

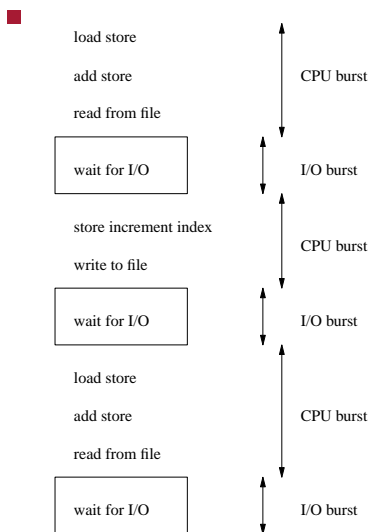
# Process scheduling

- See Silberschatz Chapter 5

# Process boundness

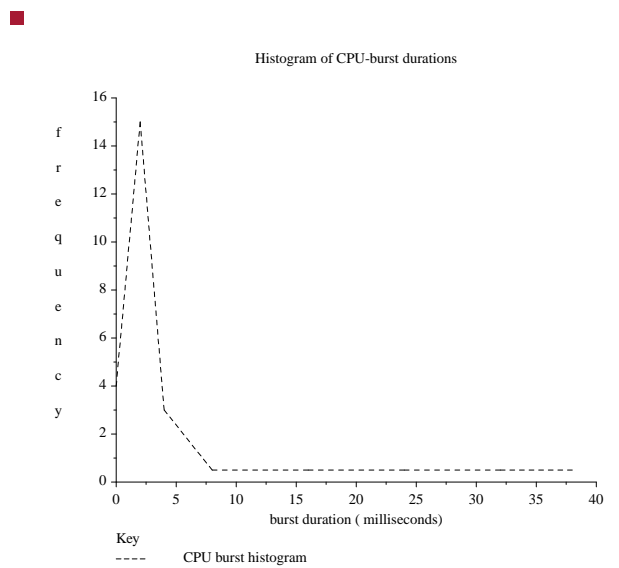
- Processes can be described as either:
  - I/O-bound process - spends more time doing I/O than computations, many short CPU bursts
  - CPU-bound process - spends more time doing computations; fewer but long CPU bursts

# Alternating Sequence of CPU and I/O Bursts



Silberschatz Fig 5.1

# Histogram of CPU-burst Times



Silberschatz Fig 5.2

## Preemptive vs non-preemptive

- Preemptive scheduling occurs when
  - CPU scheduler removes a process from the running state to allow another process to run.
- Non-preemptive scheduling means that
  - process runs to completion or performs an I/O wait or relinquishes control itself.

## Scheduling Criteria

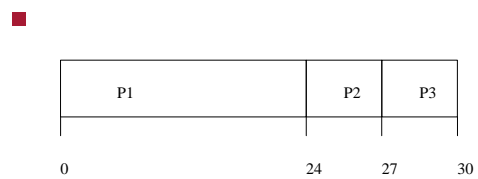
- define terms
- Waiting time = time spent in ready queue
- Turnaround time = time when process has completed

## First-Come, First-Served (FCFS) Scheduling

- | Process | CPU Burst Time |
|---------|----------------|
| P1      | 24             |
| P2      | 3              |
| P3      | 3              |

- All processes arrive at time 0 but in the order P1, P2, P3

## First-Come, First-Served (FCFS) Scheduling



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

## First-Come, First-Served (FCFS) Scheduling

## First-Come, First-Served (FCFS) Scheduling

- Suppose that the processes arrive in the order P2, P3, P1

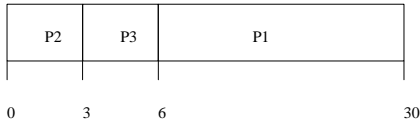
- Waiting time for P1 = 6; P2 = 0; P3 = 3

- The time chart for the schedule is:

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

- 

- much better than previous case.



