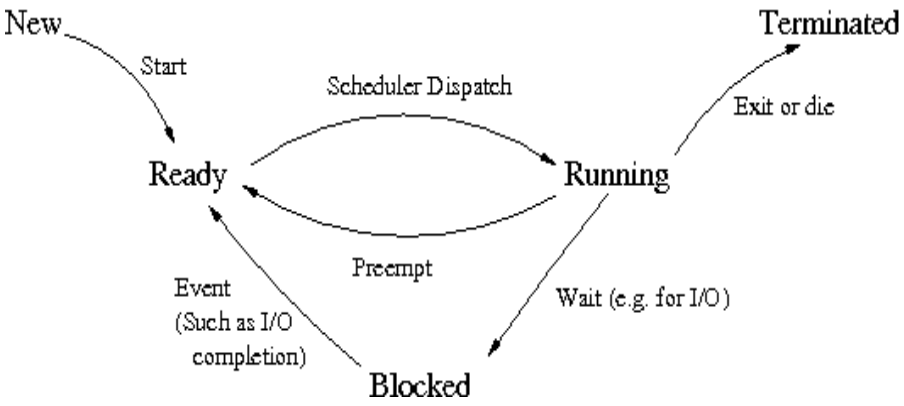
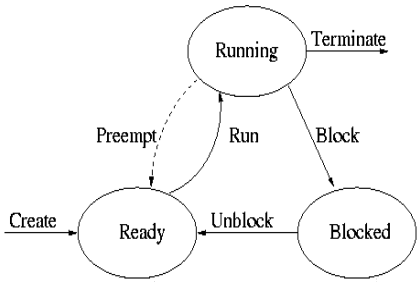
CSC 2405: Computer Systems II

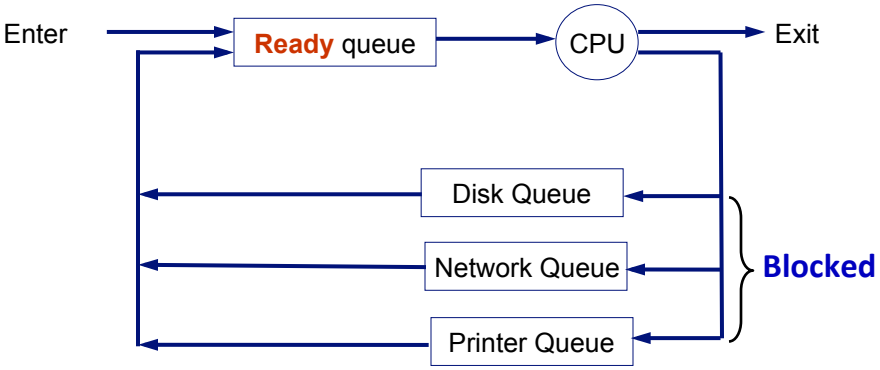
CPU Scheduling

Review: Process States &Transitions



Process Queues





CPU Scheduling

- Each process in the pool of Ready processes is ready to run

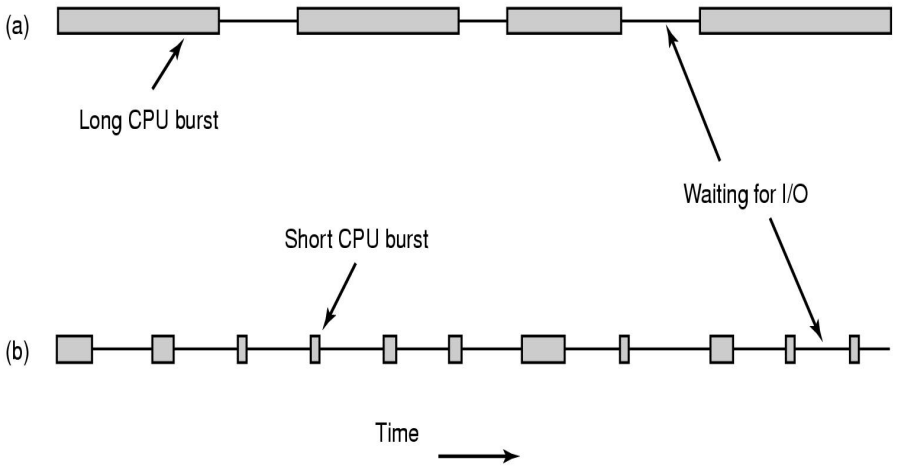
Which one to pick to run next?

