IT308: Operating Systems

CPU Scheduling

Process states

- Each process has an execution state, which indicates what it is currently doing
  - **ready**: waiting to be assigned to CPU
    - could run, but another process has the CPU
  - **running**: executing on the CPU
    - is the process that currently controls the CPU

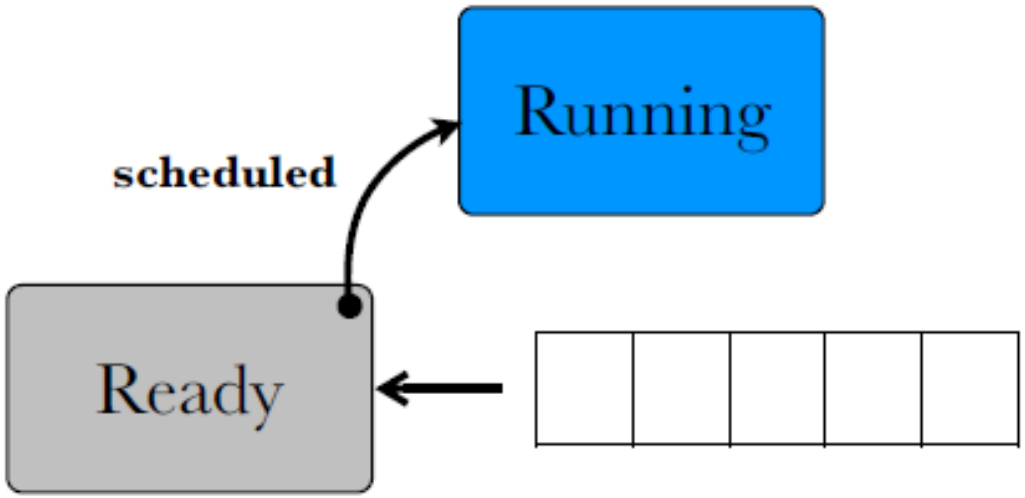
Process states

- Each process has an execution state, which indicates what it is currently doing
  - **ready**: waiting to be assigned to CPU
    - could run, but another process has the CPU
  - **running**: executing on the CPU
    - is the process that currently controls the CPU
    - pop quiz: how many processes can be in running state?

Process states

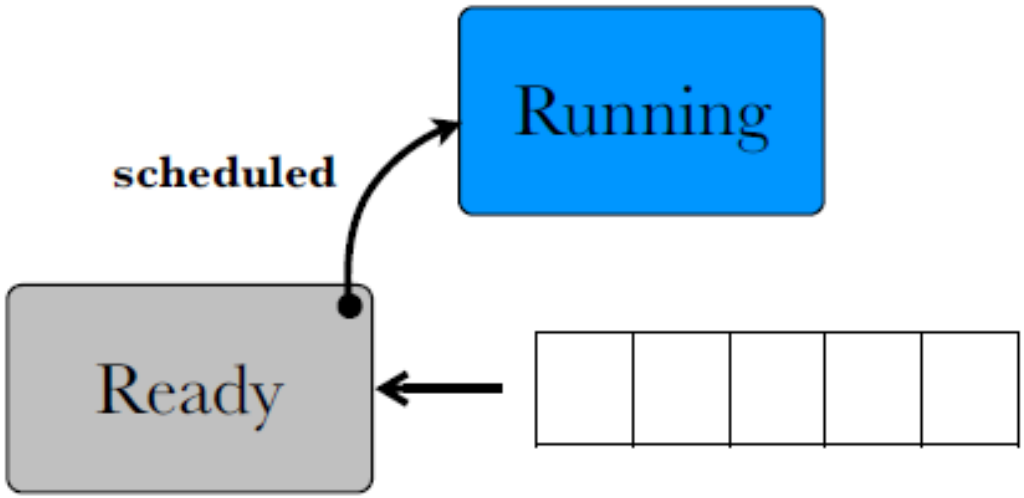
- Each process has an execution state, which indicates what it is currently doing
  - **ready**: waiting to be assigned to CPU
    - could run, but another process has the CPU
  - **running**: executing on the CPU
    - is the process that currently controls the CPU
    - pop quiz: how many processes can be in running state?
  - **waiting** = waiting for an event, e.g. I/O
    - cannot make progress until event happens

Scheduling policies



- the scheduling policy: how does the OS select a process from the ready queue to execute?

