Chapter 2

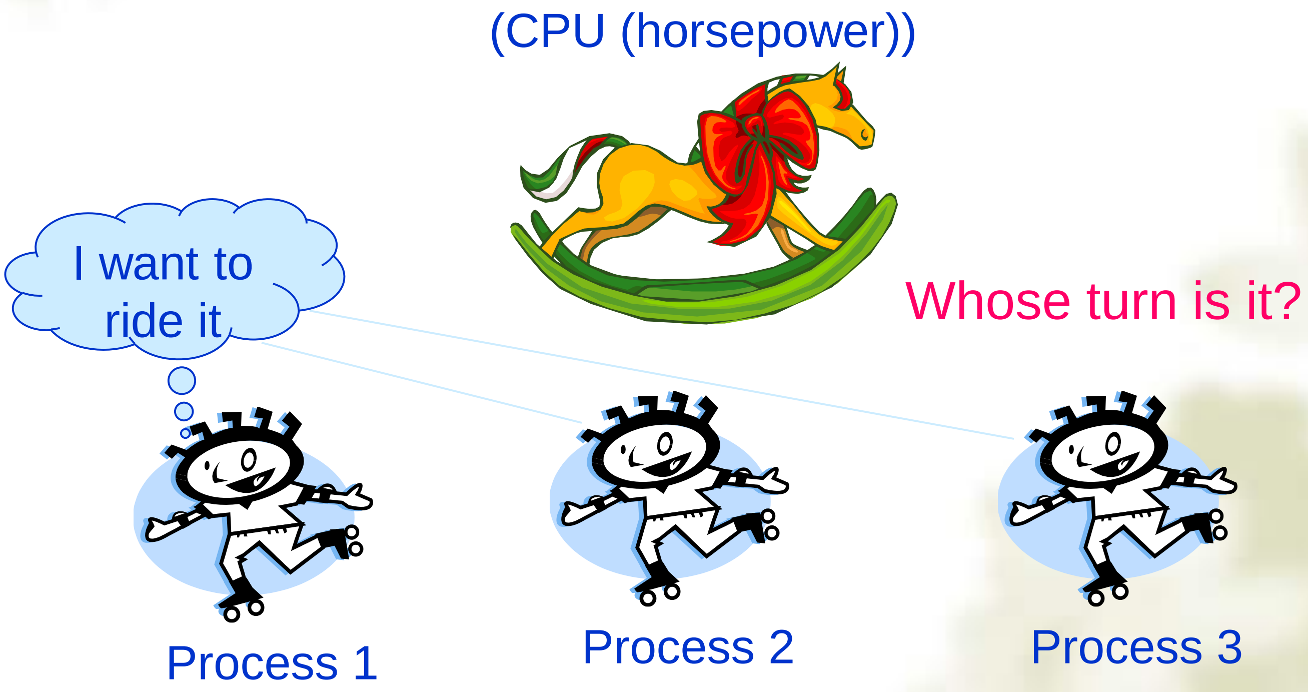
Processes and Threads

Content of this lecture

- What is scheduling?
- When to Schedule?
- Basic Scheduling Algorithm
  - Batch systems
    - First-Come First-Served
    - Shortest job first
  - Interactive systems
    - Round-robin
    - Priority scheduling
    - Multi Queue & Multi-level Feedback
    - Guaranteed Scheduling, Lottery Scheduling, and Fair Sharing Scheduling
- Summary

What is scheduling

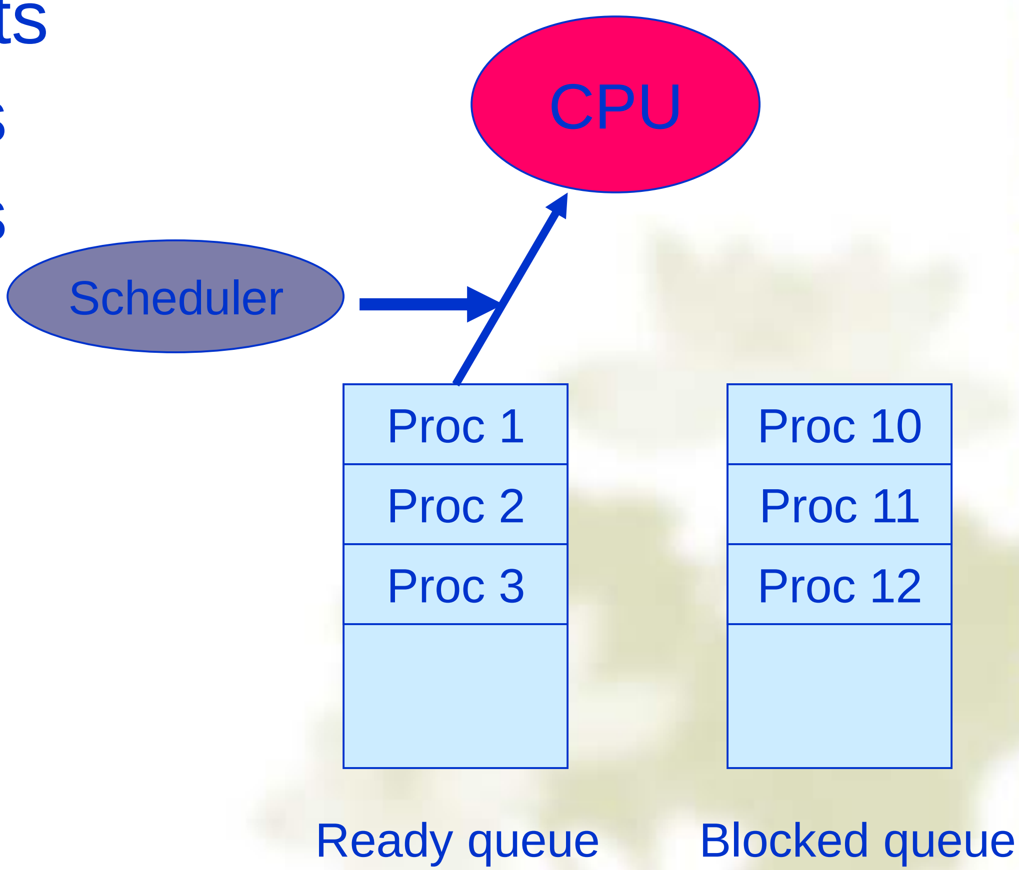
- Deciding which process/thread should occupy the resource (CPU) to run next.



