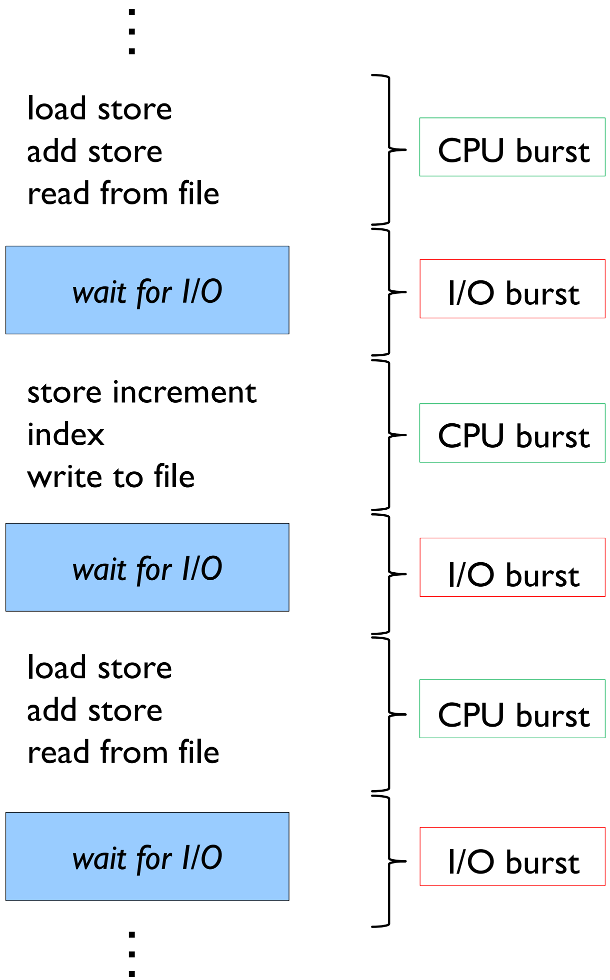
Operating System: CPU scheduling

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

CPU-I/O Burst Cycle

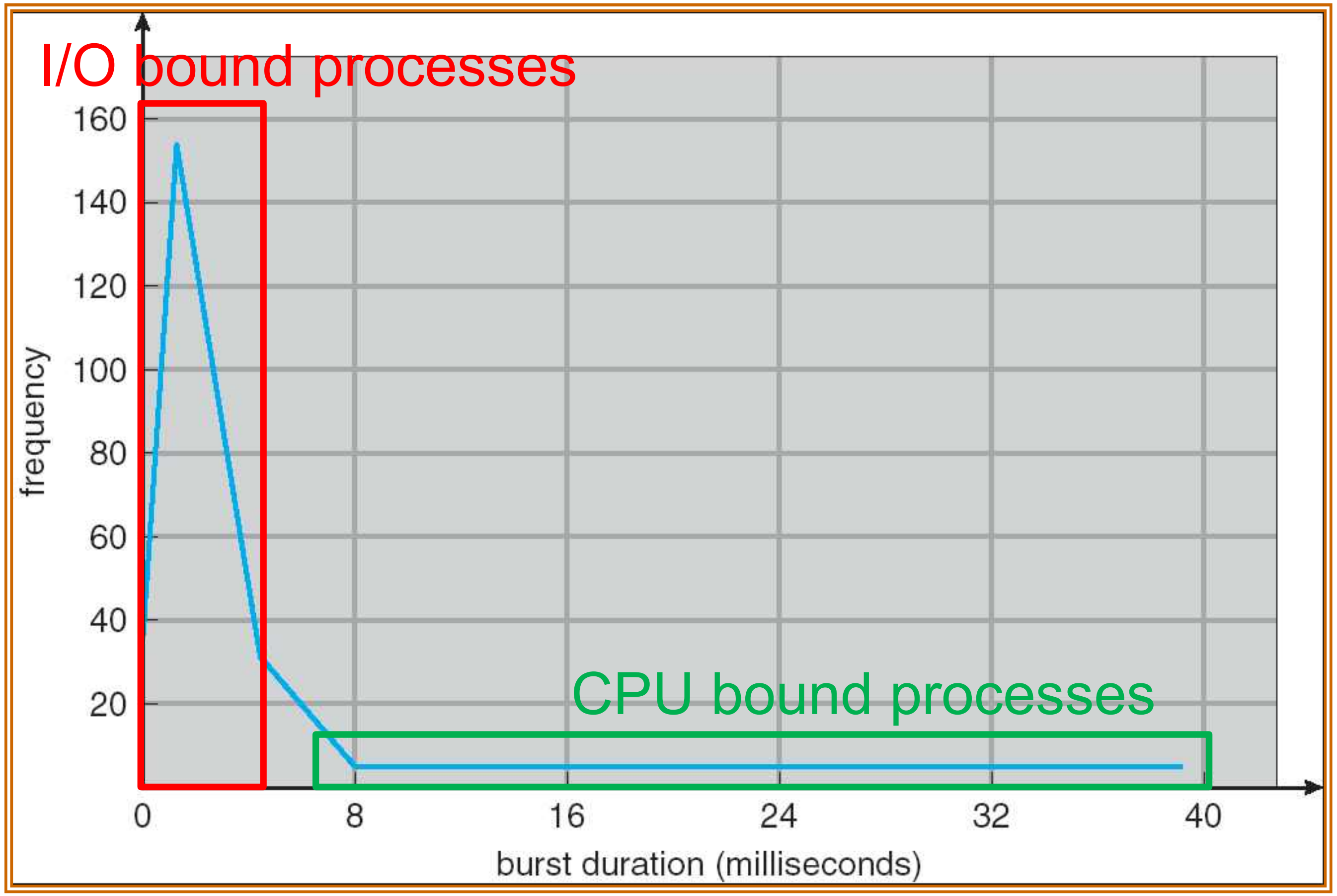
- Alternating sequence of CPU bursts and I/O bursts



CPU-I/O Burst Cycle

- Histogram of CPU burst times

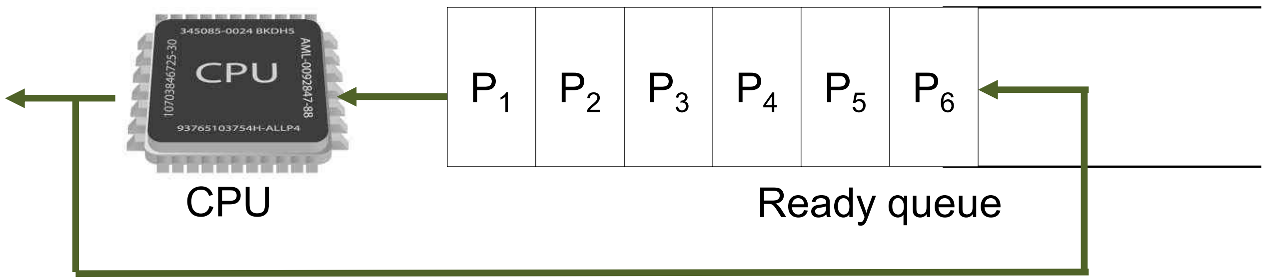
빈도수



CPU사용시간

CPU Scheduler

- CPU scheduler
  - selects a process from the processes that are ready to execute



- CPU scheduling decisions may take place when
  - (1) a process switches from running to waiting state (e.g. I/O request),
  - (2) a process switches from running to ready state (e.g. time slice expiration),
  - (3) a process switches from waiting to ready (e.g. I/O completion), or
  - (4) a process terminates
    - ➔ Scheduling under (1) and (4) is non-preemptive — CPU 자원 자발적 반납.
    - ➔ Scheduling under (2) and (3) is preemptive — 선점 — OS가 CPU 자원 배분 결정.

