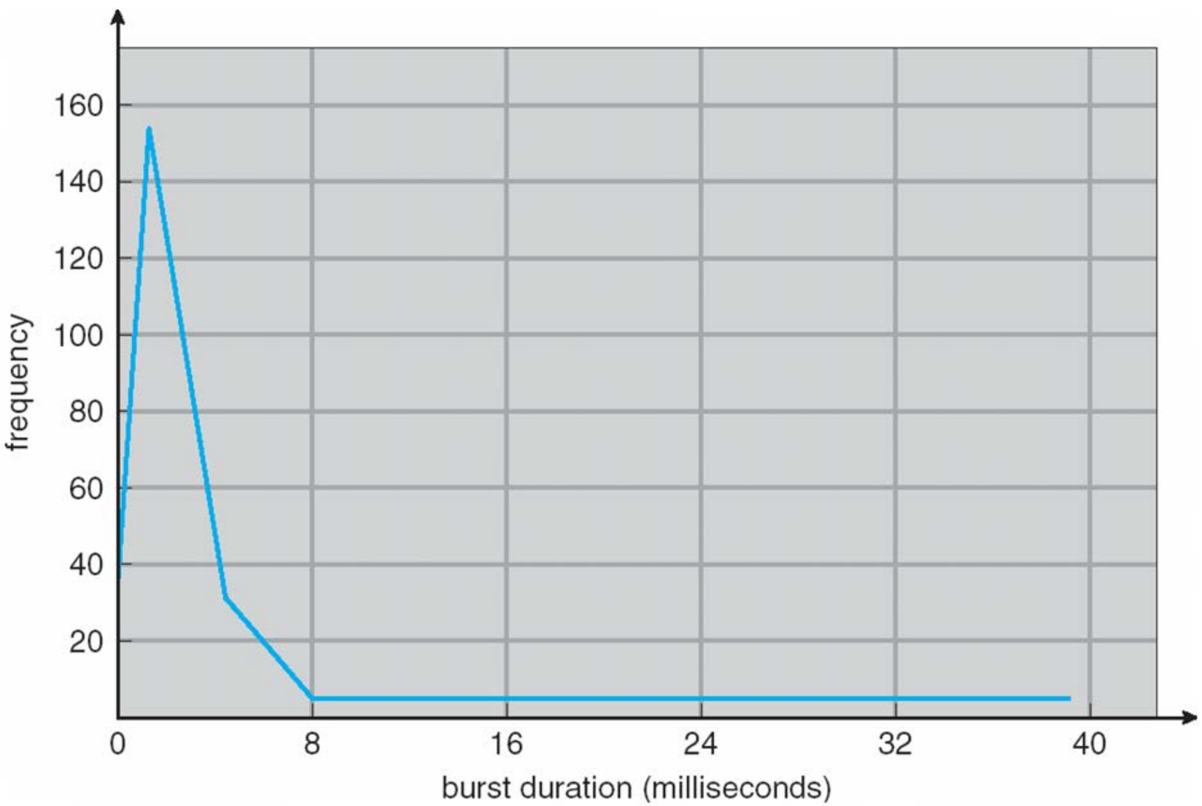
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various algorithms for CPU-scheduling
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
  - waiting for I/O is wasteful
  - 1 thread will utilize only 1 core
- 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

