CS 153
Design of Operating Systems

Winter 2016

Lecture 11: Scheduling

Scheduling Overview

- Scheduler runs when we context switching among processes/threads on the ready queue
  - What should it do? Does it matter?
- Making this decision is called scheduling
- Now, we'll look at:
  - The goals of scheduling
  - Starvation
  - Various well-known scheduling algorithms
  - Standard Unix scheduling algorithm

Multiprogramming

- In a multiprogramming system, we try to increase CPU utilization and job throughput by overlapping I/O and CPU activities
  - Doing this requires a combination of mechanisms and policy
- We have covered the mechanisms
  - Context switching, how and when it happens
  - Process queues and process states
- Now we'll look at the policies
  - Which process (thread) to run, for how long, etc.
- We'll refer to schedulable entities as jobs (standard usage) – could be processes, threads, people, etc.

Scheduling Goals

- Scheduling works at two levels in an operating system
  1. To determine the multiprogramming level – the number of jobs loaded into primary memory
     » Moving jobs to/from memory is often called swapping
     » Long term scheduler: infrequent

  2. To decide what job to run next to guarantee "good service"
     » Good service could be one of many different criteria
     » Short term scheduler: frequent
     » We are concerned with this level of scheduling
     » Is scheduler a thread always running in kernel space? (Use your PintOS experience)

Scheduling

- The scheduler (aka dispatcher) is the module that manipulates the queues, moving jobs to and from them

- The scheduling algorithm determines which jobs are chosen to run next and what queues they wait on

- In general, the scheduler runs, when PintOS calls next_thread_to_run:
  - When a job switches from running to waiting
  - When an interrupt occurs
  - When a job is created or terminated

- The scheduler runs inside the kernel. Therefore, kernel has to be entered before scheduler can run.

Preemptive vs. Non-preemptive scheduling

- We’ll discuss scheduling algorithms in two contexts
  - In preemptive systems the scheduler can interrupt a running job (involuntary context switch)
  - In non-preemptive systems, the scheduler waits for a running job to explicitly block (voluntary context switch)

Scheduling Goals

- What are some reasonable goals for a scheduler?
- Scheduling algorithms can have many different goals:
  - CPU utilization
  - Job throughput (# jobs/unit time)
  - Turnaround time ($T_{finish} - T_{start}$)
  - Waiting time ($Avg(T_{wait})$: avg time spent on wait queues)
  - Response time ($Avg(T_{ready})$: avg time spent on ready queue)
- Batch systems
  - Strive for job throughput, turnaround time (supercomputers)
- Interactive systems
  - Strive to minimize response time for interactive jobs (PC)

Starvation

Starvation is a scheduling "non-goal":

- Starvation is a situation where a process is prevented from making progress because some other process has the resource it requires
  - Resource could be the CPU, or a lock (recall readers/writers)
- Starvation usually a side effect of the sched. algorithm
  - A high priority process always prevents a low priority process from running on the CPU
  - One thread always beats another when acquiring a lock
- Starvation can be a side effect of synchronization
  - Constant supply of readers always blocks out writers

First In First Out (FIFO)

- Schedule tasks in the order they arrive
  - Continue running them until they complete or give up the processor
- Example: many cases in real life