Scheduling policies



- the scheduling policy: how does the OS select a process from the ready queue to execute?
- CPU scheduler invoked to carry out scheduling policies during hardware interrupts and also system calls

Scheduling metrics

- scheduling goals are usually predicated on optimizing certain scheduling metrics
- performance metric: turnaround time
  - the time at which the job completes minus the time at which the job arrived in the system

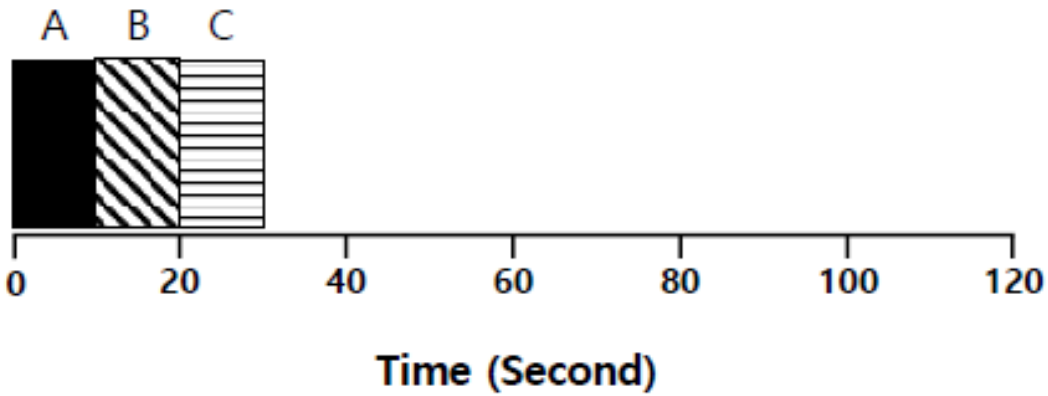
$$\boxed{T_{turnaround} = T_{completion} - T_{arrival}}$$

Workload assumptions

- initially, but will be relaxed as we go:
  - Each job runs for the same amount of time.
  - All jobs arrive at the same time.
  - All jobs only use the CPU (i.e., they perform no I/O).
  - The run-time of each job is known.

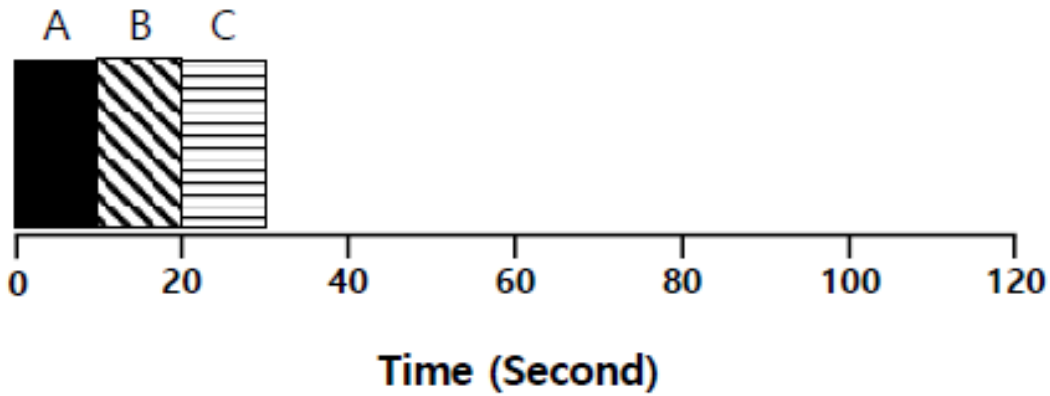
First Cum First Served (FCFS)

- Example:
  - A arrived just before B which arrived just before C.
  - Each job runs for 10 seconds.



First Cum First Served (FCFS)

- Example:
  - A arrived just before B which arrived just before C.
  - Each job runs for 10 seconds.



  - Average turnaround time $= (10+20+30)/3 = 20$ sec

First Cum First Served

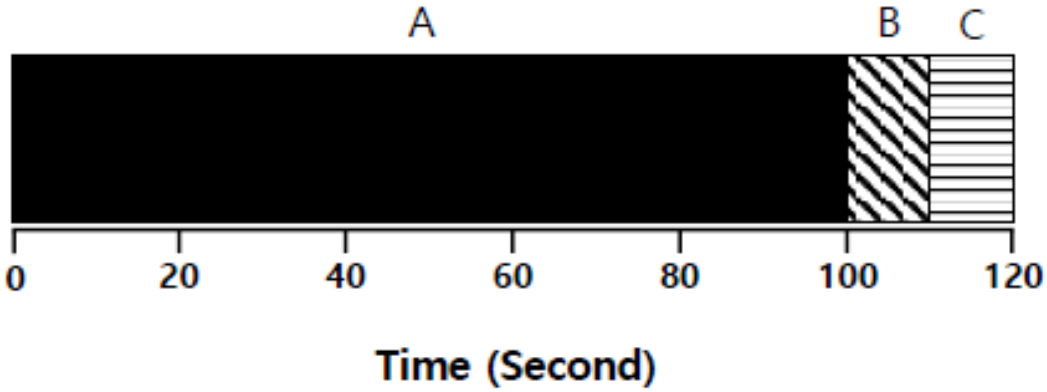
- What kind of workload could you construct to make FCFS perform poorly?