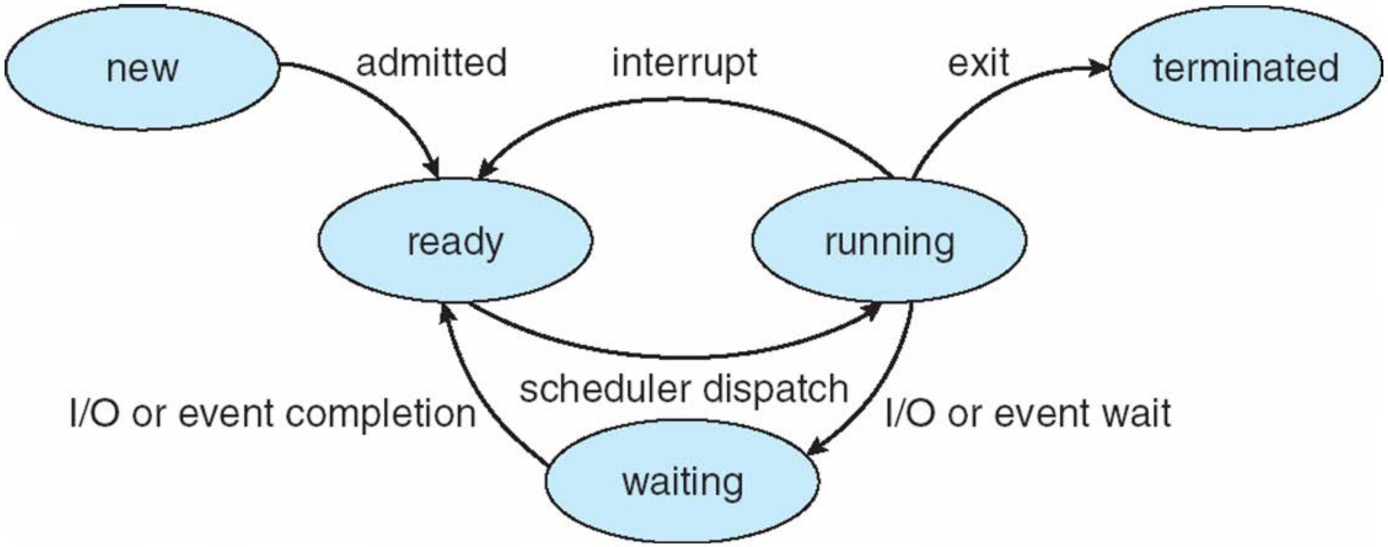


Histogram of CPU-burst Times of a Process



- Large number of short CPU bursts
- Small number of large CPU bursts
- Distribution can dictate a choice of an CPU-scheduling algo

Recap: Diagram of Process State



- As a process executes, it changes **state**
  - **new**: The process is being created
  - **ready**: The process is waiting to be assigned to a processor
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **terminated**: The process has finished execution

Levels of Scheduling

- High-Level Scheduling
  - See Long-term scheduler or Job Scheduling from Chapter 3
  - Selects jobs allowed to compete for CPU and other system resources.
- Intermediate-Level Scheduling
  - See Medium-Term Scheduling from Chapter 3
  - Selects which jobs to temporarily suspend/resume to smooth fluctuations in system load.
- Low-Level (CPU) Scheduling or Dispatching
  - Selects the ready process that will be assigned the CPU.
  - Ready Queue contains PCBs of processes.

CPU Scheduler

- **Short-term scheduler**
  - Selects 1 process from the ready queue
    - then allocates the CPU to it
  - 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 called **nonpreemptive (=cooperative)**
- All other scheduling is called **preemptive**
  - Process can be interrupted and must release the CPU
  - Special care should be taken to prevent problems that can arise
    - Access to shared data – race condition can happen, if not handled
    - Etc.

Dispatcher

- **Dispatcher**
  - a module that gives control of the CPU to the process selected by the short-term scheduler; this involves:
    - switching context
    - switching to user mode
    - jumping to the proper location in the user program to restart that program
- **Dispatch latency**
  - Time it takes for the dispatcher to stop one process and start another running
  - This time should be as small as possible

Scheduling Criteria

- How do we decide which scheduling algorithm is good?
- Many criteria for judging this has been suggested
  - Which characteristics considered can change significantly which algo is considered the best
- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes to stat responding
  - Used for interactive systems
  - Time from when a request was submitted until the first response is produced

Scheduling Algorithm Optimization Criteria

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

First- Come, First-Served (FCFS) Scheduling

Process	Burst 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

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
  - Hence, average waiting time of FCFS not minimal
  - And it may vary substantially
- FCFS is nonpreemptive
  - Not a good idea for timesharing systems
- FCFS suffers from the convoy effect, explained next

FCFS Scheduling: Convoy Effect

- **Convoy effect** – when several short processes wait for long a process to get off the CPU
- Assume
  - 1 long CPU-bound process
  - Many short I/O-bound processes
- Execution:
  - The long one occupies CPU
    - The short ones wait for it: no I/O is done at this stage
    - No overlap of I/O with CPU utilizations
  - The long one does its first I/O
    - Releases CPU
    - Short ones are scheduled, but do I/O, release CPU quickly
  - The long one occupies CPU again, etc
- Hence low CPU and device utilization