- Convoy effect long process after short process

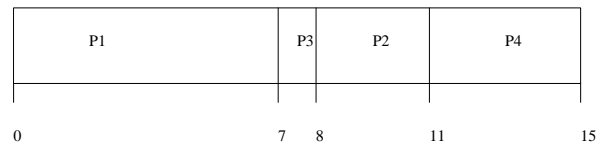
## Shortest-Job-First (SJF) Scheduling

## Example of Non-Preemptive SJF

- Allocate CPU to process with smallest CPU burst time.
- Two schemes:
  - nonpreemptive - once CPU given to the process it cannot be preempted until completes its CPU burst.
  - preemptive - if a new process arrives with CPU burst length less than remaining time of current executing process, preempt
  - 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

Process	Arrival Time	CPU Burst Time
P1	0	7
P2	2	3
P3	4	1
P4	5	4

- 

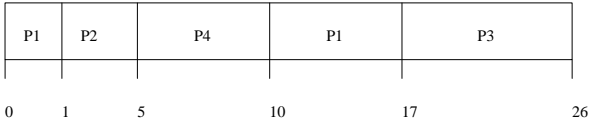


- Average waiting time =  $(0 + (8 - 2) + (7 - 4) + (11 - 5))/4 = 3.75$

## Example of Preemptive SJF



Process	Arrival Time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5



- Average waiting time =  $((10 - 1) + 0 + (17 - 2) + (5 - 3))/4 = 6.5$

## Priority Scheduling

- a priority number (integer) is associated with each process
- the CPU is allocated to the process with the highest priority (smallest integer is the highest priority)
  - Preemptive
  - nonpreemptive

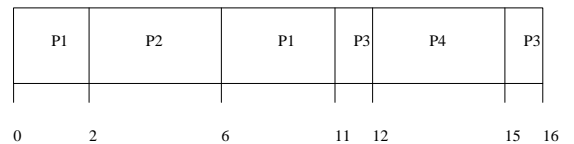
## Priority Scheduling

- SJF is a priority scheduling where priority is the 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 Preemptive Priority



Process	Arrival Time	CPU time	Priority
P1	0	7	2
P2	2	4	1
P3	4	2	4
P4	12	3	3



- Average waiting time =  $((6 - 2) + 0 + ((11 - 4) + (15 - 12)) + 0)/4 = 3.5$

## Round Robin (RR)

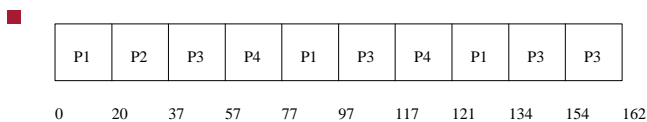
- Each process gets a unit of CPU time (time quantum), usually 10-200 milliseconds.
- After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are  $n$  processes in the ready queue and the time quantum is  $q$ , then each process gets  $\frac{1}{n}$  of the CPU time in chunks of at most  $q$  time units

## Round Robin (RR)

- No process waits more than  $(n - 1)q$  time units
- Performance
  - if we use a large quanta then round robin behaves as FCFS
  - if we use a too small quanta then the overhead is too high
    - thus it must be large with respect to context switch

### Example of RR with Time Quantum = 20

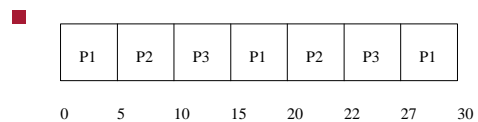
- | Process | CPU Burst Time |
|---------|----------------|
| P1      | 53             |
| P2      | 17             |
| P3      | 68             |
| P4      | 24             |



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

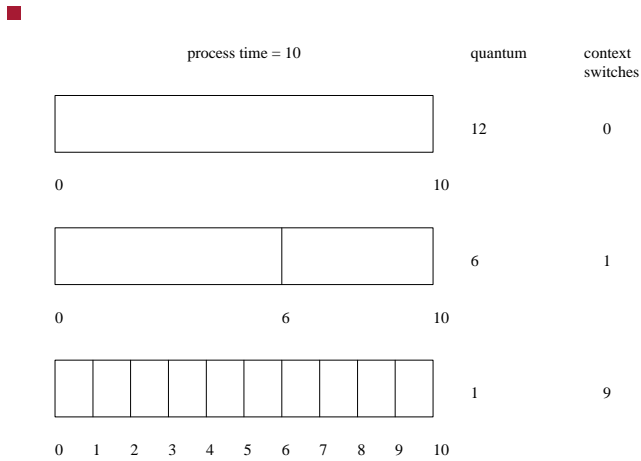
### Simpler example of RR with Time Quantum = 5