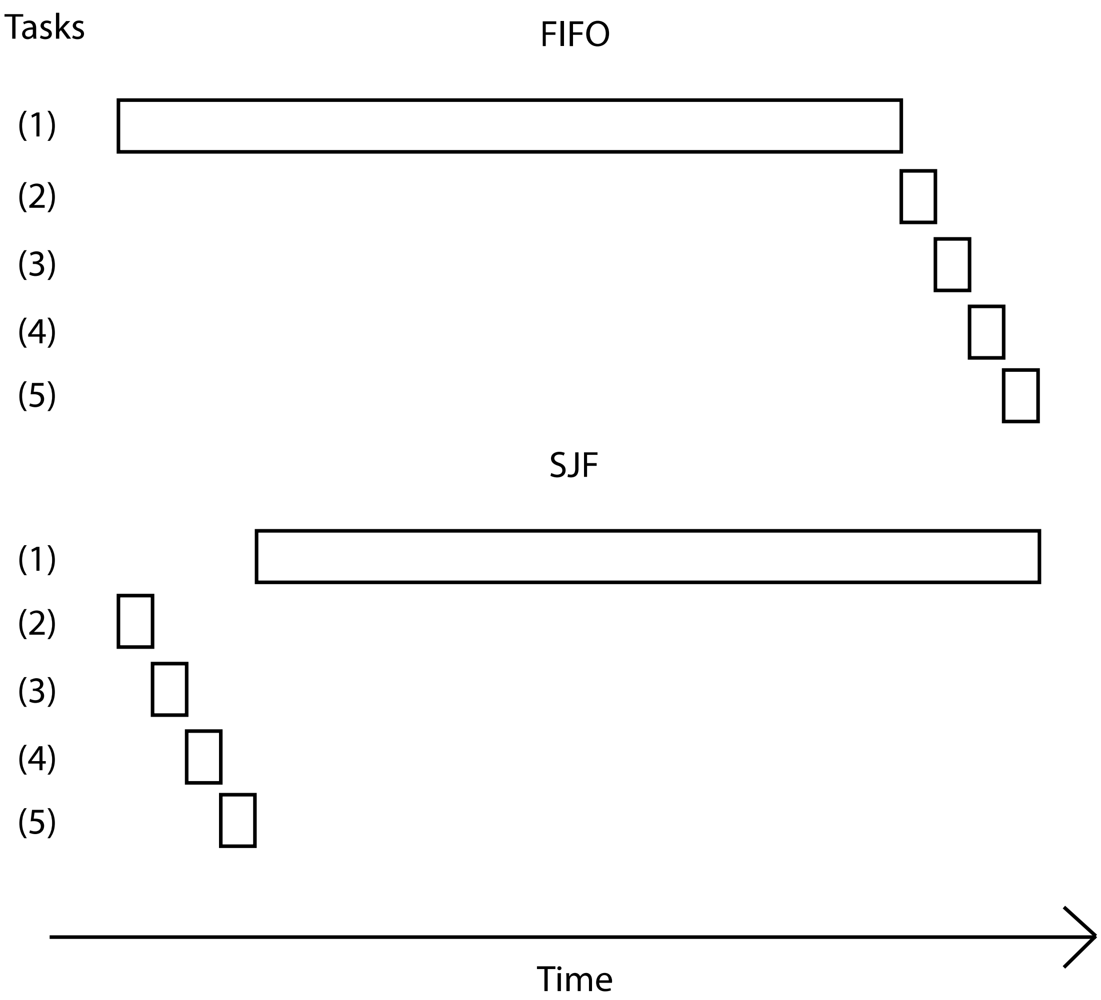
- On what workloads is FIFO particularly bad?
  - Imagine being at supermarket to buy a drink of water, but get stuck behind someone with a huge cart (or two!)
    - …and who pays in pennies!
  - Can we do better?

Shortest Job First (SJF)

- Always do the task that has the shortest remaining amount of work to do
  - Often called Shortest Remaining Time First (SRTF)

- Suppose we have five tasks arrive one right after each other, but the first one is much longer than the others
  - Which completes first in FIFO? Next?
  - Which completes first in SJF? Next?

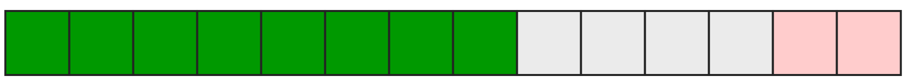
FIFO vs. SJF



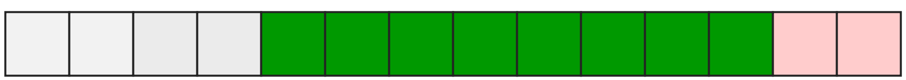
Whats the big deal? Don't they finish at the same time?

Average Turnaround Time (ATT)

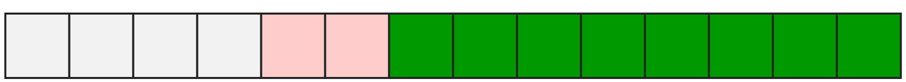
FIFO:



$ATT = (8 + (8+4)+(8+4+2))/3 = 11.33$

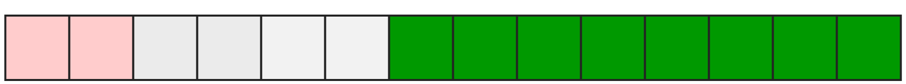


$ATT = (4 + (4+8)+(4+8+2))/3 = 10$



$ATT = (4+ (4+2)+(4+2+8))/3 = 8$

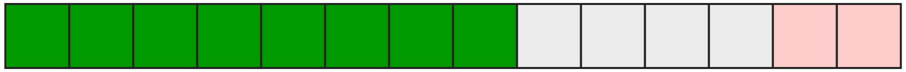
SJF:



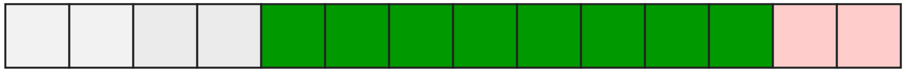
$ATT = (2 + (2+4)+(2+4+8))/3 = 7.33$

Average Response Time (ART)

