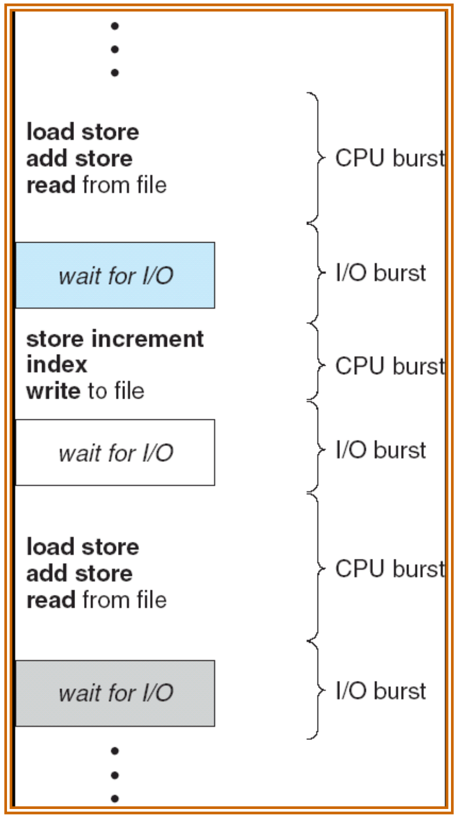
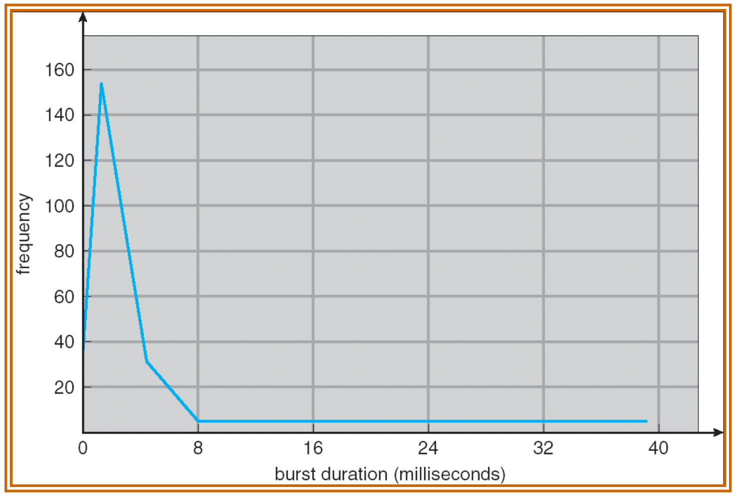
CPU Scheduling

Basic Concepts (1)

- Multiprogramming requires CPU scheduling
- CPU-I/O burst cycle
- CPU scheduler
  - Short-term scheduler
  - Ready queue may be a FIFO queue, priority queue, tree, linked list, etc. of PCBs





Basic Concepts (2)

- 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
- Preemptive scheduling requires synchronisation
- Preemptive scheduling affects kernel design
  - What happens if you are prempted while in a system call?
  - Problematic in real-time computing
- Interrupt handling

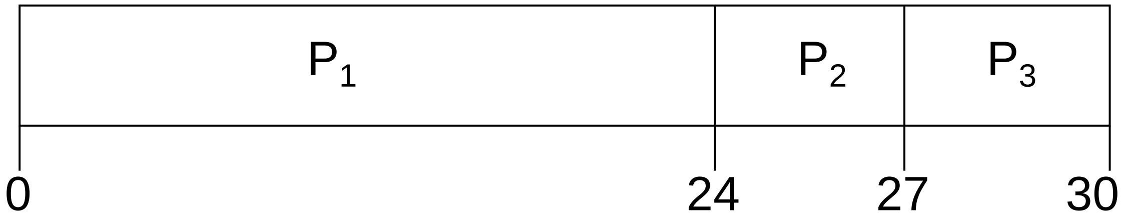
- Dispatcher: gives control of the CPU to the process selected by the scheduler
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart it
  - Dispatch latency: the time taken to stop one process and start another