- | Process | CPU Burst Time |
|---------|----------------|
| P1      | 13             |
| P2      | 7              |
| P3      | 10             |



- Waiting time for P1  
=  $0 + (15 - 5) + (27 - 20) = 17$
- Waiting time for P2 =  $5 + (20 - 10) = 15$
- Waiting time for P3 =  $10 + (22 - 15) = 17$
- Average waiting time  
=  $(17 + 15 + 17) / 3 = 16.33$

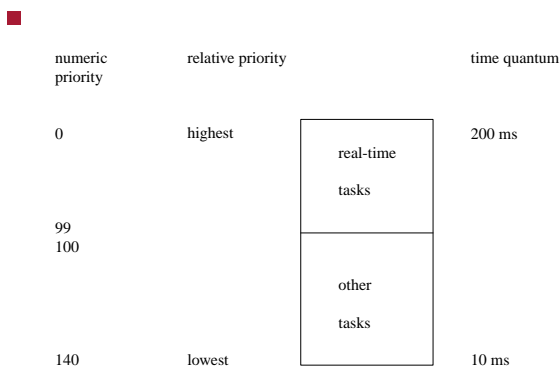
## Time Quantum and Context Switch Time



## Real-life scheduling

- Both Windows and Unix operating systems use scheduling algorithms which incorporate pre-emptive priority mechanisms
- The Unix command nice can be used to alter the priority of a process
- The Linux scheduler favours I/O bound over CPU bound processes
  - the Linux kernel also has the ability to change duration of quantum (time-slice)

## Priorities and time slice length in Linux



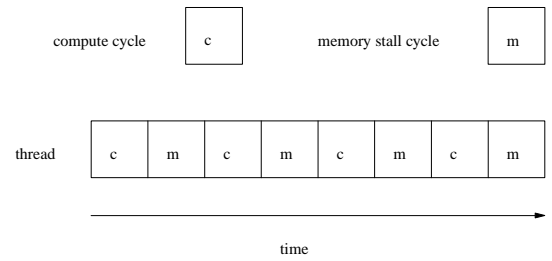
## Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- Homogeneous processors within a multiprocessor
- Asymmetric multiprocessing
  - only one processor accesses the system data structures, alleviating the need for data sharing
- Symmetric multiprocessing (SMP)
  - each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes

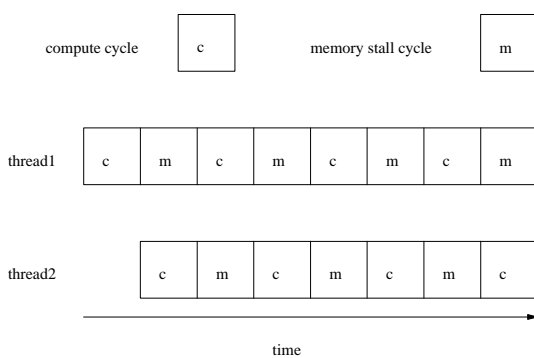
## Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consume less power
- Multiple threads per core
- Takes advantage of memory stall to make progress on another thread while memory retrieve happens cf. CPU and I/O bound process definition

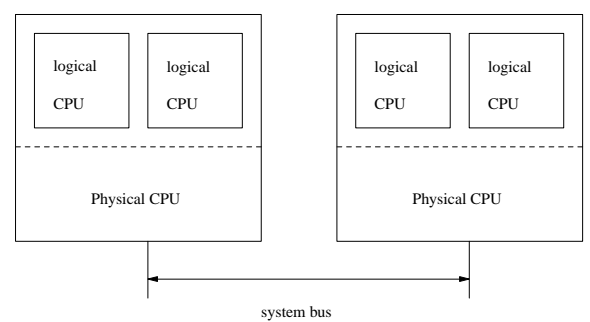
## Single thread activity



## Multithreaded multicore activity



## Thread scheduling on multicore processors



- Scheduling takes place between threads on logical CPUs - Silberschatz (pg 205, Ed 8) suggests algorithms such as round-robin and priority/urgency are employed at this level