Scheduling

Scheduling

- When more than one process is ready to run, but only one CPU is available, a choice is to make
- Part of OS that does it is scheduler
- The algorithm it uses is scheduling algorithm

Importance of Scheduling

- Good scheduling algorithms can make a big difference
  - Resource utilization
  - Perceived performance & User satisfaction
  - Meeting other system goals (e.g., important tasks being taken care of immediately)

When to Schedule

- When a new process is created:
  - Parent or child? Both are Ready
  - which one to run?
- When a process exits:
  - One of the ready processes should be run
- When a process blocks: Another process has to be selected to run
  - Blocking may occur for:
    - I/O
    - Semaphore

When to Schedule

- When an I/O interrupt occurs:
  - In case of an interrupt of an I/O device having completed its work, some blocked process may now be ready
- If a h/w clock provides periodic interrupt: A scheduling decision can be made at each (or kth) clock interrupt

Preemptive & Non-preemptive

Classification of Scheduling Algorithm depending on dealing with clock interrupt

- Non-preemptive: Picks a process to run and lets it run until it **blocks** or voluntarily releases the CPU. In effect at each clock interrupt, no scheduling is done.
- Preemptive: Picks a process and lets it run for a maximum of some fixed time. If still running, it is suspended and another is picked.
- Preemptive scheduling requires having a clock interrupt occur at the end of the time interval to give control of the CPU back to the scheduler

Batch Systems

- Common performance metrics
  - Throughput: number of jobs completed per hour
  - Turnaround time: average time between the submission and completion of a job
- Maximizing Throughput may not necessarily minimize Turnaround time

First Come First Serve (FCFS)

- Process that requests the CPU FIRST is allocated the CPU FIRST.
- Also called FIFO
- non-preemptive
- Used in Batch Systems
- Real life analogy?
  - Transaction at Sonali Bank
- Implementation
  - FIFO queues
  - A new process enters the tail of the queue
  - The schedule selects from the head of the queue.

FCFS Example

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

The final schedule:

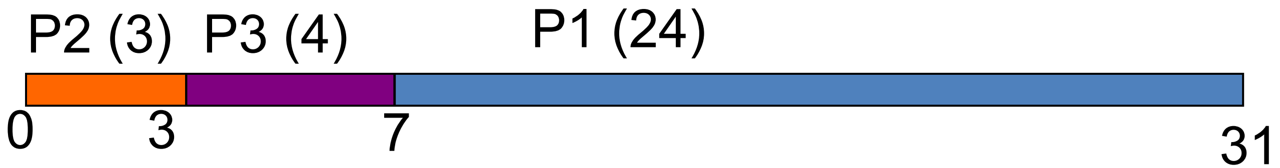
P1 turnaround: 24
P2 turnaround: 27
P3 turnaround: 31

The average turnaround:
$(24+27+31)/3 = 27.33$

FCFS Example 2

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

The final schedule:



P1 turnaround: 31
P2 turnaround: 3
P3 turnaround: 7

The average turnaround:
$(31+3+7)/3 = 13.67$

Advantage

- Easy to understand and implement
- Fair for equivalent processes

Problems with FCFS

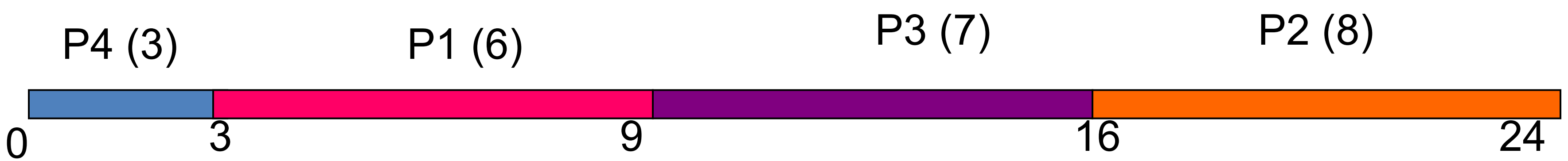
- Non-preemptive
- Non optimal turnaround
- Cannot utilize resources in parallel:
  - Assume 1 process CPU bounded and many I/O bounded processes
  - result: **Convoy effect**,
    - low CPU and I/O Device utilization
  - Why?