First Cum First Served

- What kind of workload could you construct to make FCFS perform poorly?
- Let's relax assumption 1: Each job no longer runs for the same amount of time.

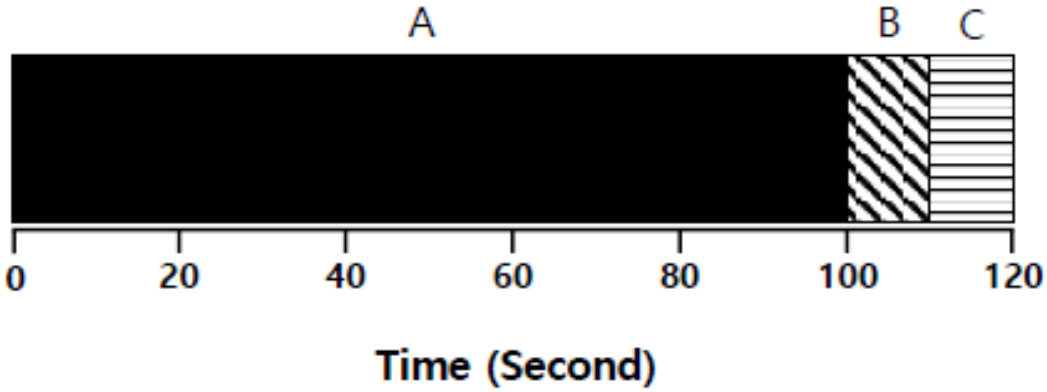
First Cum First Served

- What kind of workload could you construct to make FCFS perform poorly?
- Let's relax assumption 1: Each job no longer runs for the same amount of time.
- Example:
  - A arrived just before B which arrived just before C.
  - runs for 100 seconds, B and C run for 10 each.



First Cum First Served

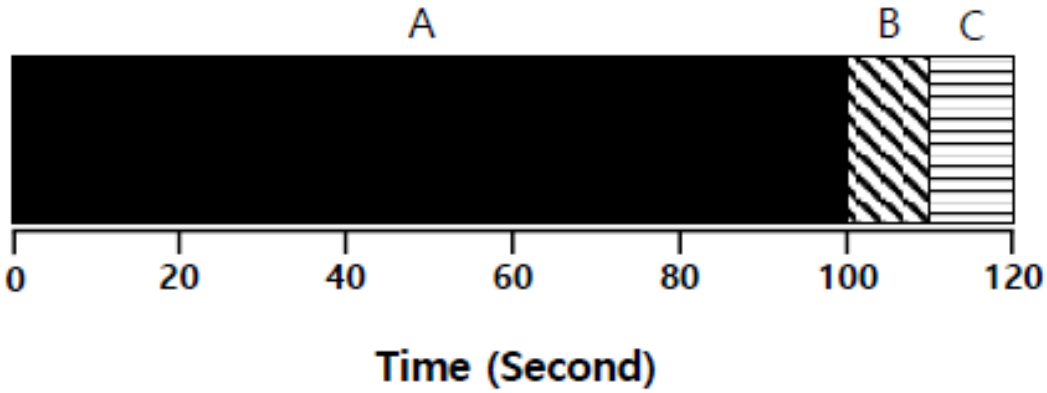
- What kind of workload could you construct to make FCFS perform poorly?
- Let's relax assumption 1: Each job no longer runs for the same amount of time.
- Example:
  - A arrived just before B which arrived just before C.
  - runs for 100 seconds, B and C run for 10 each.