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU <u>burst</u>
  - SJF uses these lengths to schedule the process with the shortest time
- Notice, the <u>burst</u> is used by SJF,
  - **not** the process end-to-end running time
    - implied by word “job” in S**J**F
  - Hence, it should be called ``Shorted-Next-CPU-Burst”
  - However, “job” is used for historic reasons
- Two versions of SJF: preemptive and nonpreemptive
  - Assume
    - A new process Pnew arrives while the current one Pcur is still executing
    - The burst of Pnew is less than what is left of Pcur
  - Nonpreemptive SJF – will let Pcur finish
  - Preemptive SJF – wil preempt Pcur and let Pnew execute
    - This is also called shortest-remaining-time-first scheduling

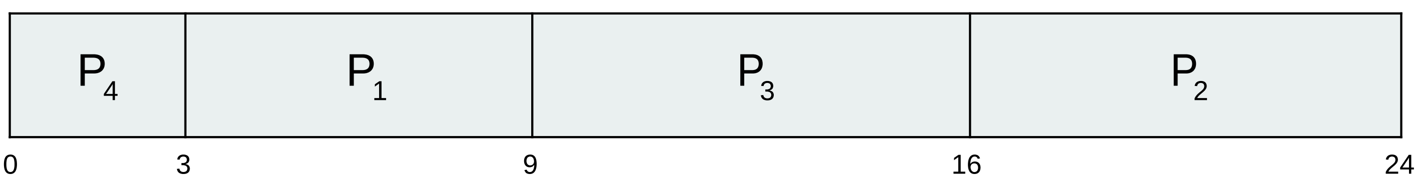
SJF (Cont.)

- **Advantage:**
  - SJF is **optimal** in terms of the average waiting time
- **Challenge of SJF:**
  - Hinges on knowing the length of the next CPU burst
    - But how can we know it?
    - Solutions: ask user or estimate it
  - In a batch system and long-term scheduler
    - Could ask the user for the job time limit
    - The user is motivated to accurately estimate it
      - Lower value means faster response
      - Too low a value will cause time-limit violation and job rescheduling
  - In a short-term scheduling
    - Use estimation
    - Will be explained shortly