Shortest Job First (SJF)

- Scheduling algorithm in batch systems
- Schedule the job with the shortest run time first
- Requirement: the run time needs to be known in **advance**
- SJF is optimal in terms of turnaround, if **all** jobs arrive at **same time**

SJF: Example

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



Do it yourself

P4 turnaround: 3
P1 turnaround: 9
P3 turnaround: 16
P2 turnaround: 24

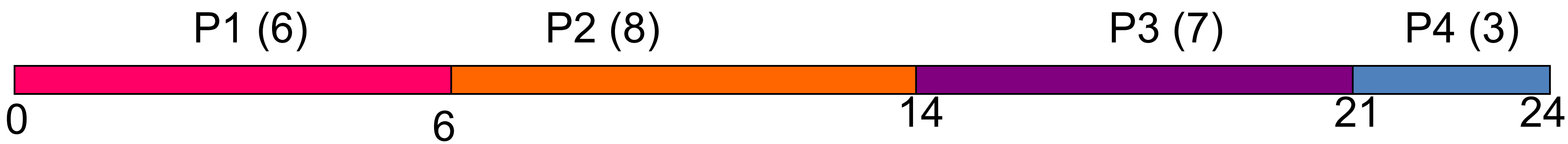
Total execution time: 24

The average turnaround:

$(3+9+16+24)/4 = 13$

Comparing to FCFS

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



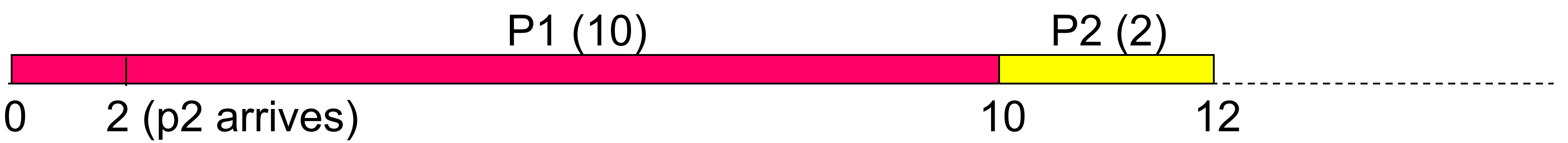
P1 turnaround: 6
P2 turnaround: 14
P3 turnaround: 21
P4 turnaround: 24

The total time is the same.
The average turnaround:
(6+14+21+24)/4 = 16.25
(comparing to 13)

SJF is not always optimal

- SJF optimal only if all jobs have arrived at scheduling time

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



P1 turnaround: 10
P2 turnaround: 10

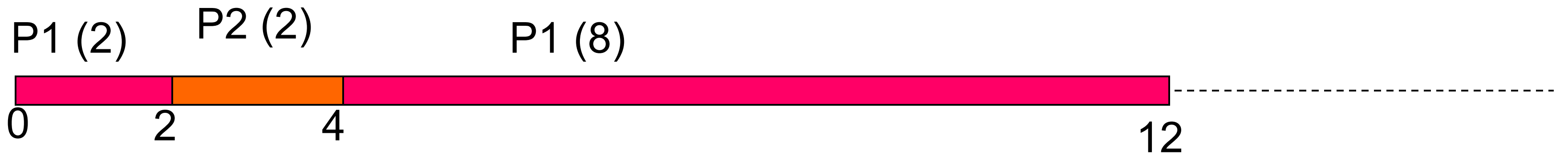
The average turnaround (AWT): (10+10)/2 = 10

Preemptive SJF

- Also called Shortest Remaining Time Next
  - Schedule the job with the shortest remaining time required to complete
  - When new job arrives, compare its total time with the remaining time of the running job
  - If the new job needs less time the current job is suspended and the new job started
- Requirement: the run time needs to be known in advance

Preemptive SJF: Same Example

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



P1 turnaround: 12
P2 turnaround: 2

The average turnaround:
$(2+12)/2 = 7$

Problem with Preemptive SJF?