Scheduling Criteria

- CPU utilization
  - keeps the CPU as busy as possible. (0% ~ 100%)
- Throughput
  - The number of processes that are completed per time unit
- Turnaround time
  - Time from the submission of a request to time of completion
- Waiting time
  - Sum of time a process has been waiting in the ready queue
- Response time
  - Time from the submission of a request until the first response is produced

수직 좋대,

Scheduling Criteria

- Scheduling algorithm's goals
  - Maximize the CPU utilization
  - Maximize the throughput
  - Minimize the turnaround time
  - Minimize the waiting time
  - Minimize the response time
    - ➢ It may be more important to minimize the variance than the average of response time
    - ➢ E.g. interactive system

하나가 높아지면 다른게 낮아질 수도 있어서 목적에 맞게 설계함.

평균보다 분산이 작은게 중요 (알고리즘 차원)

Scheduling: Introduction

- Workload assumptions:
  1. Each job runs for the **same amount of time**
  2. All jobs **arrive** at the same time
  3. All jobs only use the **CPU** (i.e., they perform no I/O)
  4. The **run-time** of each job is known

Scheduling Metrics

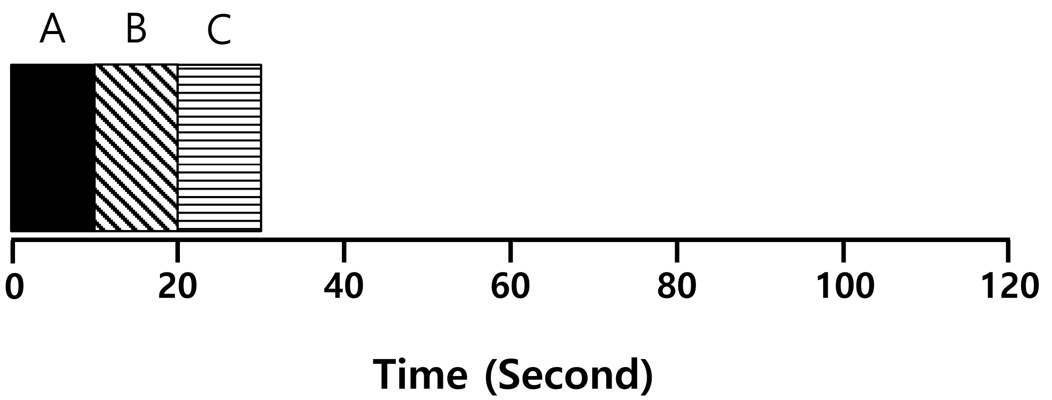
- Performance metric: Turnaround time
  - The time at which **the job completes** minus the time at which **the job arrived** in the system

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

- Another metric is fairness
  - Performance and fairness are often at odds in scheduling

First In, First Out (FIFO)

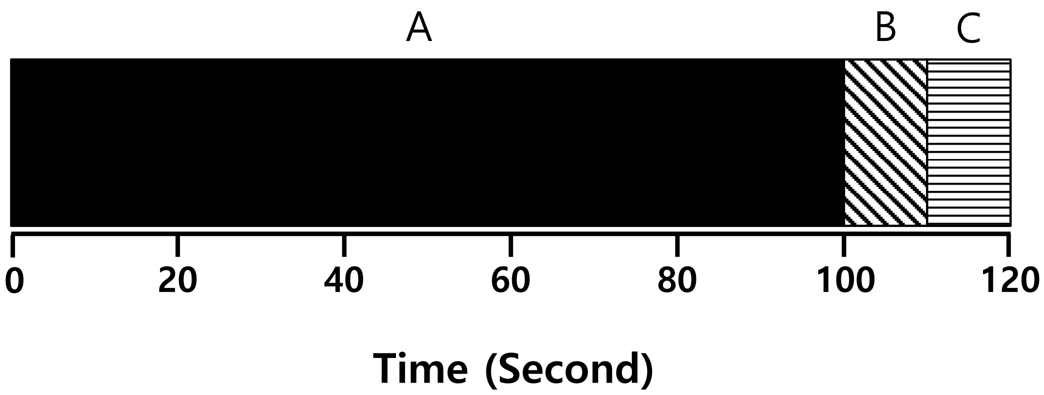
- First Come, First Served (FCFS)
  - Very simple and easy to implement
- Example:
  - A arrived just before B which arrived just before C
  - Each job runs for 10 seconds