  - Average turnaround time $= (100+110+120)/3 = 110$ sec

First Cum First Served

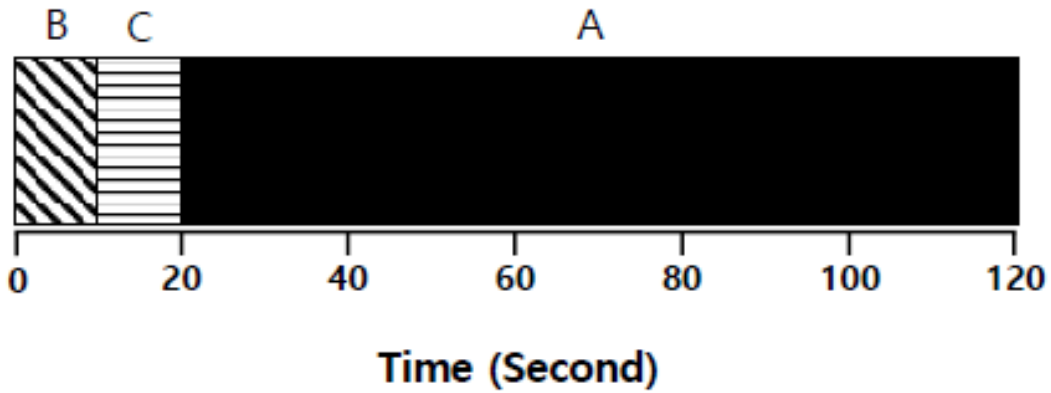
- What kind of workload could you construct to make FCFS perform poorly?
- Let's relax assumption 1: Each job no longer runs for the same amount of time.
- Example:
  - A arrived just before B which arrived just before C.
  - runs for 100 seconds, B and C run for 10 each.



  - Average turnaround time = (100+110+120)/3 = 110 sec
  - Convoy effect (supermarket checkout)

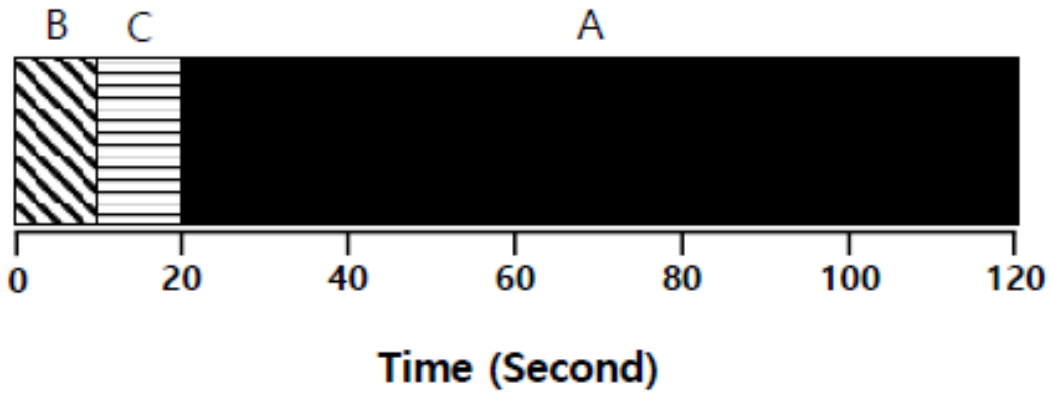
Shortest Job First (SJF)

- Run the shortest job first, then the next shortest, and so on
  - Non-preemptive = scheduler not allowed to interrupt a running process
- Example:
  - A arrived just before B which arrived just before C.
  - A runs for 100 seconds, B and C run for 10 each.



Shortest Job First (SJF)

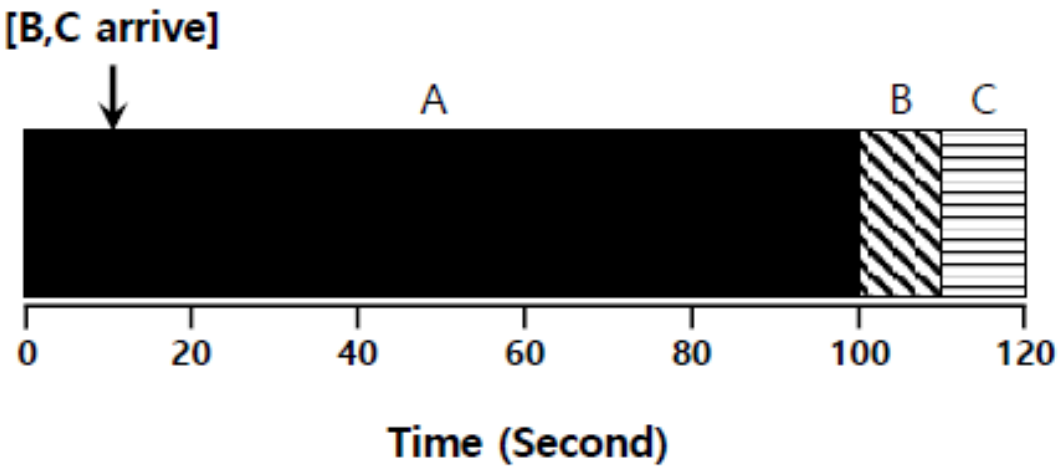
- Run the shortest job first, then the next shortest, and so on
  - Non-preemptive = scheduler not allowed to interrupt a running process
- Example:
  - A arrived just before B which arrived just before C.
  - A runs for 100 seconds, B and C run for 10 each.