Process Scheduler

- Proc1: 20 time units
- Proc2: 3 time units
- Proc3: 4 time units

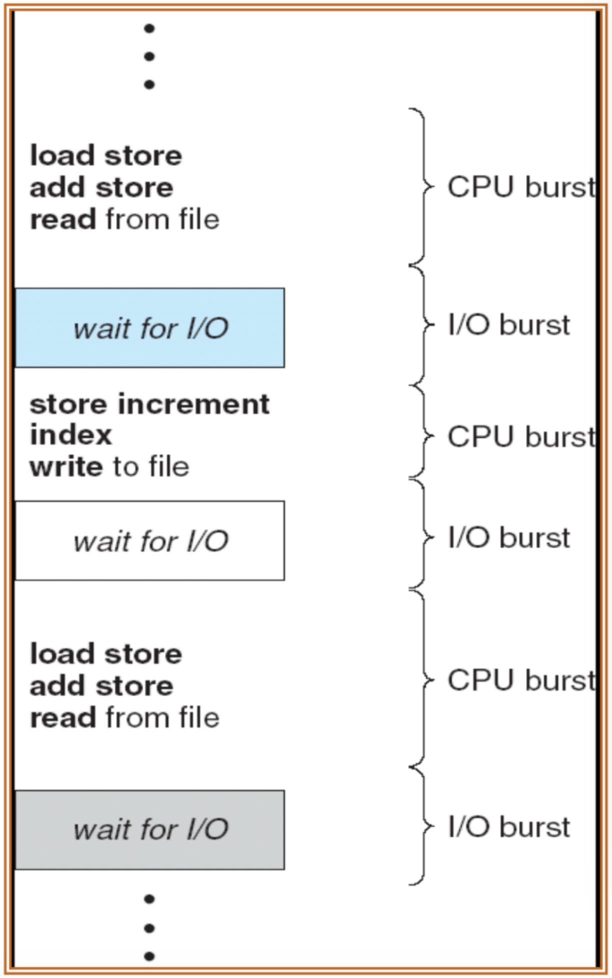
- Preemptive vs. non-preemptive



Process Behavior

- Processes typically consist of
  - CPU bursts
    - A period of time when a process needs the CPU is called a CPU burst.
  - I/O bursts
    - A period of time when a process needs I/O is called a I/O burst.

Process Behavior

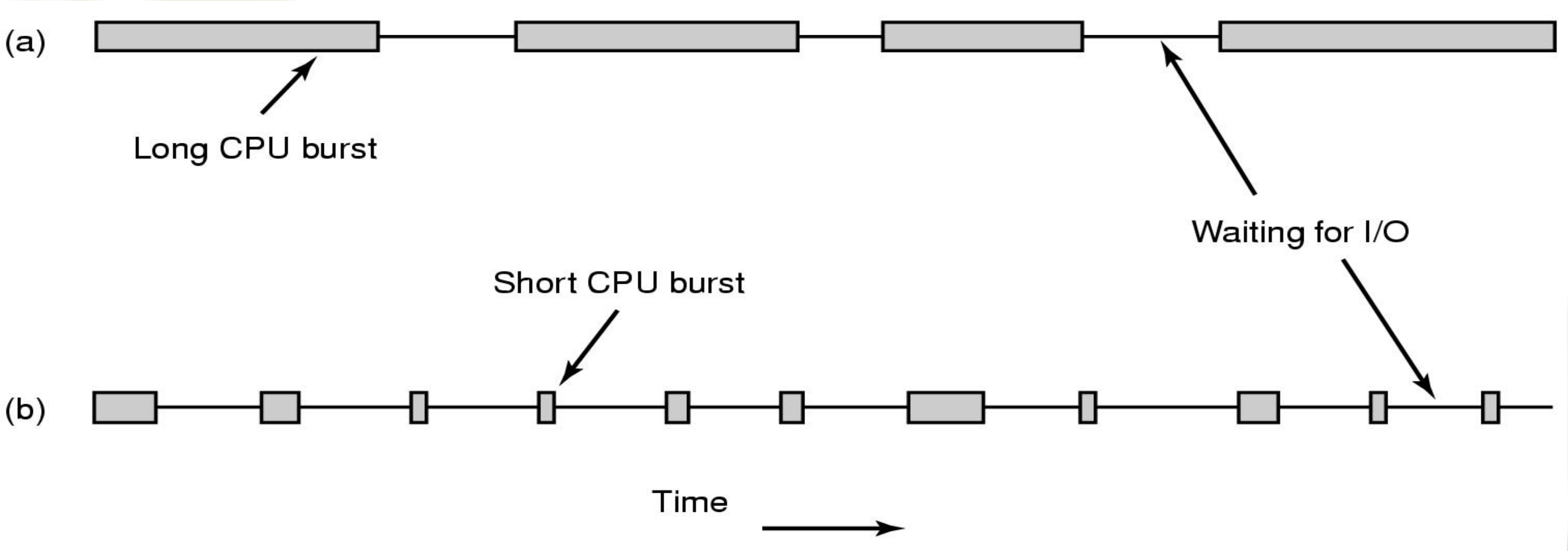


Alternating Sequence of CPU and I/O Bursts

Process Behavior

- Duration and frequency of bursts vary greatly from process to process.
  - CPU-bound Process
    - Long CPU bursts, infrequent I/O waits.
    - E.g. Number crunching tasks, image processing.
  - I/O-bound Process
    - Short CPU bursts, frequent I/O waits.
    - E.g. A task that processes data from disk, for example, counting the number of lines in a file is likely to be I/O bound.
- As CPU get faster, processes tend to get more I/O bound.

Process Behavior



- Bursts of CPU usage alternate with periods of I/O wait
  a) a CPU-bound process
  b) an I/O bound process

CPU Scheduling

- Problem to solve
  - When (scheduling opportunity)

    when to allocate CPU to process?
  - What (scheduling algorithm)

    what is the principle of allocation?
  - How (context - switch)

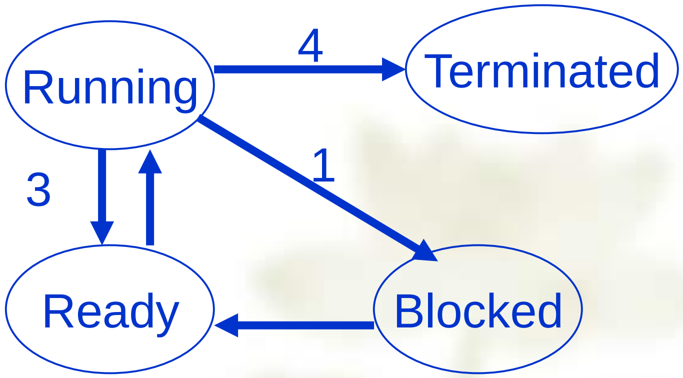
    How to allocate CPU? scheduling- process?

When to schedule?

- A new process is created
- The running process exits
- The running process is blocked
- I/O interrupt (some processes will be ready)
- Clock interrupt (every 10 milliseconds)

Preemptive vs. Non-preemptive

- Non-preemptive scheduling
  - The running process keeps the CPU until it voluntarily gives up the CPU
    - process exits
    - switches to blocked state
    - 1and 4 only (no 3)
  - Disadvantage?