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

Interactive System

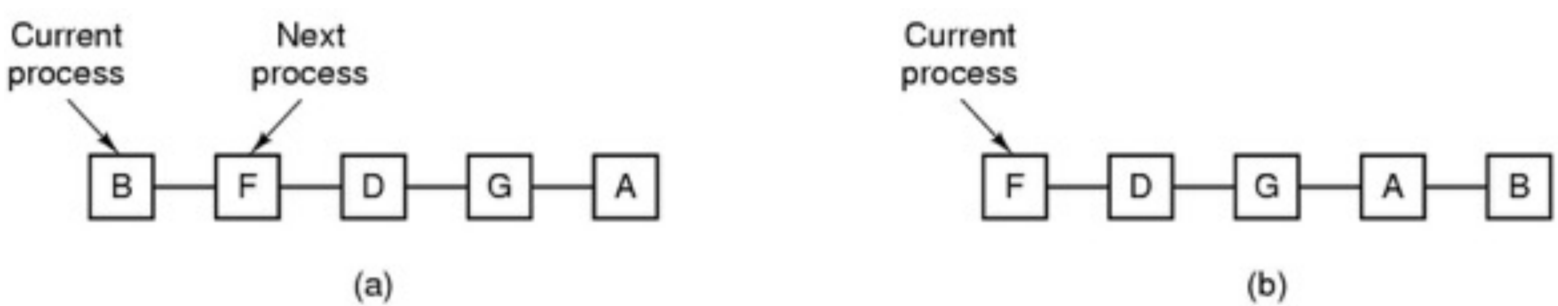
- Example: Servers
  - Serve multiple remote users all of whom are in a big hurry
- Performance Criteria
  - Min response time:
    - amount of time it takes from when a request was submitted until the first response is produced, not output
    - respond to requests quickly

Interactive System

- Algorithms used here usually preemptive
  - Time is **sliced** into quantum (time intervals)
  - Scheduling decision is also made at the beginning of each quantum
- Representative algorithms:
  - Round-robin
  - Priority-based
  - Shortest process time
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair Sharing Scheduling

Round Robin

- **Round Robin (RR)**
  - Often used for timesharing
  - Each process is given a time slice called a ***quantum***
  - It is run for the quantum or until it blocks
  - RR allocates the CPU uniformly (fairly) across participants from ready queue.
- Problem:
  - Do not consider priority
  - Context switch overhead



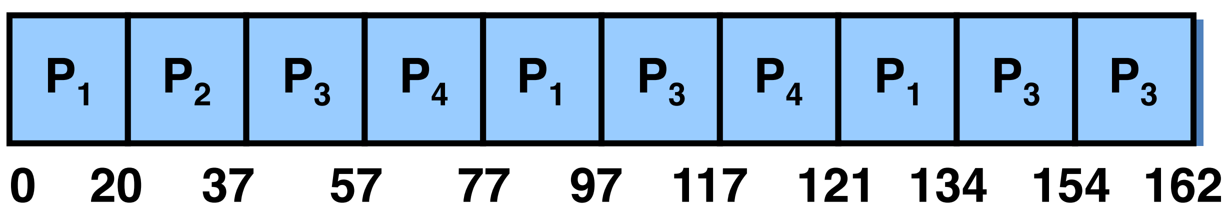
Implementing Round Robin

- Keep the ready queue as a FIFO queue of processes.
- New processes are added to the tail of the ready queue.
- The scheduler
  - picks the first process from the ready queue
  - sets a timer to interrupt after 1 time quantum, and
  - Starts the process.
- When the quantum is over
  - The running process will be put at the **tail** of the ready queue.

RR with Time Quantum = 20

Process	Run Time
P_1	53
P_2	17
P_3	68
P_4	24

- All processes arrive at time 0
- The Gantt chart is



- Higher average turnaround than SJF
- But better response time

RR: Choice of Time Quantum

- Performance depends on length of the timeslice
  - Context switching isn’t a free operation.
  - If timeslice time is set too high
    - attempting to amortize context switch cost, you get FCFS.
      - i.e. processes will finish or block before their slice is up anyway
      - Poor response time
  - If it’s set too low
    - you’re spending all of your time context switching between threads.

Priority Scheduling

- Each job is assigned a priority
- Select highest priority job to run next
- Rational: higher priority jobs are more important
  - Example: simulation vs. auto save a document
- Problems:
  - Low priority process may starve
- Solution:
  - Priority need to be **adjusted** depending on the situation

Assign 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
  - Percentage of CPU time used in last X hours

Example: Dynamic Priority Assignment

- Whenever highly I/O bound processes wants the CPU it should be given the CPU immediately.
- Why?
- A simple algorithm for giving priority to I/O bound processes is to set the priority to $1/f$
  - f is the fraction of the last quantum used by a process
  - A process that used only 1 msec of its 50 msec quantum would get priority 50
  - A process that used 25 msec of its 50 msec quantum would get priority 2