  - Average turnaround time = (10+20+120)/3 = 50 sec

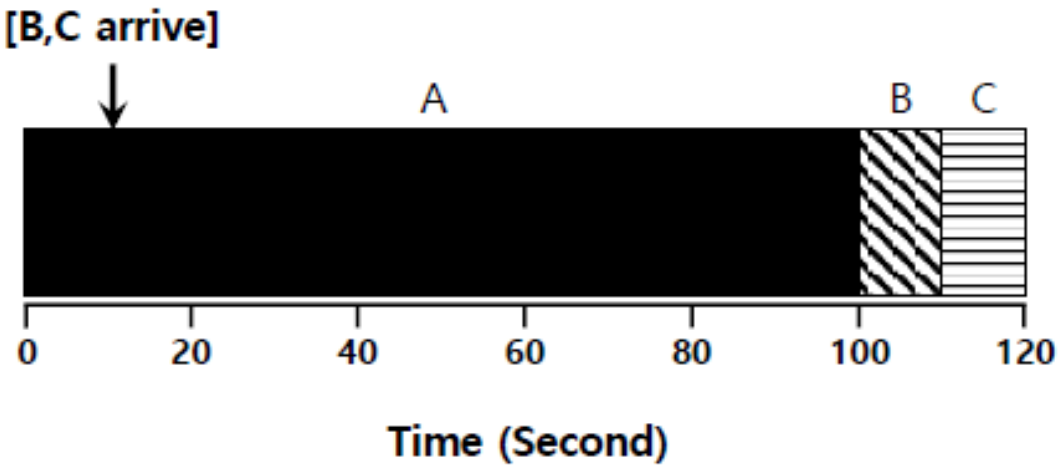
SJF with late arrivals from B and C

- Let's relax assumption 2: Jobs can arrive at any time.
- Example:
  - A arrives at t=0 and needs to run for 100 seconds.
  - B and C arrive at t=10 and each need to run for 10 seconds



SJF with late arrivals from B and C

- Let's relax assumption 2: Jobs can arrive at any time.
- Example:
  - A arrives at t=0 and needs to run for 100 seconds.
  - B and C arrive at t=10 and each need to run for 10 seconds



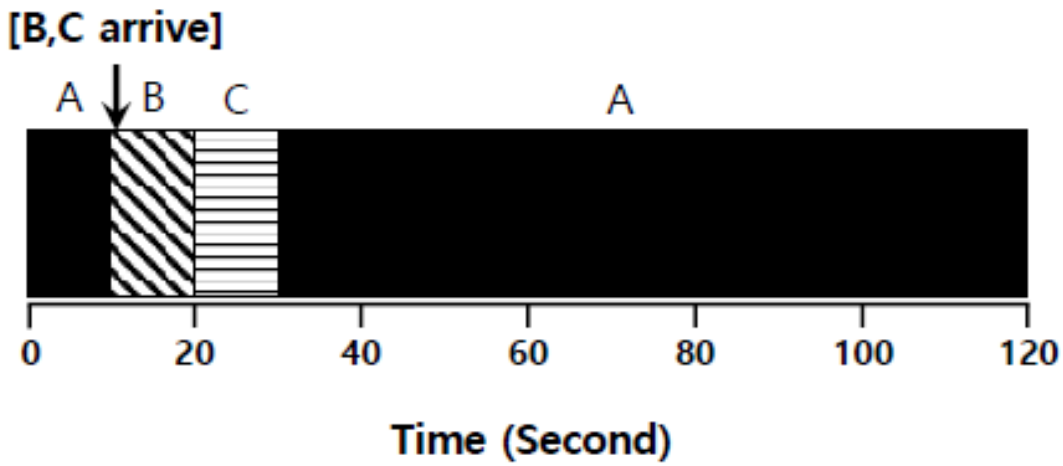
  - Avg. turnaround time = (100 + (110-10) + (120-10))/3 = 103.3 sec

Shortest Time-to-Completion First (STCF)

- Add preemption to SJF
  - Also knows as Preemptive Shortest Job First (PSJF)
- A new job enters the system:
  - Determine run-time of the remaining jobs and new job
  - Schedule the job which has the least time left