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

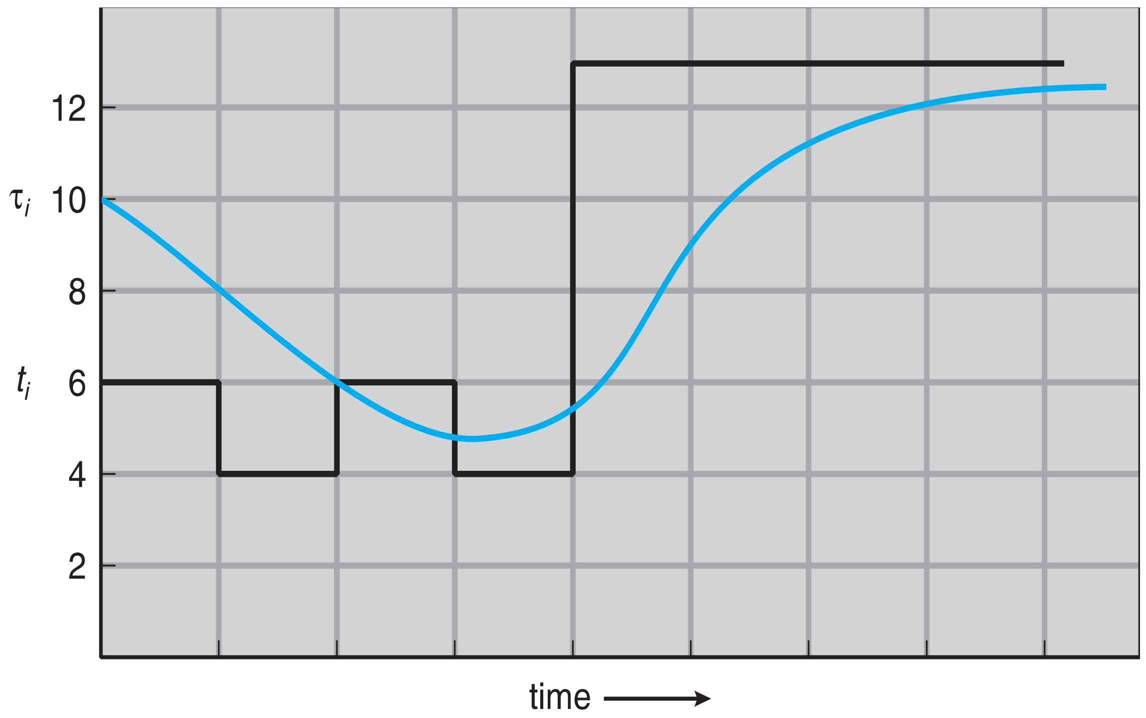
Estimating Length of Next CPU Burst

- For short-term scheduling SJF needs to estimate the burst length
  - Then pick process with shortest predicted next CPU burst
- Idea:
  - use the length of previous CPU bursts
  - apply exponential averaging

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

- Commonly, α set to ½

Prediction of the Length of the Next CPU Burst



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

Examples of Exponential Averaging

- $\tau_{n=1} = \alpha\, t_n + (1 - \alpha)\tau_n .$
- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \dots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \dots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart

P_1	P_2	P_4	P_1	P_3
0	1	5	10	17

(end: 26)

- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

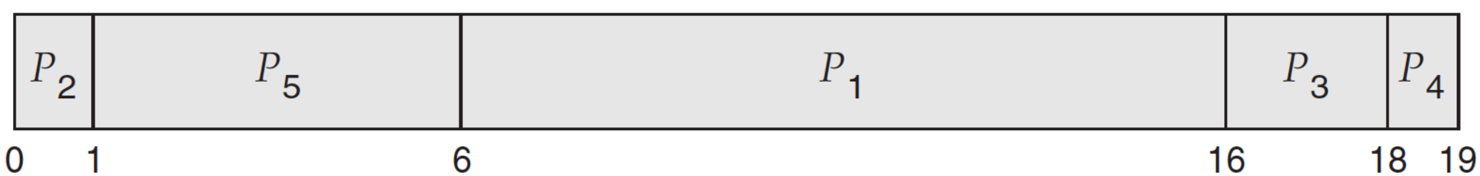
Priority Scheduling

- 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

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

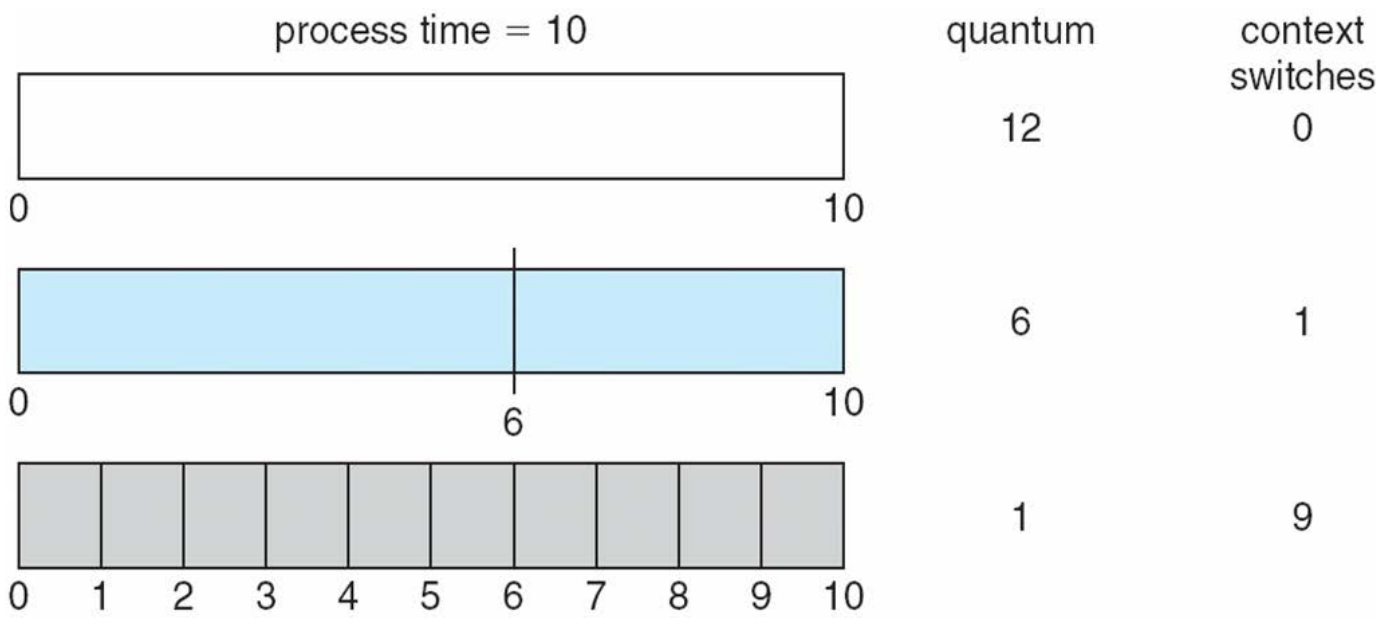
Round Robin (RR)