- Preemptive scheduling
  - The running process can be interrupted and must release the CPU (can be forced to give up CPU)
  - Preemptive principles?

Categories of Scheduling Algorithms

- Batch System
  - Non-preemptive algorithms
  - Preemptive algorithms with long time periods for each process
- Interactive System
  - Preemption is essential
- Real-Time System
  - Preemption is sometimes not needed

Scheduling Algorithm

Properties of a GOOD Scheduling Algorithm:

- Fair (nobody cries)
- Priority (lady first)
- Efficiency (make best use of equipment)
- Encourage good behavior (good boy/girl)
- Support heavy loads (degrade gracefully)
- Adapt to different environments (interactive, real-time…)

Performance Criteria

- Different Systems, Different Focuses
- For All Systems
  - Fairness
    - No process should suffer starvation
  - Efficiency
    - Keep resources as busy as possible
  - Policy Enforcement
    - Seeing that stated policy is carried out

Performance Criteria

- Batch Systems
  - Throughput
    - Number of processes that completes in unit time
  - Turnaround Time (also called elapse time)
    - Amount of time to execute a particular process from the time its entered
  - Waiting Time
    - amount of time process has been waiting in ready queue
  - Processor Utilization
    - Percent of time CPU is busy

Performance Criteria

- Interactive Systems
  - Response Time
    - amount of time from when a request was first submitted until first response is produced.
  - Proportionality
    - meet users' expectation
- Real-Time Systems
  - Meeting Deadlines
    - avoid losing data
  - Predictability
    - Same time/cost regardless of load on the system

Single Processor Scheduling Algorithms

- Batch systems
  - First Come First Served (FCFS)
  - Shortest Job First
- Interactive Systems
  - Round Robin
  - Priority Scheduling
  - Multi Queue & Multi-level Feedback
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair Sharing Scheduling

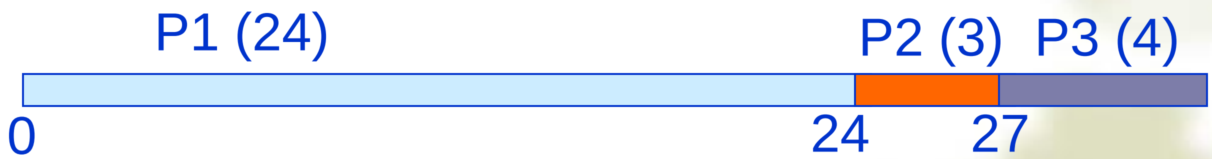
(1) First Come First Served (FCFS)

- Also called FIFO. Process that requests the CPU FIRST is allocated the CPU FIRST.
- Non-preemptive
- Used in Batch Systems
- Implementation: FIFO queues
  - A new process enters the tail of the queue
  - The scheduler selects from the head of the queue.
- Performance Metric: **Average Waiting Time**.
- Given Parameters:
  - Burst Time (in ms), Arrival Time and Order

FCFS Example

Process	Burst Time	Order	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

The final schedule:



P1 waiting time: 0
P2 waiting time: 24
P3 waiting time: 27

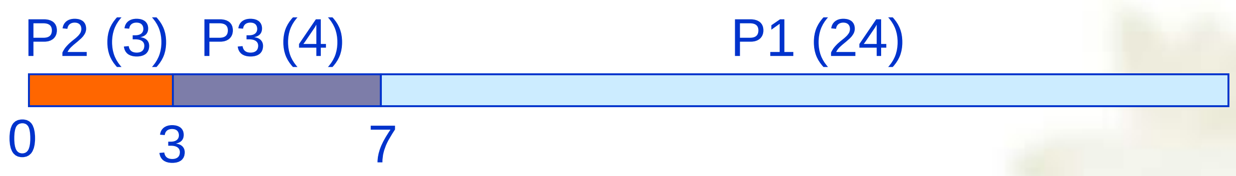
The average waiting time:
(0+24+27)/3 = 17

FCFS Example (cont.)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



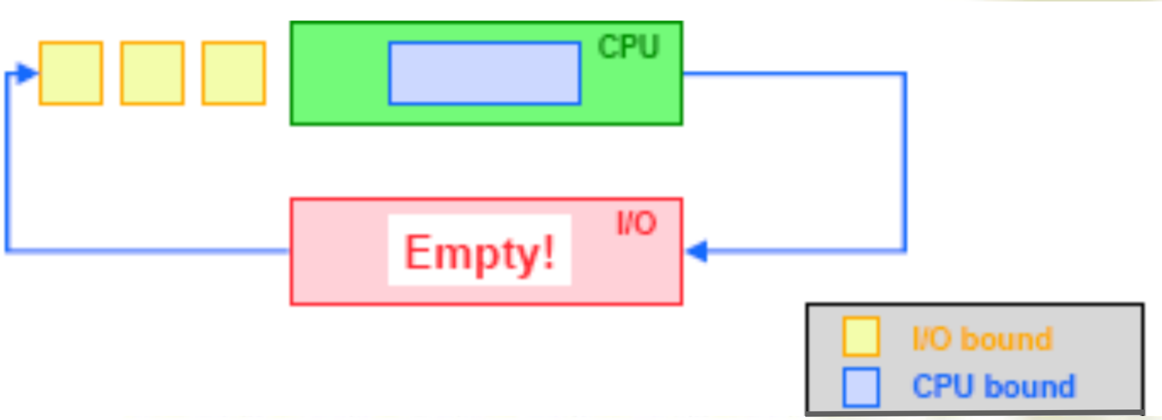
- Waiting time for $P_1 = 7; P_2 = 0; P_3 = 3$
- Average waiting time: $(7 + 0 + 3)/3 = 3.3$
- Much better than previous case.
- *Convoy effect*: short process behind long process

Problems with FCFS

- Non-preemptive
- Not optimal AWT (Average Waiting Time)
- Cannot utilize resources in parallel:
  - Assume 1 process CPU bounded and many I/O bounded processes
  - result: Convoy effect, low CPU and I/O Device utilization
  - Why?