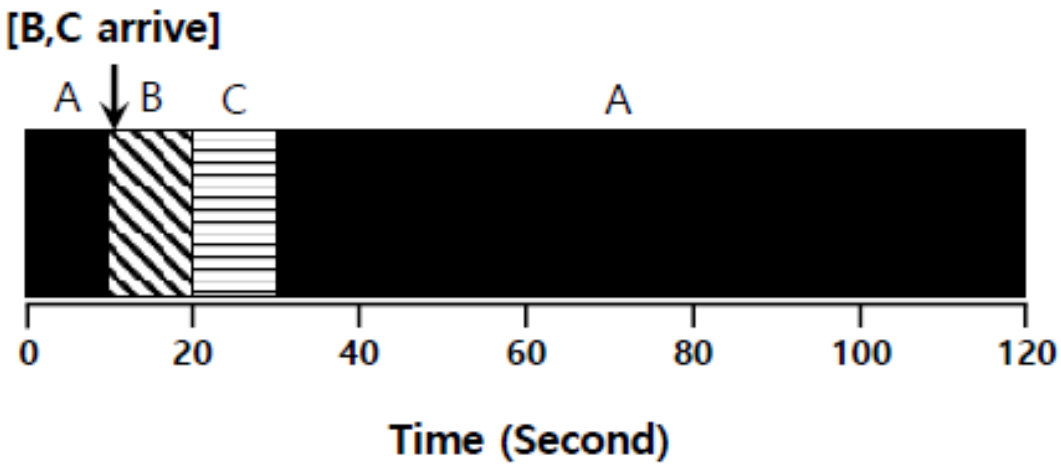
Shortest Time-to-Completion First (STCF)

- Example:
  - A arrives at t=0 and needs to run for 100 seconds.
  - B and C arrive at t=10 and each need to run for 10 seconds



Shortest Time-to-Completion First (STCF)

- Example:
  - A arrives at t=0 and needs to run for 100 seconds.
  - B and C arrive at t=10 and each need to run for 10 seconds



- Avg. turnaround time = ((120-0) + (20-10) + (30-10))/3 = 50 sec

New scheduling metric: Response time

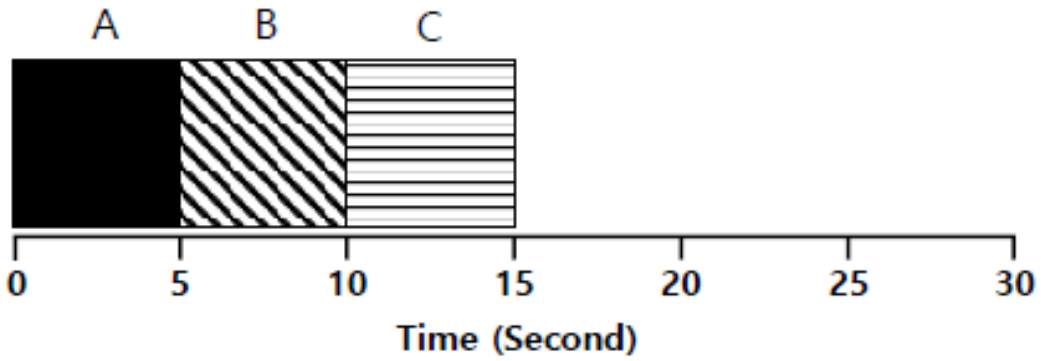
- The time from when the job arrives to the first time it is scheduled.

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

- SJF (and STCF) are not particularly good for response time.

Why SJF is not that great for response time?

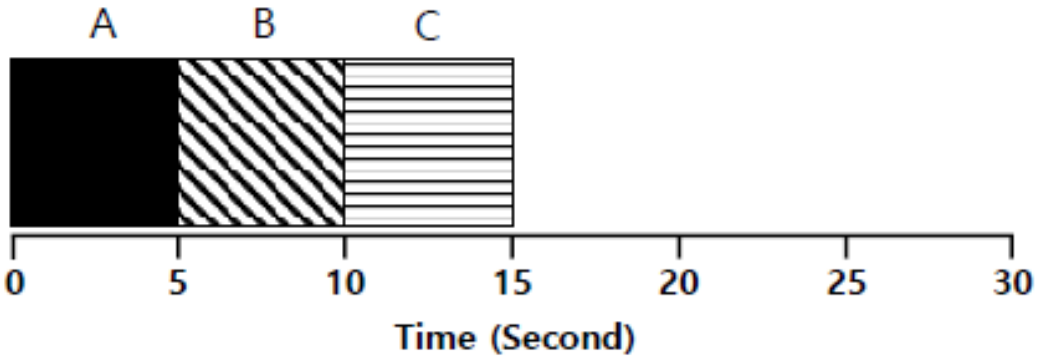
- A, B and C arrive at the same time.
- They each wish to run for 5 seconds.



SJF (Bad for Response Time)

Why SJF is not that great for response time?