- CPU scheduling
  - Selecting a new process to run from the Ready queue

Types of Scheduling

- *Preemptive* scheduling
  - Running process may be interrupted and moved to the Ready queue
- *Non-preemptive* scheduling:
  - once a process is running, it continues to execute until it terminates or blocks for I/O

Process Behavior: I/O and CPU Bursts



- Scheduler makes decisions:
  - When timer interrupt occurs (during a long CPU burst)
  - When I/O is requested or an I/O interrupt occurs

Scheduling Metrics

- How do we compare different scheduling policies?

- Simplifying assumptions:
  - Single processor (CPU)
  - Performance metric: *turnaround time*, equal to the difference between completion time and arrival time

- Other performance metrics include fairness and response time for interactive processes

Scheduling Algorithms

FCFS or FIFO

SJF / STCF

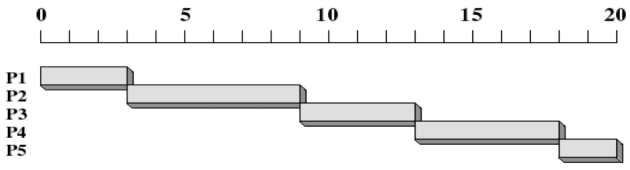
RR

First Come First Served (FCFS)

- Also known as FIFO (First In First Out)
- Simplest scheduling algorithm:
  - Run jobs in order that they arrive
  - Scheduling mode: non-preemptive
  - In cases of I/O requests, place job at back of Ready queue when I/O completes

FCFS Example

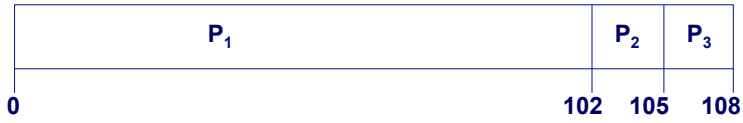
Process	Arrival Time	CPU Burst Time
P_1	0	3
P_2	2	6
P_3	4	4
P_4	6	5
P_5	8	2



Why FCFS is Not That Great

Process	CPU Burst Time
P_1	102
P_2	3
P_3	3

- Processes arrive at time 0 in the order: P_1 , P_2 , P_3
- FCFS Scheduling:



- Average turnaround time: (102+105+108)/3 = 105

A Twist on the FCFS Example

- What if the processes arrive in the order

  P_2 (3), P_3 (3), P_1 (102)

- Average turnaround time: _____

- Long processes love FCFS; short processes DON'T!

FCFS Major Disadvantage

Short processes may
be
stuck
behind long processes
(Convoy effect)

So What Should We Do?

- How can we develop better algorithms to deal with the reality that jobs run for different amounts of time?

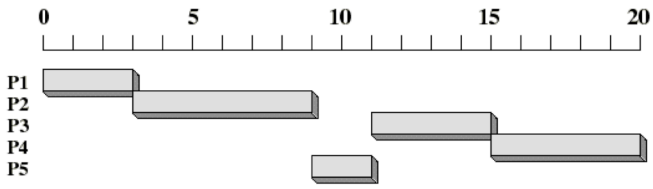
- Solutions:
  - Shortest Job First (SJF)
  - Shortest Time to Completion First (STCF)
  - Round Robin (RR)

Shortest Job First (SJF)

- Select the shortest process next (the one with the shortest next CPU burst)
  - Need to know the length the next CPU burst

SJF Example

Process	Arrival Time	CPU Burst Time
P_1	0	3
P_2	2	6
P_3	4	4
P_4	6	5
P_5	8	2



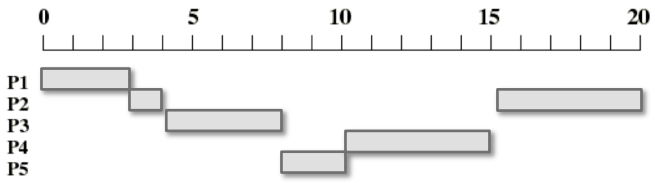
Why SJF is Not That Great

- What if a short process arrives just after a LONG process started executing?