Scheduling Criteria

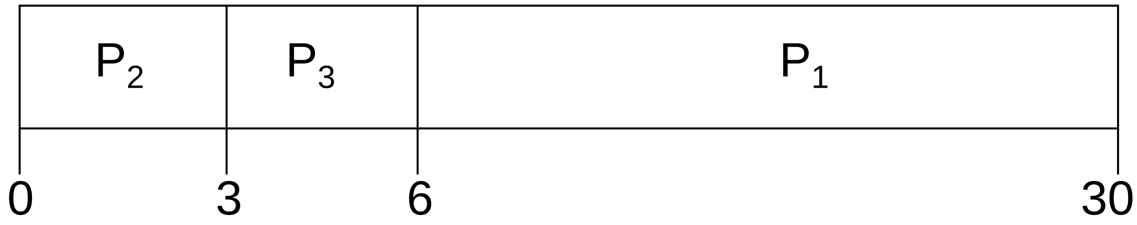
- CPU utilisation: % of time the CPU is used
  - Keep the CPU busy
- Throughput: no. of completed process per unit of time
- Turnaround time: interval from process submission to process completion
- Waiting time: amount of time spent waiting in ready queue
- Response time: time from submission to first response
  - Limited by the speed of the output device
- Maximise: utilisation, throughput
- Minimise: turnaround, waiting and response times
  - Usually optimise the average
  - Sometime optimise minimum or maximum
  - Minimise variance?

Scheduling Algorithms (1)

- First-Come, First-Served
  - FIFO ready queue
  - Non-preemptive scheduling
  - Substantially variable average waiting time
  - Suffers from the convoy effect

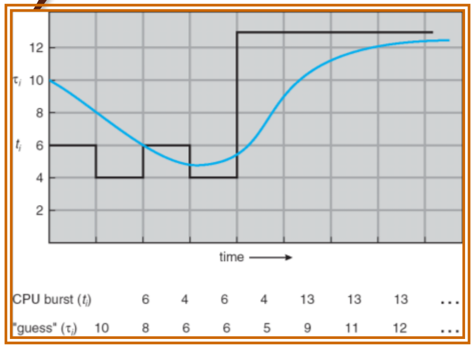


  - Average waiting time: (0 + 24 + 27)/3 = 17



  - Average waiting time: (6 + 0 + 3)/3 = 3

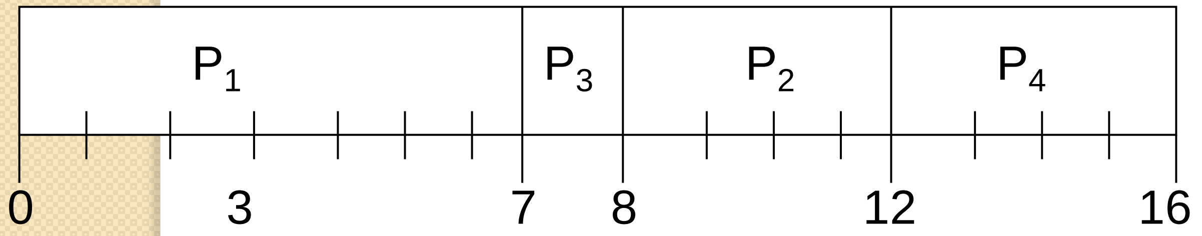
Scheduling Algorithms (2)



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

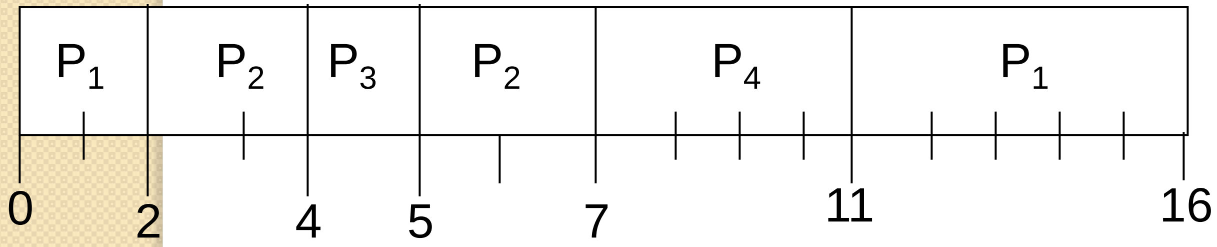
- Shortest-job-first
  - Length of next CPU burst
    - FCFS for tie breaks
  - Provably optimal in average waiting time
  - Often used in long-term scheduling, impossible to use in short-term scheduling
    - Predict bursts as exponential averages

$$\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n .$$



Non-preemptive

a.w.t. = (0 + 6 + 3 + 7)/4 = 4



Preemptive

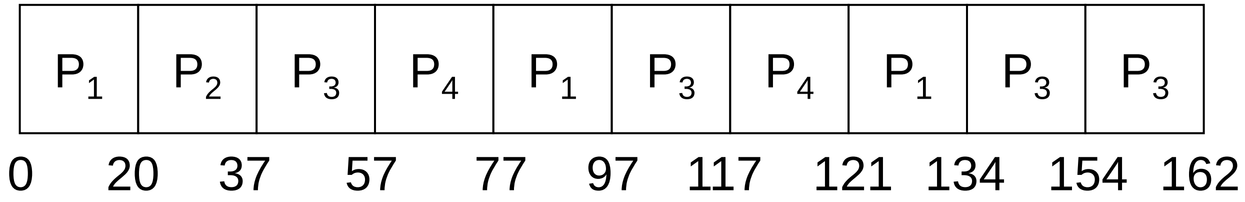
a.w.t. = (9 + 1 + 0 +2)/4 = 3

Scheduling Algorithms (3)