- A, B and C arrive at the same time.
- They each wish to run for 5 seconds.



SJF (Bad for Response Time)

- Average response time $= (0+5+10)/3 = 5$ sec

Round Robin (RR) Scheduling

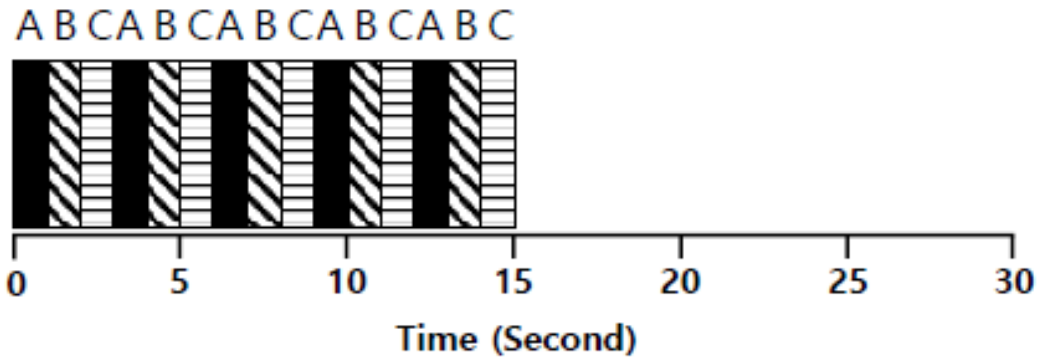
- How can we build a scheduler that is sensitive to response time?

Round Robin (RR) Scheduling

- How can we build a scheduler that is sensitive to response time?
- Time slicing Scheduling
  - Run a job for a time slice and then switch to the next job in the run queue until the jobs are finished.
    - Time slice is sometimes called a scheduling quantum.
  - It repeatedly does so until the jobs are finished.
  - The length of a time slice must be a multiple of the timer-interrupt period.

RR Scheduling Example

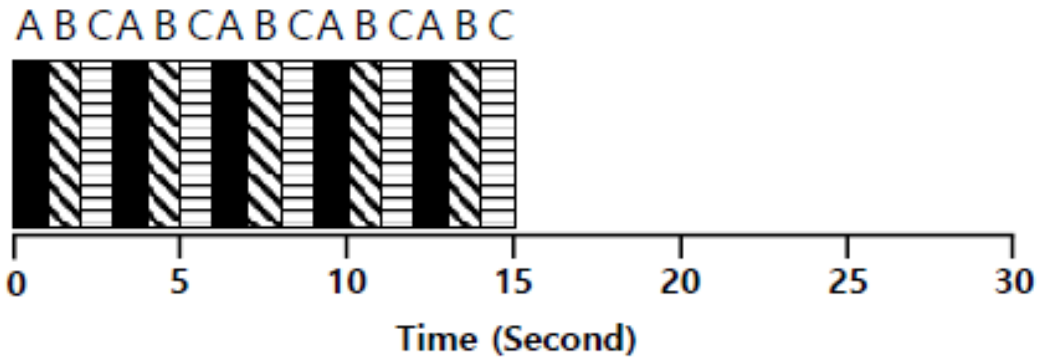
- A, B and C arrive at the same time.
- They each wish to run for 5 seconds.



RR with a time-slice of 1sec (Good for Response Time)

RR Scheduling Example

- A, B and C arrive at the same time.
- They each wish to run for 5 seconds.



RR with a time-slice of 1sec (Good for Response Time)

- Average response time $= (0+1+2)/3 = 1$ sec

The length of the time slice is critical

- The shorter time slice
  - Better response time
  - The cost of context switching will dominate overall performance.