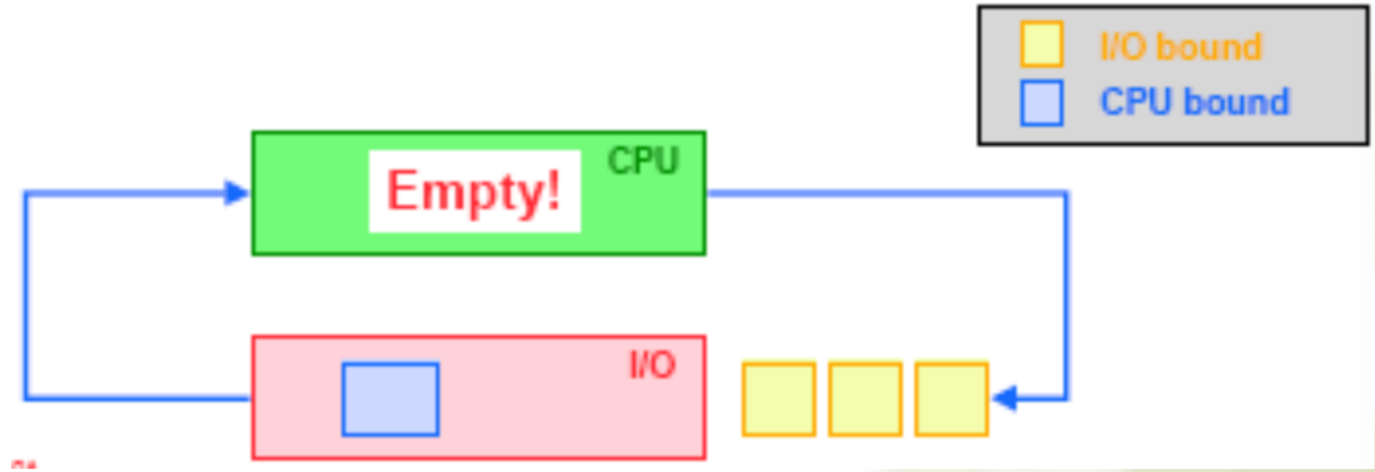
Why Convoy Effects?

- Consider *n-1* jobs in system that are I/O bound and 1 job that is CPU bound.
- I/O bound jobs pass quickly through the ready queue and suspend themselves waiting for I/O.
- CPU bound job arrives at head of queue and executes until complete.
- I/O bound jobs rejoin ready queue and wait for CPU bound job to complete.
- I/O devices idle until CPU bound job completes.



Why Convoy Effects?

- When CPU bound job complete, other processes rush to wait on I/O again.
- CPU becomes idle.



(2) Shortest Job First (SJF)

- Schedule the job with the shortest elapse time first
- Scheduling in Batch Systems
- Two types:
  - Non-preemptive
  - Preemptive
- Requirement: the elapse time needs to know in advance
- Optimal if all the jobs are available simultaneously (provable)
  - Gives the best possible AWT (average waiting time)

Non-preemptive SJF: Example

Process	Burst Time	Order	Arrival Time
P1	6	2	0
P2	8	4	0
P3	7	3	0
P4	3	1	0

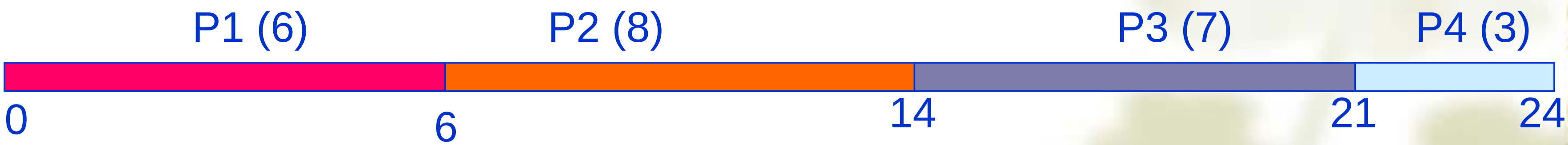
P4 waiting time: 0
P1 waiting time: 3
P3 waiting time: 9
P2 waiting time: 16

The total time is: 24

The average waiting time (AWT):
$(0+3+9+16)/4 = 7$

Comparing to FCFS

Process	Burst Time	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0



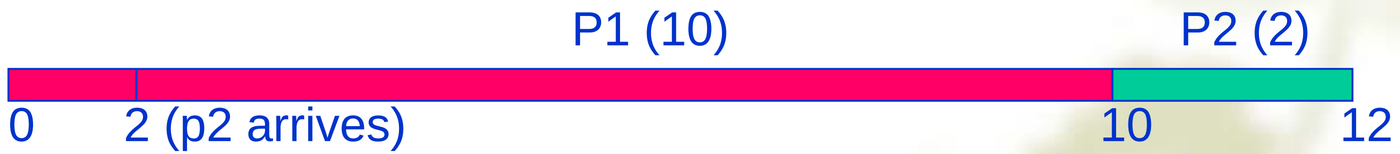
P1 waiting time: 0
P2 waiting time: 6
P3 waiting time: 14
P4 waiting time: 21

The total time is the same.
The average waiting time (AWT):
(0+6+14+21)/4 = 10.25
(comparing to 7)

SJF is not always optimal

- Is SJF optimal if all the jobs are not available simultaneously?

Process	Burst Time	Order	Arrival Time
P1	10	1	0
P2	2	2	2

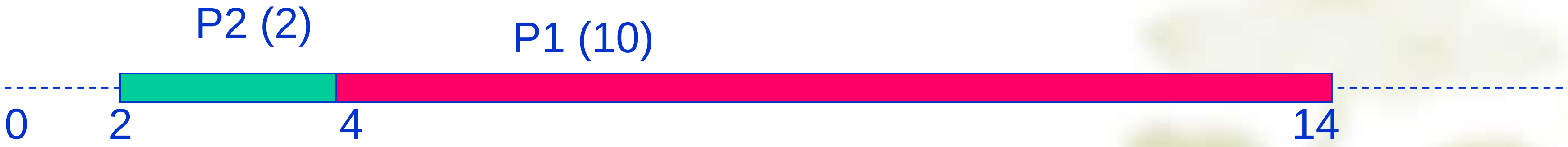


P1 waiting time: 0
P2 waiting time: 8

The average waiting time (AWT):
$(0+8)/2 = 4$

What if the scheduler waits for 2 time units? (Do it yourself)

Process	Burst Time	Order	Arrival Time
P1	10	2	0
P2	2	1	2



P1 waiting time: 4
P2 waiting time: 0

The average waiting time (AWT):
(0+4)/2 = 2

However: waste 2 time units of CPU

Preemptive SJF

- Also called Shortest Remaining Time First
  - Schedule the job with the shortest remaining time required to complete
- Requirement: the elapse time needs to be known in advance

Preemptive SJF: Same Example

Process	Burst Time	Order	Arrival Time
P1	10	1	0
P2	2	2	2

P1 waiting time: 4-2 =2
P2 waiting time: 0

The average waiting time (AWT):
(0+2)/2 = 1

No CPU waste!!!