- Priority
  - Assign priorities to processes
    - FCFS for tie-break
    - SJF special case
  - Low versus high priority – low/high number?
  - Internal versus external priorities
    - Time limits, memory requirements, no. of open files, ratio of average I/O to CPU bursts
    - Importance, money paid, sponsoring department, other political factors
  - Starvation is a major problem – solution aging
    - Increase priority every period of time

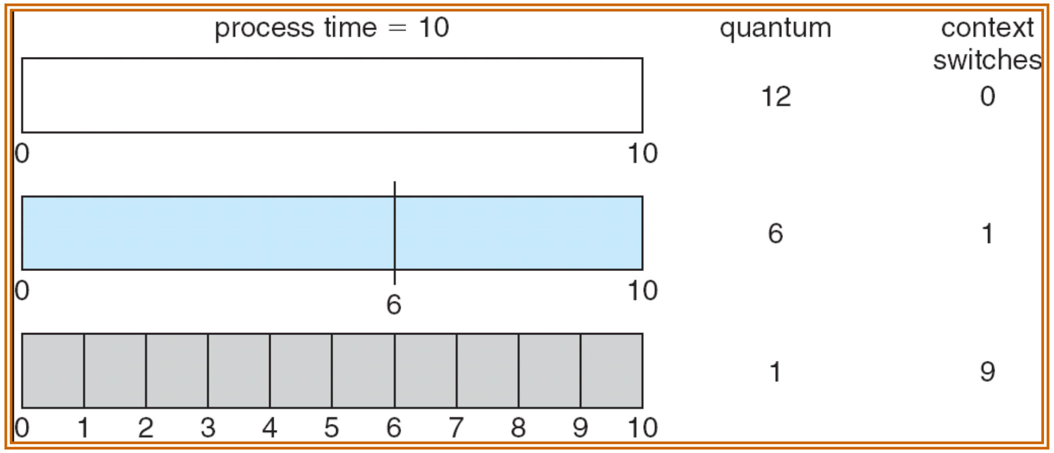
Scheduling Algorithms (4)

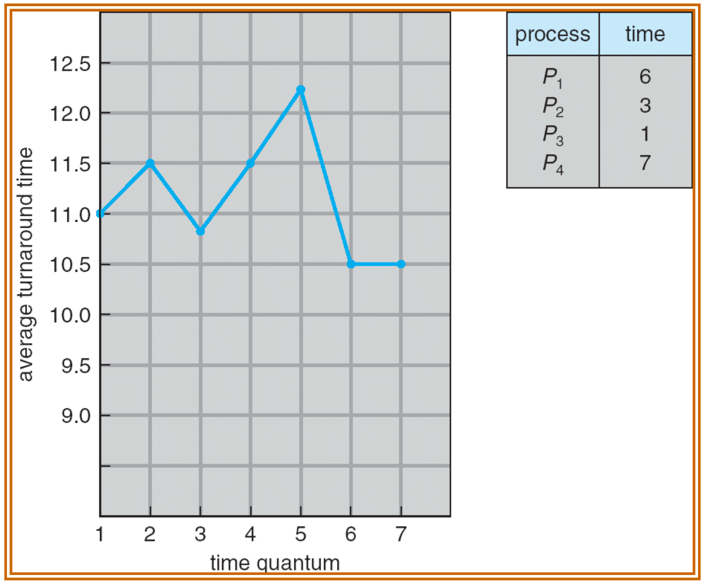
- Round-Robin
  - Designed for time-sharing systems
  - FCFS with preemption every time quantum/slice (10 – 100 ms)
  - Ready queue as a circular queue



  - Average waiting time is often long, but average response time is quite short
  - Size of time quantum determines performance from FCFS to processor sharing
    - Overhead of context switch

Scheduling Algorithms (5)