- 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
- The process might run for ≤ *q* time
  - For example, when it does I/O
- **If**
  - *n* processes in the ready queue, and
  - the time quantum is *q*
- **then**
  - “Each process gets 1/*n* of the CPU time”
    - Incorrect statement from the textbook
  - in chunks of ≤ *q* time units at once
  - Each process waits ≤ (*n*-1)*q* time units

Round Robin (cont.)

- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ overhead of context switch time is too high
- Hence, q should be large compared to context switch time
  - q usually 10ms to 100ms,
  - context switch < 10 usec

Time Quantum and Context Switch Time



The smaller the quantum, the higher is the number of context switches.

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1
0–4	4–7	7–10	10–14	14–18	18–22	22–26	26–30

0 4 7 10 14 18 22 26 30

- Typically:
  - Higher average turnaround (end-to-end running time) than SJF
  - But better ***response*** than SJF

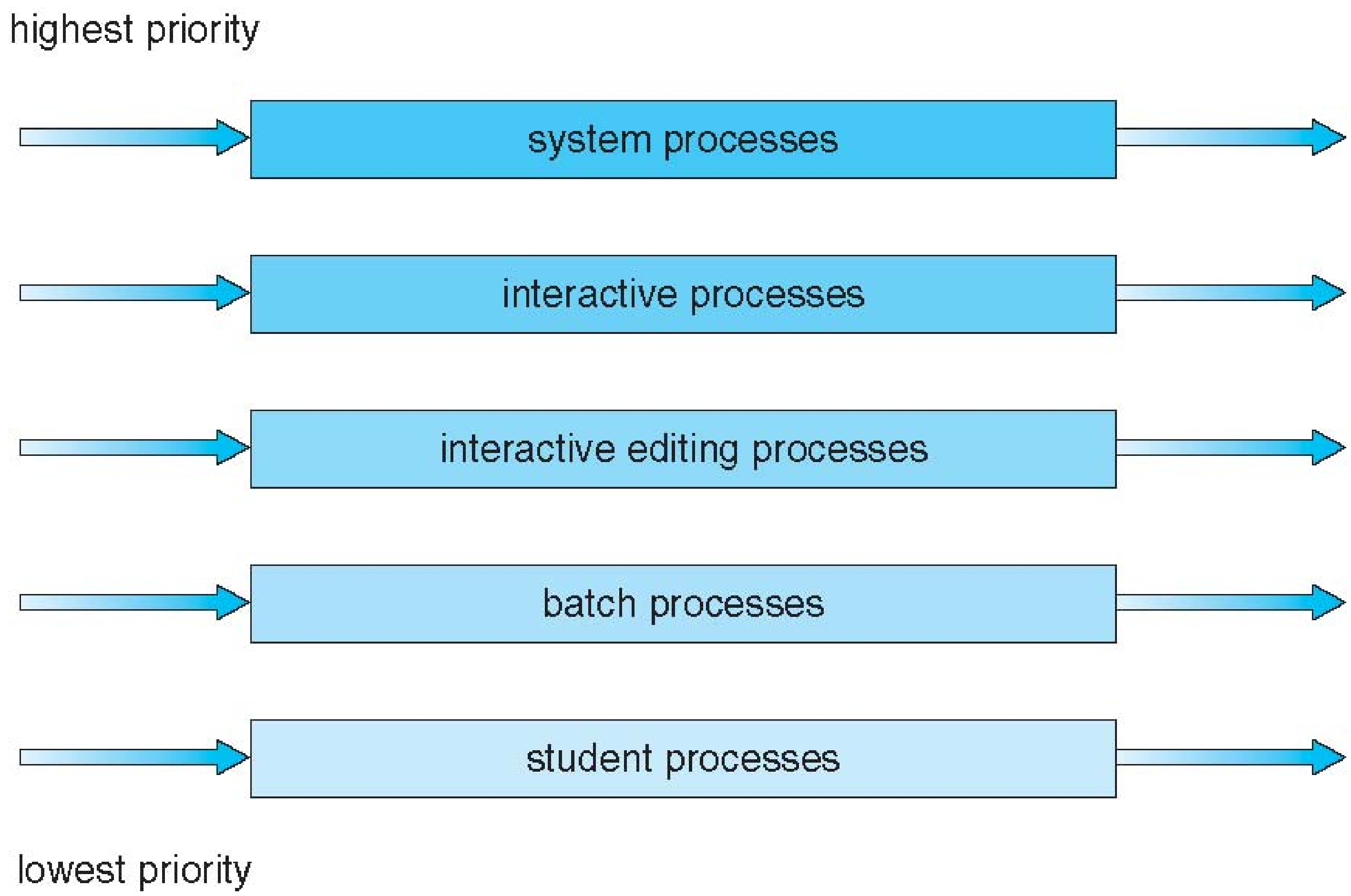
Multilevel Queue

- Another class of scheduling algorithms when processes are classified into groups, for example:
  - **foreground** (interactive) processes
  - **background** (batch) processes
- Ready queue is partitioned into separate queues, e.g.:
  - **Foreground** and **background** queues
- Process is permanently assigned to one queue
- Each queue has its own scheduling algorithm, e.g.:
  - foreground – RR
  - background – FCFS

Multilevel Queue

- Scheduling must be done between the queues:
  - Fixed priority scheduling
    - For example, foreground queue might have absolute priority over background queue
      - Serve all from foreground then from background
      - Possibility of starvation
  - Time slice scheduling
    - Each queue gets a certain amount of CPU time which it can schedule amongst its processes, e.g.:
      - 80% to foreground in RR
      - 20% to background in FCFS

Multilevel Queue Scheduling



No student process can run until all queues above are empty

Multilevel Feedback Queue

- The previous setup: a process is permanently assigned to one queue
  - **Advantage:** Low scheduling overhead
  - **Disadvantage:** Inflexible

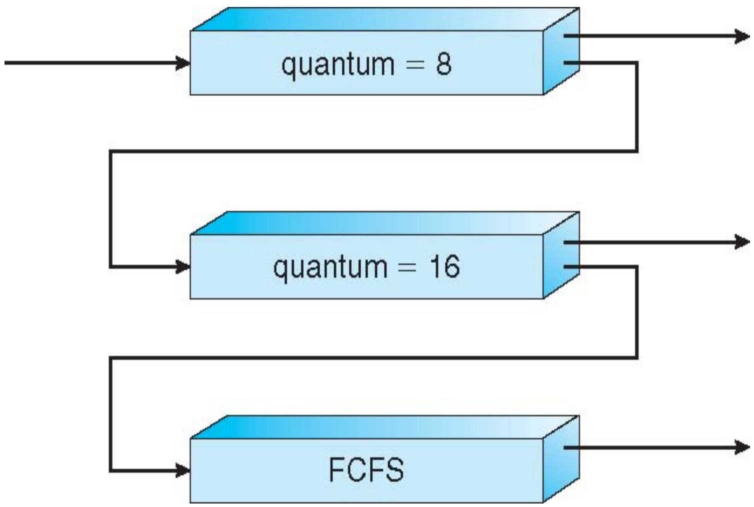
- Multilevel-feedback-queue scheduling algorithm
  - Allows a process to move between the various queues
    - More flexible
  - **Idea:** separate processes based on the characteristics of their CPU bursts
  - If a process uses too much CPU time => moved to lower-priority queue
    - Keeps I/O-bound and interactive processes in the high-priority queue
  - A process that waits too long can be moved to a higher priority queue
    - This form of aging can prevent starvation

Multilevel Feedback Queue

- 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
- Multilevel-feedback-queue scheduler
  - The most general CPU-scheduling algorithm
  - It can be configured to match a specific system under design
  - Unfortunately, it is also the most complex algorithm
    - Some means are needed to select values for all the parameters