A Problem with SJF

- Starvation
  - In some condition, a job is waiting for ever
  - Example: SJF
    - Process *A* with elapse time of 1 hour arrives at time 0
    - But ever 1 minute from time 0, a short process with elapse time of 2 minutes arrive
    - Result of SJF: *A* never gets to run

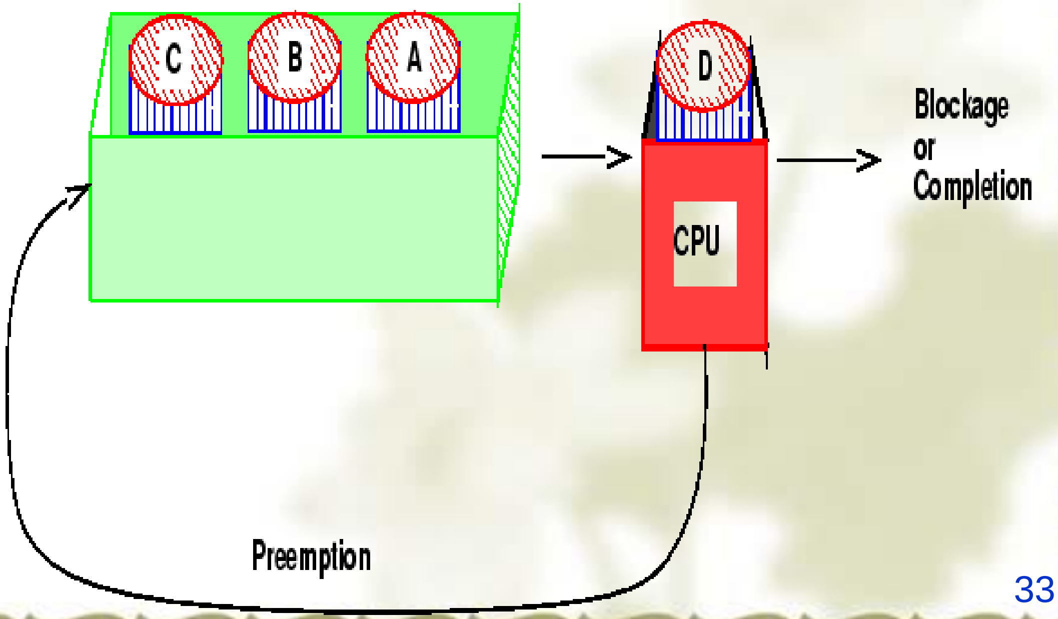
Interactive Scheduling Algorithms

- Usually preemptive
  - Time is sliced into quantum (time intervals)
  - Scheduling decision is also made at the beginning of each quantum
- Performance Criteria
  - Min Response Time
  - best proportionality
- Representative algorithms:
  - Round-robin
  - Priority-based
  - Multi Queue & Multi-level Feedback
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair Sharing Scheduling

(3) Round-robin

- One of the oldest, simplest, most commonly used scheduling algorithm
- Select process/thread from ready queue in a round-robin fashion (take turns)

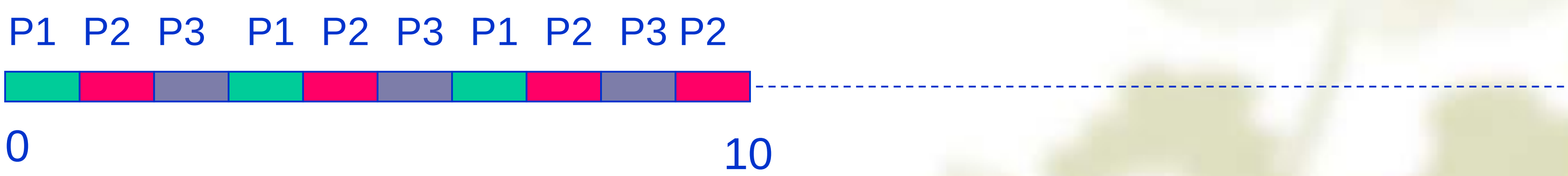
Problem:
- Do not consider priority
- Context switch overhead



Round-robin: Example

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is: 1 unit, P1, P2 & P3 never block



P1 waiting time: 4
P2 waiting time: 6
P3 waiting time: 6

The average waiting time (AWT):
(4+6+6)/3 = 5.33

Time Quantum

- Time slice too large
  - FCFS behavior
  - Poor response time
- Time slice too small
  - Too many context switches (overheads)
  - Inefficient CPU utilization
- Heuristic: 70-80% of jobs block within time-slice
- Typical time-slice 10 to 100 ms

(4) Priority Scheduling

- Each job is assigned a priority.
- FCFS within each priority level.
- Select highest priority job over lower ones.
- Rational: higher priority jobs are more mission-critical
  - Example: display a video film in real time vs. send email
- Problems:
  - May not give the best AWT
  - indefinite blocking or starving a process

Set Priority

- Two approaches
  - Static (for system with well known and regular application behaviors)
  - Dynamic (otherwise)
- Priority may be based on:
  - Cost to user
  - Importance of user
  - Process type
  - Requirement to resource
  - Aging
  - Percentage of CPU time used in last X hours.

Priority Scheduling: Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

P2 waiting time: 0
P4 waiting time: 8
P3 waiting time: 11
P1 waiting time: 18

The average waiting time (AWT):
(0+8+11+18)/4 = 9.25
(worse than SJF:7)

Priority in Unix

```
yyzhou|csil-linux1|~|[73]% ps -l
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
100 S 14828  2047  2045  0  79   4    -   822 rt_sig pts/1    00:00:00 csh
000 R 14828 23001  2047  0  80   4    -   791 -      pts/1    00:00:00 ps
yyzhou|csil-linux1|~|[74]%
yyzhou|csil-linux1|~|[74]%
```

Nobody wants to

Be “nice” in Unix

```
NICE(1)                          FSF                          NICE(1)

NAME
       nice - run a program with modified scheduling priority

SYNOPSIS
       nice [OPTION] [COMMAND [ARG]...]

DESCRIPTION
       Run COMMAND with an adjusted scheduling priority.  With no
       COMMAND, print the current scheduling priority.  ADJUST is
       10  by default.  Range goes from -20 (highest priority) to
       19 (lowest).

       -ADJUST
              increment priority by ADJUST first

       -n, --adjustment=ADJUST
              same as -ADJUST

       --help display this help and exit

       --version
lines 1-25
```