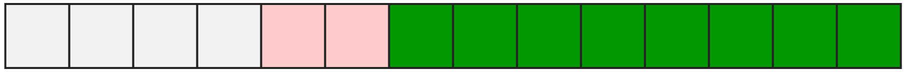
FIFO:



$ART = (0 + 8 + (8+4))/3 = 6.67$

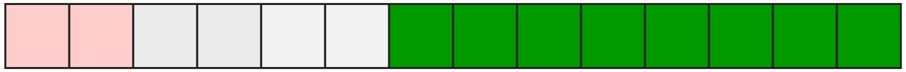


$ART = (0 + 4 + (4+8))/3 = 5.33$



$ART = (0 + 4 + (4+2))/3 = 3.33$

SJF:



$ART = (0 + 2 + (2+4))/3 = 2.67$

SJF

- Claim: SJF is optimal for average response time
  - Why?

- For what workloads is FIFO optimal?
  - For what is it pessimal (i.e., worst)?

- Does SJF have any downsides?
  - Does it work in a supermarket?

Shortest Job First (SJF)

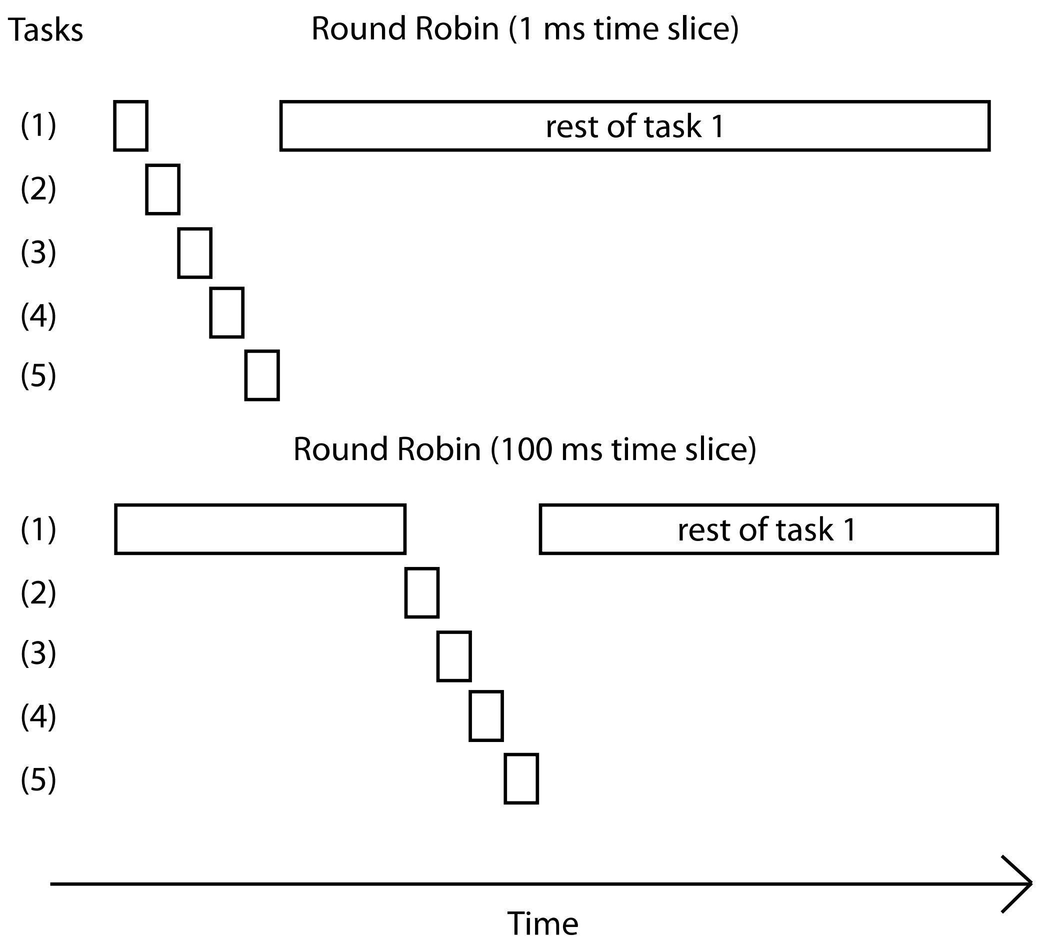
- Problems?
  - Impossible to know size of CPU burst
    - » Like choosing person in line without looking inside basket/cart
  - How can you make a reasonable guess?
  - Can potentially starve

- Flavors
  - Can be either preemptive or non-preemptive
  - Preemptive SJF is called shortest remaining time first (SRTF)

Round Robin

- Each task gets resource for a fixed period of time (time quantum)
  - If task doesn't complete, it goes back in line
- Need to pick a time quantum
  - What if time quantum is too long?
    - » Infinite?
  - What if time quantum is too short?
    - » One instruction?