- Solution: Shortest Time to Completion First
  - Any time a new job enters the system, the STCF scheduler determines which of the remaining jobs (including the new job) has the least time left, and schedules that one

STCF Example (Preemptive)

Process	Arrival Time	CPU Burst Time
P_1	0	3
P_2	2	6
P_3	4	4
P_4	6	5
P_5	8	2



STCF is *Optimal* but Unfair

- Gives *minimum* average time spent by processes in the Ready Queue
- However:
  - Long jobs may starve if too many short jobs

A New Metric: *Response Time*

- Users of time-shared machines demand interactive performance
- *Response time* becomes important
  - time from when the job arrives in a system to the first time it is scheduled

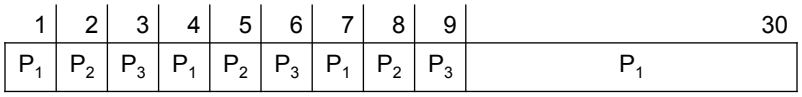
$$T_{response} = T_{firstrun} - T_{arrival}$$

- How can we build a scheduler that is sensitive to response time?
  - Round Robin

Round Robin

- Instead of running jobs to completion,
  - Run process for some "time quantum" (time slice)
  - When time is up, move it to the back of the Ready queue

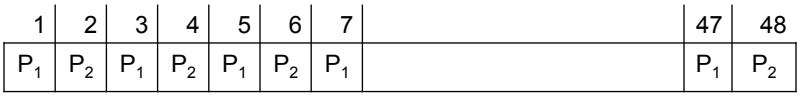
- Advantages:
  - Fair allocation of CPU across jobs
  - Assume $P_1(24)$, $P_2(3)$, $P_3(3)$ and time quantum = 1



  - What is the average response time? (0+1+2)/3 = 1

A Twist on the RR Example

- What if the processes have similar sizes? Assume 2 processes of 24 time units each:



  - What is the average turnaround time?
  - How does it compare to FCFS for the same two processes?

Same RR Example with Time Slice = 8

- Two processes of 24 time units each, time slice = 8

8	16	24	32	40	48
P_1	P_2	P_1	P_2	P_1	P_2

  - What is the average turnaround time?
  - How does it compare to FCFS for the same two processes?

- Main interesting thing is the length of the time slice

Round Robin's Main Disadvantage

Performance depends
on
the sizes of processes and
the length of the time slice

- Timeslice frequently set to ~100 milliseconds
- Context switching typically cost < 1 millisecond
  - negligible (< 1% per time slice)

Traditional UNIX Scheduling

- A *priority number* is associated with each process
- 128 priorities possible (0-127)
- One Round Robin queue per priority
- At every scheduling event, the scheduler
  - picks the non-empty queue of highest priority
  - runs processes in round-robin
  - priorities are adjusted based on the amount of CPU used, the current load, and how long the process has been waiting

Linux Lottery Scheduling

- Give each process some number of tickets
- At each scheduling event, randomly pick a ticket
- Run winning process
- More tickets implies more CPU time
- How to use?
  - Give few tickets to processes of low priority, and many tickets to processes of high priority
- If a job has at least one ticket, it won't starve

Summary

- FCFS:
  - *Advantage*: simple
  - *Disadvantage*: short jobs can get stuck behind long ones
- SJF, STCF:
  - *Advantage*: optimal
  - *Disadvantage*: hard to predict the future
  - (SJF) unfair to long running jobs
- RR:
  - *Advantage*: better response time, better for short jobs
  - *Disadvantage*: poor when jobs are the same length
- Every system uses a combination of these