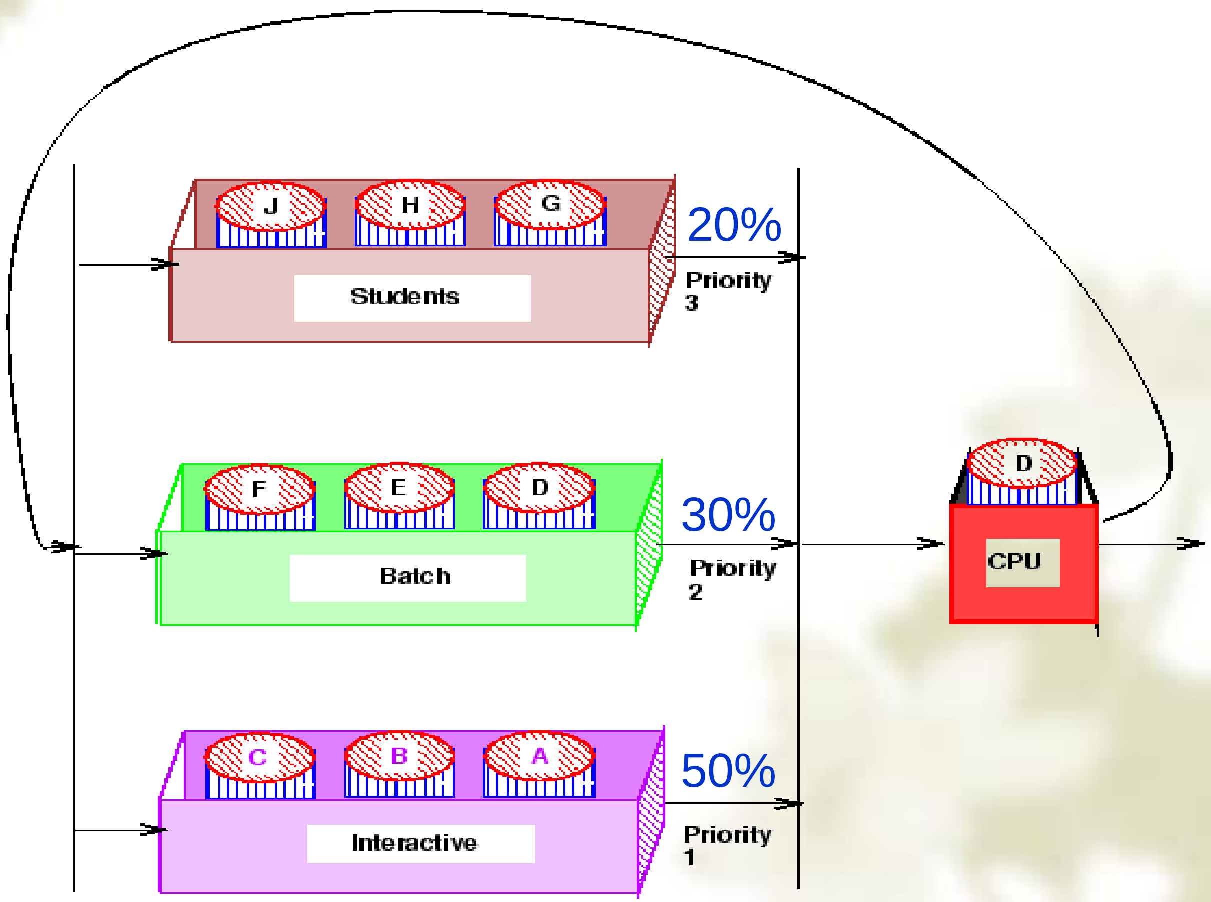
(5) Multi-Queue Scheduling

- Hybrid between priority and round-robin
- Used in scenarios where the processes can be classified into groups based on property like process type, CPU time, IO access, memory size, etc.
- Split the Ready Queue in several queues, processes assigned to one queue permanently
- Each queue with its own scheduling algorithm
  - E.g. interactive processes: RR, background processes: FCFS/SRTF

Multi-Queue Scheduling

- Example
  - System processes (highest priority)
  - Interactive programs (Round Robin)
  - Background Processes (FCFS)
  - Student Processes
- Scheduling between queues
  - Fixed Priorities (Possibility of starvation)
  - % CPU spent on queue

Multi-Queue Scheduling: Example



Real Life Analogy

- Tasks (to-do list) for poor *Bob*
  - Class 1 priority (highest): tasks given by his boss
    - Finish the project (50%)
  - Class 2 priority: tasks for his wife
    - Buy a valentine present (30%)
  - Class 3 priority (lowest): Bob's tasks
    - Watch TV (20%)

(6)A Variation: Multi-level Feedback Algorithm

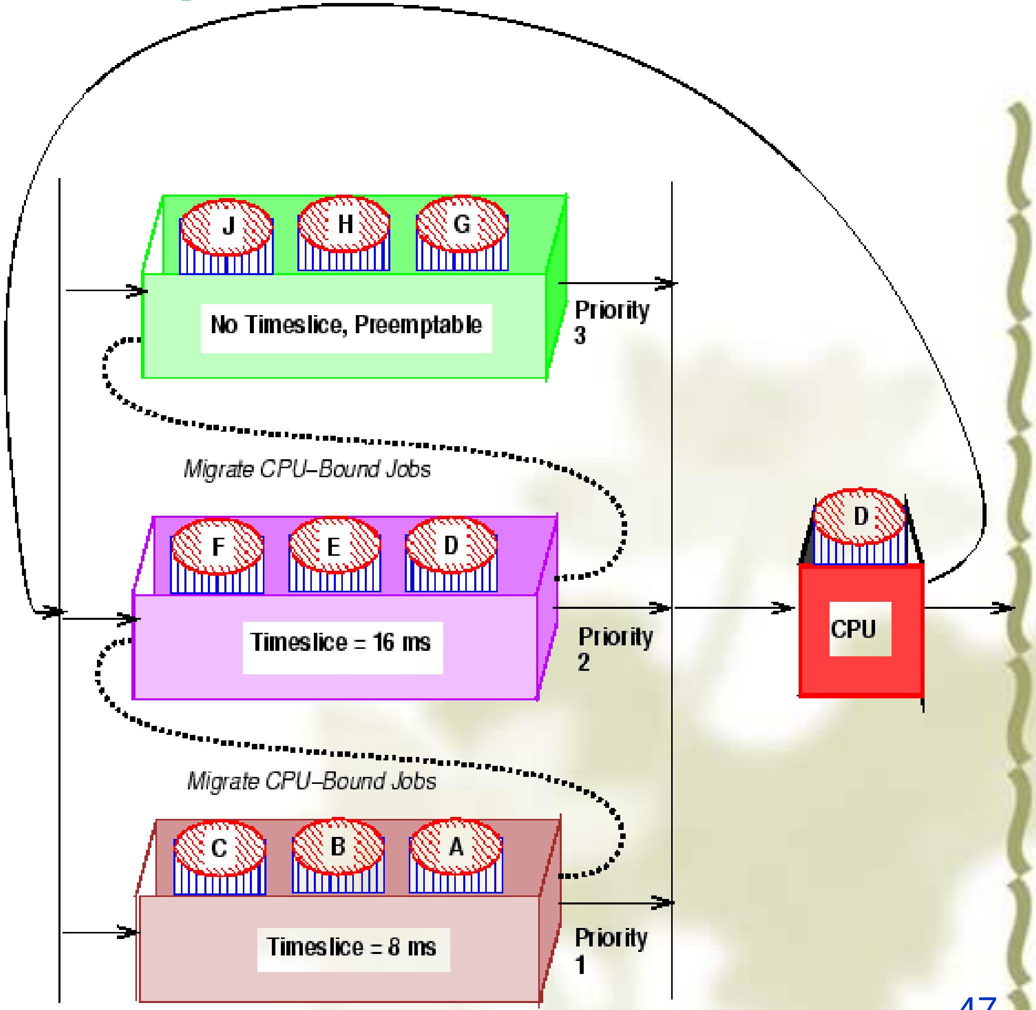
- Multi-Level Queue with priorities
- Processes **move** between queues
  - Start each process in a high-priority queue; as it finishes each CPU burst, move it to a lower- priority queue.
- Each queue represents jobs with similar CPU usage
- Jobs in a given queue are executed with a given time-slice
- Rational:
  - Once an I/O process completes an I/O request, it should have higher CPU priority.