Round Robin



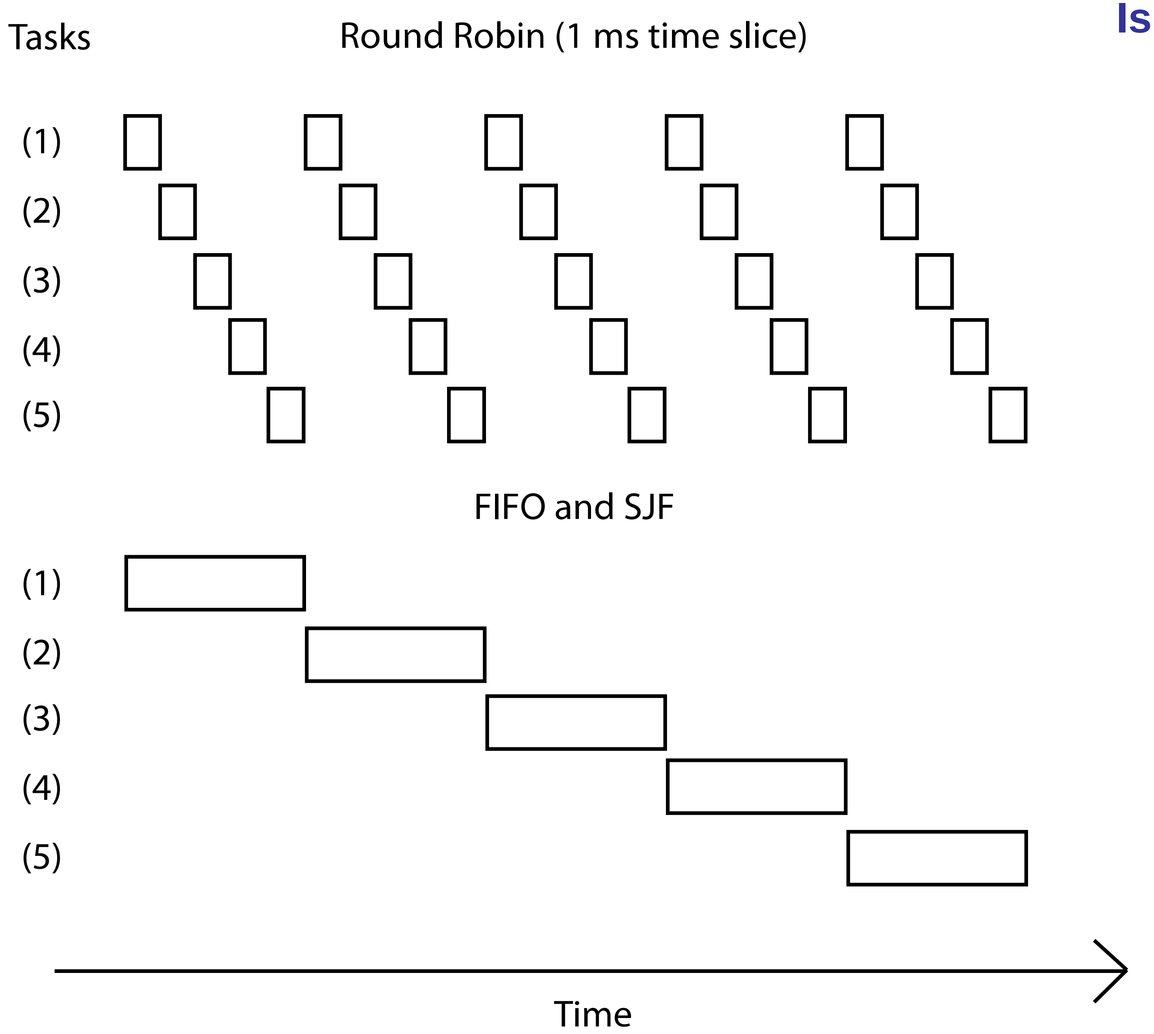
Tasks
Round Robin (1 ms time slice)
(1)
(2)
(3)
(4)
(5)
rest of task 1
Round Robin (100 ms time slice)
(1)
(2)
(3)
(4)
(5)
rest of task 1
Time

Round Robin vs. FIFO

- Many context switches can be costly
- Other than that, is Round Robin always better than FIFO, in terms of average response time or average turnaround time?

Round Robin vs. FIFO

Is Round Robin always fair?



Next Class

- Deadlock continued