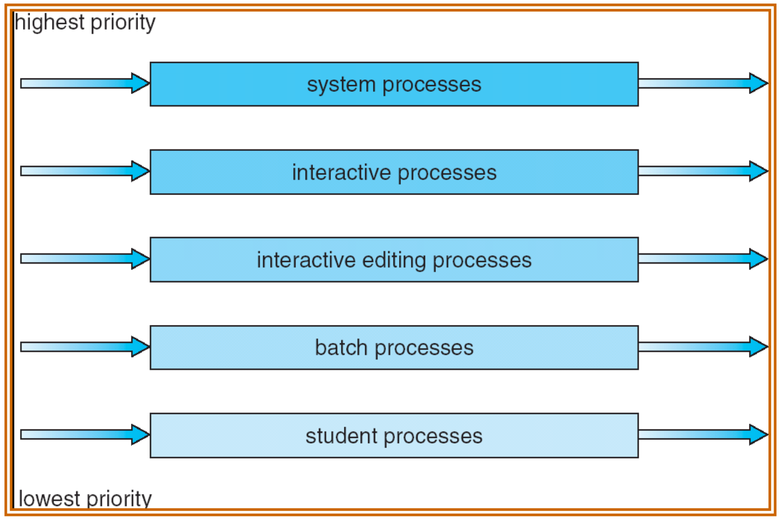




process	time
P_1	6
P_2	3
P_3	1
P_4	7

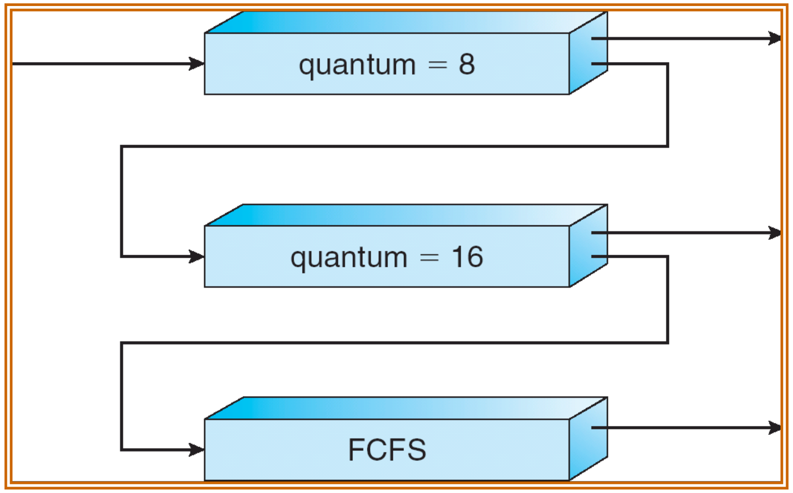
Scheduling Algorithms (6)

- Multi-level queue
  - Example: foreground and background processes
  - Partition the ready queue into a number of queues
  - Each queue with its own scheduling algorithm
    - Example: foreground RR, background FCFS
  - Scheduling between the queue: fixed priority preemptive scheduling or timeslice between queues
    - Example: foreground always has priority over background, or foreground 80%, background 20%



Scheduling Algorithms (7)

- Multilevel feedback queue
  - Increase scheduling overhead for flexibility
  - Parameters
    - Number of queues
    - Scheduling algorithm for each queue
    - Process upgrade method
    - Process demotion method
    - Process entry method



For contemplation (1)

- Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?
- Discuss how the following pairs of scheduling criteria conflict in certain settings.
  - CPU utilization and response time
  - Average turnaround time and maximum waiting time
  - I/O device utilization and CPU utilization
- Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
*P*1	10	3
*P*2	1	1
*P*3	2	3
*P*4	1	4
*P*5	5	2

- The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.
  - Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.
  - What is the turnaround time of each process for each of the scheduling algorithms in part a?
  - What is the waiting time of each process for each of the scheduling algorithms in part a?
  - Which of the schedules in part a results in the minimal average waiting time (over all processes)?

For contemplation (2)

- Which of the following scheduling algorithms could result in starvation?
    - First-come, first-served
    - Shortest job first
    - Round robin
    - Priority
- Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.
    - What would be the effect of putting two pointers to the same process in the ready queue?
    - What would be the major advantages and disadvantages of this scheme?
    - How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

For contemplation (3)

- Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1millisecond and that all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when:
  - The time quantum is 1 millisecond
  - The time quantum is 10 milliseconds
- Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate w; when it is running, its priority changes at a rate r. All processes are given a priority of 0 when they enter the ready queue. The parameters w and r can be set to give many different scheduling algorithms.
  - What is the algorithm that results from $r > w > 0$?
  - What is the algorithm that results from $w < r < 0$?

For contemplation (4)

- Explain the differences in the degree to which the following scheduling algorithms discriminate in favour of short processes:
  - FCFS
  - RR
  - Multilevel feedback queues
- The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: The higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

  Priority = (recent CPU usage / 2) + base

  where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated. Assume that recent CPU usage for process P1 is 40, process P2 is 18, and process P3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?