Multi-level Feedback Algorithm (Details)

- Example (CTSS): *$Queue_i$* has time-slice $t\ 2^i$
  - If a job in *$Queue_i$* doesn't block by end of time-slice, it is moved to *$Queue_{i+1}$*
  - Lowest priority Queue is FCFS

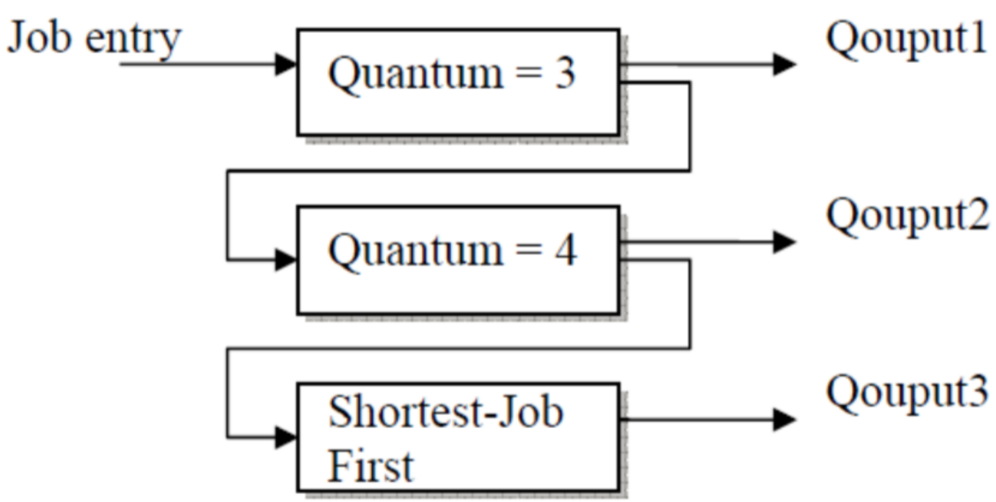
Multi-level Feedback Algorithm: Example

- Three Queues
  - Q_0: time slice 8ms
  - Q_1: time slice 16ms
  - Q_2: FCFS



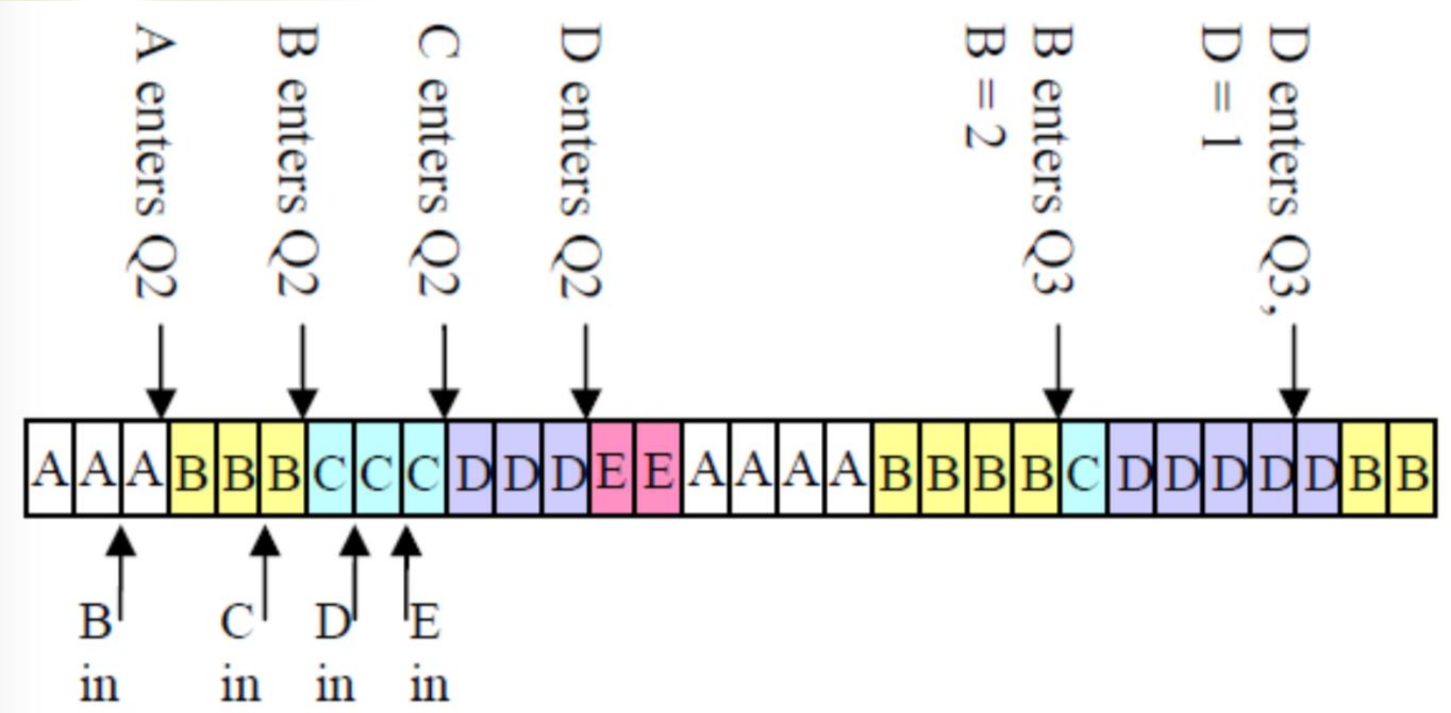
Quiz

Show your schedule with timeline and calculate the average turnaround time when use multi-level feedback queue as below (Please take arrival time into account). Note that the priority of the top 2 queues is based on arrival times.