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
  - A process in Q_1 will preempt any process from Q_2, but will be executed only if Q_0 is empty
- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 ms
    - If it does not finish in 8 milliseconds
      - job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - This happens only if is Q_0 empty
    - If it still does not complete, it is preempted and moved to queue Q_2
  - Processed in Q_2 run only when Q_0 and Q_1 empty
- In this example priority is given to processes with bursts less than 8 ms.
- Long processed automatically sink to queue Q_2



Multiple-Processor Scheduling

- Multiple CPUs are available
  - Load sharing becomes possible
  - Scheduling becomes more complex
- Solutions: Have one ready queue accessed by each CPU
  - Self scheduled - each CPU dispatches a job from ready Q
    - Called symmetric multiprocessing (SMP)
    - Virtually all modern OSes support SMP
  - Master-Slave - one CPU schedules the other CPUs
    - The others run user code
    - Called asymmetric multiprocessing
    - One processor accesses the system data structures
      - Reduces the need for data sharing

Real-Time CPU Scheduling

- Special issues need to be considered for real-time CPU scheduling
  - They are different for soft vs hard real-time systems
- **Soft real-time systems**
  - Gives preference to critical processed over over non-critical ones
  - But no guarantee as to when critical real-time process will be scheduled
- **Hard real-time systems**
  - Task must be serviced by its deadline
  - Otherwise, considered failure
- Real-time systems are often event-driven
  - The system must detect the event has occurred
  - Then respond to it as quickly as possible
  - Event latency – amount of time from when event occurred to when it is services
  - Different types of events will have different event latency requirements

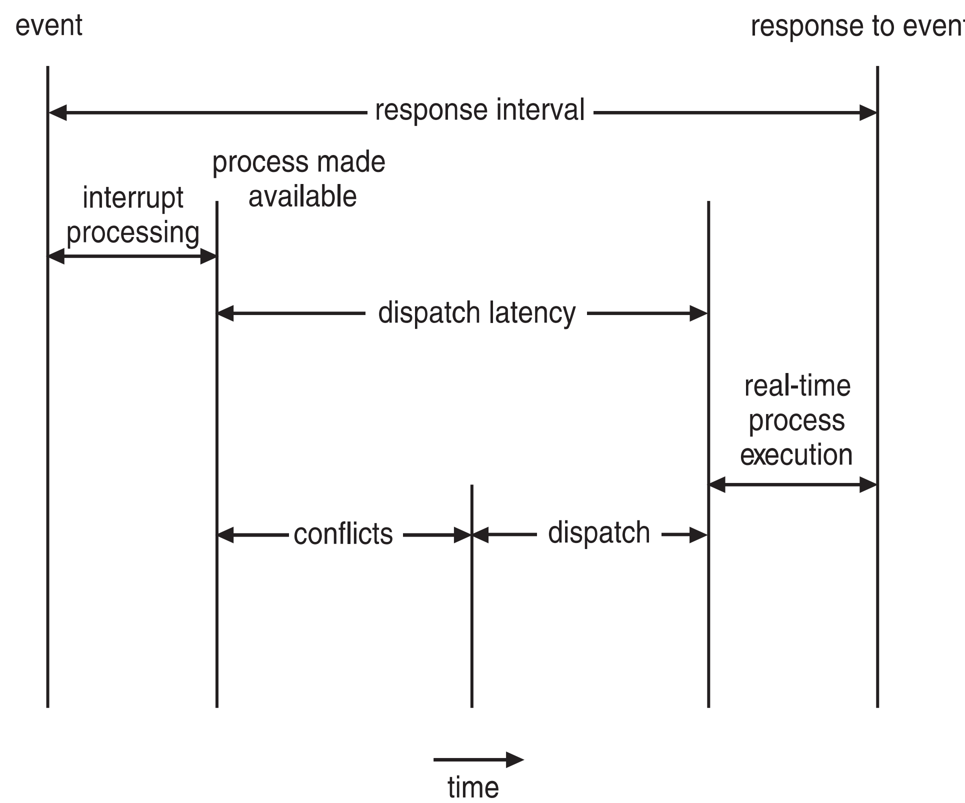
Real-Time CPU Scheduling

- Two types of latencies affect performance

1. Interrupt latency
    - time from arrival of interrupt to start of routine that services interrupt
    - Minimize it for soft real-time system
    - Bound it for hard real-time
2. Dispatch latency
    - time for scheduler to take current process off CPU and switch to another
    - Must also be minimized

Real-Time CPU Scheduling (Cont.)

- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode
  2. Release by low-priority process of resources needed by high-priority processes



Priority Inversion and Inheritance

- Issues in real-time scheduling
- **Problem:** Priority Inversion
  - Higher Priority Process needs kernel resource currently being used by another lower priority process
    - higher priority process must wait.
- **Solution:** Priority Inheritance
  - Low priority process now inherits high priority until it has completed use of the resource in question.

Many Different Real-Time Schedulers

- Priority-based scheduling
- Rate-monotonic scheduling
- Earliest-deadline scheduling
- Proportional share scheduling
- …

Algorithm Evaluation

- How to select CPU-scheduling algorithm for an OS?
  - Determine criteria, then evaluate algorithms
- Evaluation Methods
  - Deterministic modeling
  - Queuing models
  - Simulations
  - Implementation

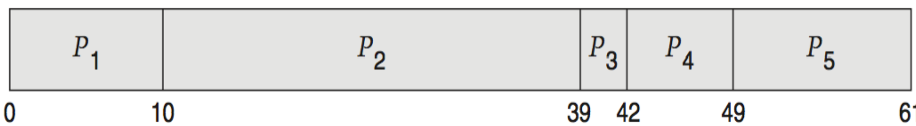
Deterministic Modeling

- **Analytic evaluation** – class of evaluation methods such that
  - **Given:** scheduling algorithm A and system workload W
  - **Produces:** formula or a number to evaluate the performance of A one W
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

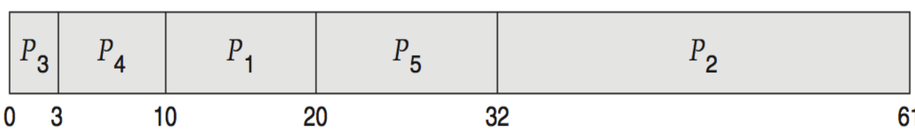
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Deterministic Evaluation

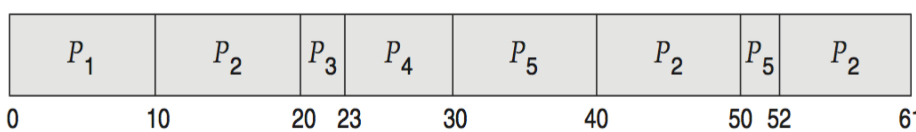
- Find which algorithm gets the minimum of the average waiting time
  - FCFS is 28ms:



  - Non-preemptive SFJ is 13ms:



  - RR is 23ms:



  - **Pros:** Simple and fast
  - **Cons:** Requires exact workload, the outcomes apply only to that workload

Queueing Models

- Defines a probabilistic model for
  - Arrival of processes
  - CPU bursts
  - I/O bursts
- Computes stats
  - Such as: average throughput, utilization, waiting time, etc
  - For different scheduling algorithms

Little's Formula

- n = average queue length
- W = average waiting time in queue
- λ = average arrival rate into queue
- Little's law – in steady state, processes leaving queue must equal processes arriving, thus:

  $$n = \lambda \times W$$

  - Valid for any scheduling algorithm and arrival distribution
- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

Simulations

- **Simulations** more accurate evaluation of scheduling algorithms
  - than limited Queuing models
- Need to program a model of computer system
- Clock is represented as a variable
  - As it increases, the simulator changes the state of the system
- Gather statistics indicating algorithm performance during simulation
- Data to drive simulation gathered via
  - Random number generator according to probabilities
  - Distributions defined mathematically or empirically
  - Use trace tapes - records of sequences of real events in real systems
    - This sequence is used then to drive the simulation

Implementation

- Even simulations have limited accuracy
- Just implement new scheduler and test in real systems
  - **Cons**: Environments vary over time – e.g., users might see a new scheduler and change the way their programs behaves, thus changing the environment

- In general, scheduling needs might be different for different sets of apps
  - Hence, most flexible schedulers are those can be modified/tuned for specific apps or a set of apps
  - For example, some versions of UNIX allow sysadmins to fine-tune the scheduling parameters

End of Chapter 6