The length of the time slice is critical

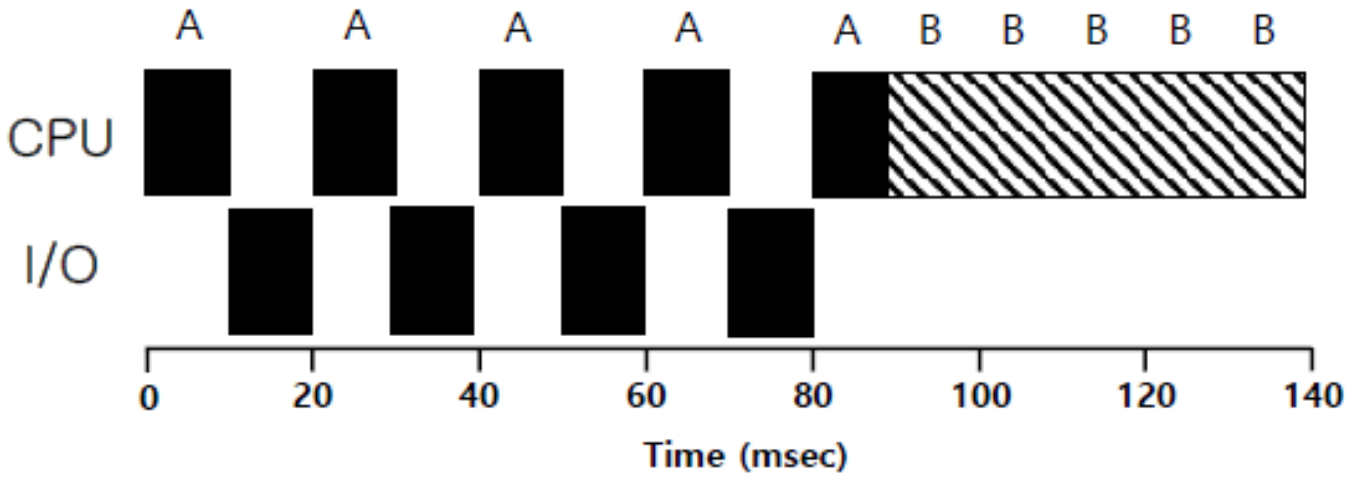
- The shorter time slice
  - Better response time
  - The cost of context switching will dominate overall performance.
- The longer time slice
  - Amortize the cost of switching
  - Worse response time

So Far

- Two types of schedulers
  - SJF and STCF: optimize turnaround time, but are bad for response time
  - RR: optimize response time, but is bad for turnaround time

Incorporating I/O

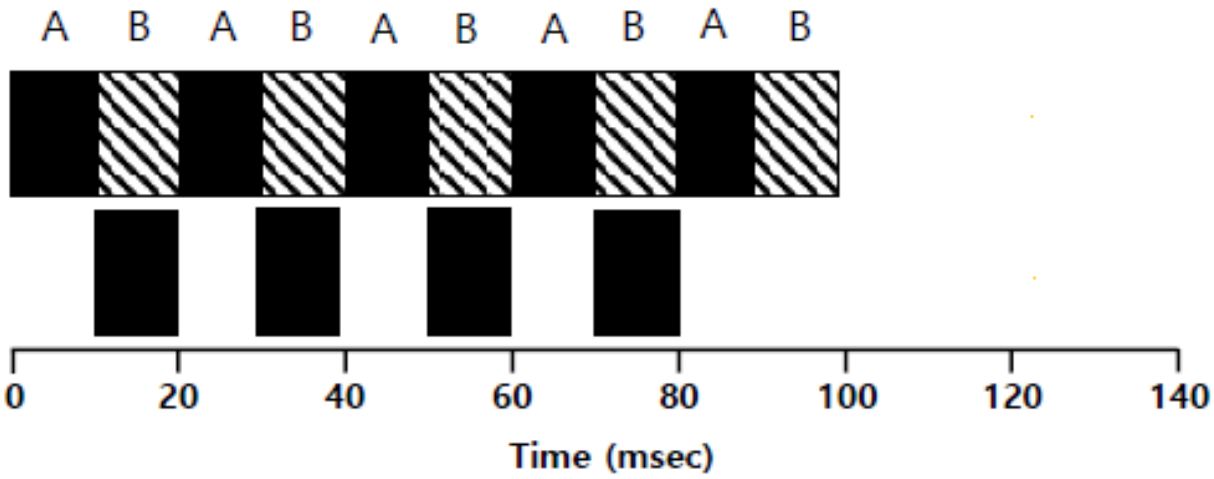
- Let's relax assumption 3: we also consider programs that do I/O
- Example:
  - A and B need 50 ms of CPU time each.
  - A runs for 10 ms and then issues an I/O request
    - I/Os each take 10 ms
  - B simply uses the CPU for 50 ms and performs no I/O
  - The scheduler runs A first, then B after



Poor Use of Resources

Incorporating I/O (contd.)

- Overlap CPU & I/O: Better use of CPU



Overlap Allows Better Use of Resources

Incorporating I/O (contd.)

- When a job initiates an I/O request
  - The job is blocked waiting for I/O completion.
  - The scheduler should schedule another job on the CPU.
- When the I/O completes
  - An interrupt is raised.
  - The OS moves the process from blocked back to the ready state.

Problems

- we would like to
  - optimize turnaround time
  - minimize response time (for interactive users)
- Two types of schedulers
  - SJF and STCF: optimize turnaround time, but are bad for response time
  - RR: optimize response time, but is bad for turnaround time

Lecture reading

Read OS:TEP Chapters 7 and 8