Process ID	Arrival Time	Burst Time
A	0	7
B	2	9
C	5	4
D	7	8
E	8	2

Solution



Turnaround time:

A=18, B=30-2=28, C=23-5=18, D=28-7=21, E=14-8=6

Average turnaround time:

(18 + 28 + 18 + 21 + 6)/5 = 91/5 = 18.2

Review: Scheduling Algorithms

- Batch systems
  - First come first served
  - Shortest job first
- Interactive systems
  - Round-robin
  - Priority scheduling
  - Multi-Queue & Multi-level feedback

(7) Guaranteed Scheduling (QoS)

- Make real promises to the users about performance and then live up to them.
- Example:
  - with *n* processes running, the scheduler makes sure that each one gets 1/*n* of the CPU cycles.
- Scheduling:
  - compute the ratio of actual CPU time consumed to CPU time entitled
  - Select the one with the lowest ratio

(8) Lottery Scheduling

- More commonly used
- Probability-based
  - Give processes lottery tickets. At scheduling time, a lottery ticket is chosen at random, and the process holding that ticket gets that resource.
- Give more tickets for higher priority processes (approximate Priority), or give short jobs more tickets (approximate SJF)
- Advantages
  - Simple
  - Highly responsive
  - Can support cooperation between processes
  - Easy to support priority and proportion requirement

(9) Fair-Share Scheduling

- Is Round-robin fair?
  - Yes, it is (from process point of view)
  - No, it may be not (from user point view)
- User-based fair share scheduling
  - Each user gets fair share
- Example:
  - Alice has 4 processes: A1, A2, A3, A4
  - Bob has 1 process: B1
  - Then A1, A2, A3, A4 are entitled only to 50% CPU, while B1 alone is entitled to 50%
  - Possible scheduling sequence: A1,B1,A2,B1,A3,B1,A4,B1,……

Scheduling in Real-Time Systems

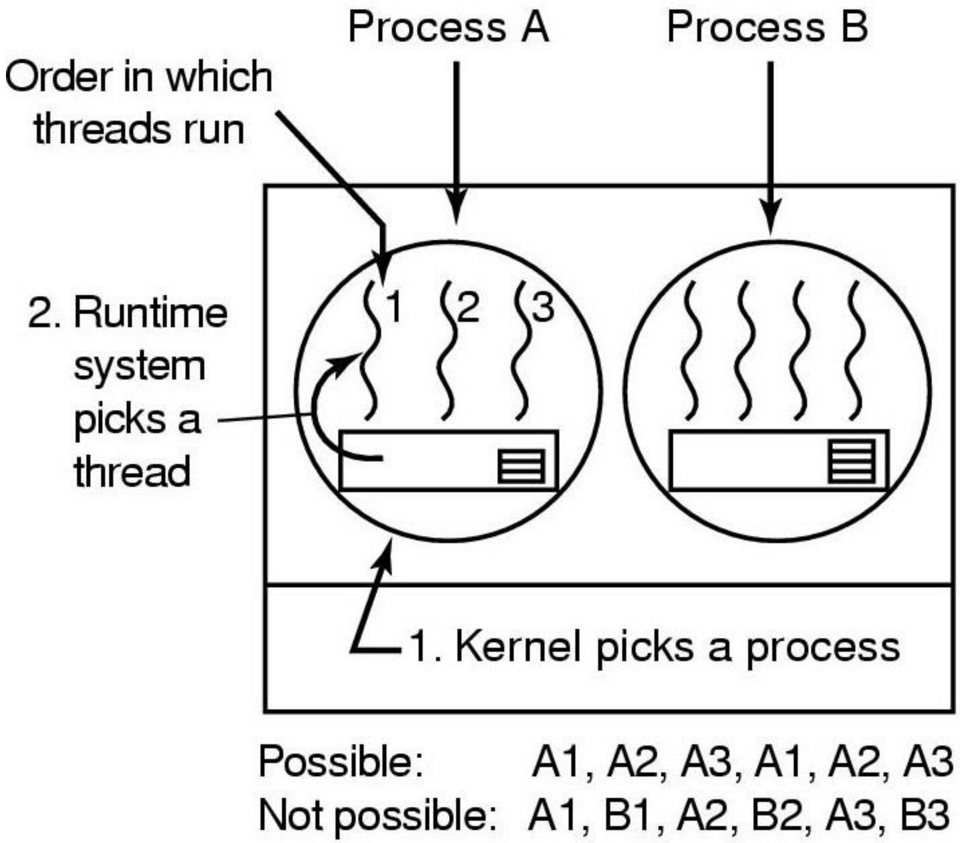
- The scheduler makes real promises to the user in terms of deadlines or CPU utilization.
- Schedulable real-time system
  - Given
    - *m* periodic events
    - event *i* occurs within period P_i and requires C_i seconds
  - Then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

User-level Thread Scheduling

Possible Scheduling

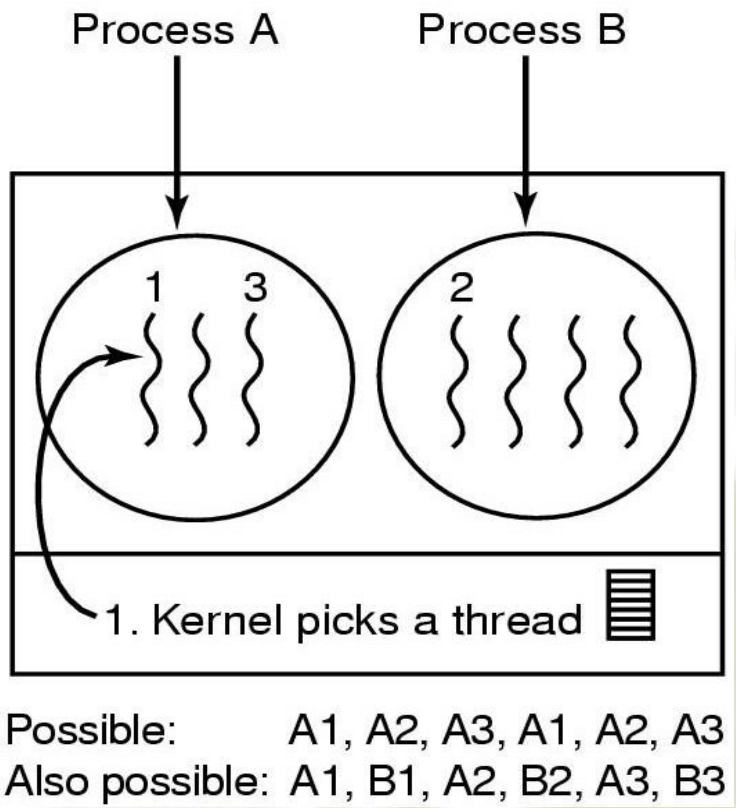
- 50-msec quantum
- run 5 msec/CPU burst



Kernel-level Thread Scheduling

Possible scheduling

- 50-msec quantum
- threads run 5 msec/CPU burst



Summary

- What is scheduling
- Scheduling objectives
- CPU Scheduling
- FCFS
- Shortest job first (SJF)
- Priority
- Round-robin
- Multi-Queue
- Multi-level Feedback

- Scheduling algorithms
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair Sharing Scheduling
- Scheduling for
  - Real-time systems
  - Threads

Homework

- 45、50
- Reading assignment: 10.3