$$Average\ turnaround\ time = \frac{10 + 20 + 30}{3} = 20\ sec$$

Why FIFO is not that great? - Convoy effect

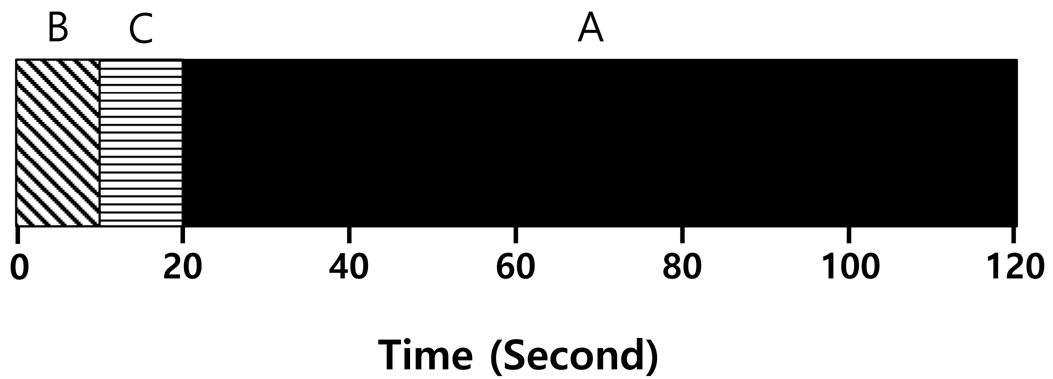
- Let’s relax assumption 1: Each job **no longer** runs for the same amount of time
- Example:
  - A arrived just before B which arrived just before C
  - A runs for 100 seconds, B and C run for 10 each



$$Average\ turnaround\ time = \frac{100 + 110 + 120}{3} = 110\ sec$$

Shortest Job First (SJF)

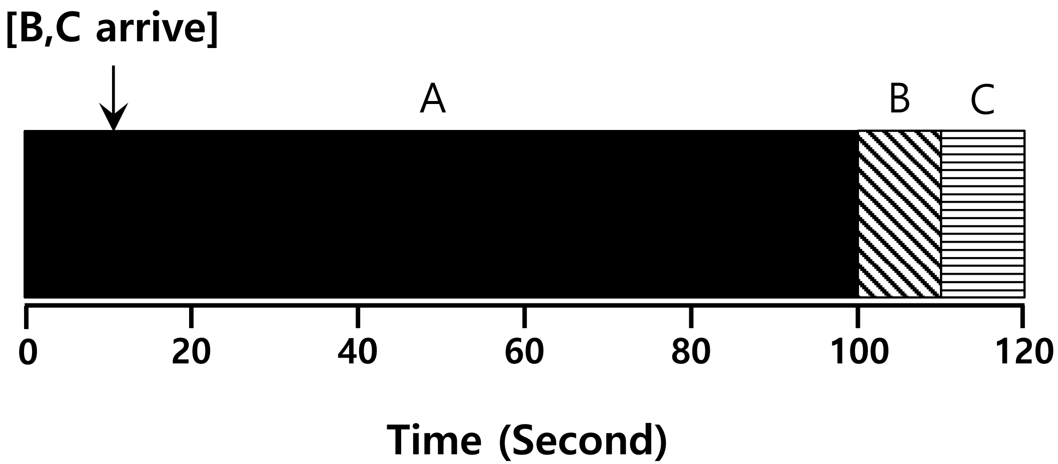
- Run the shortest job first, then the next shortest, and so on
  - Non-preemptive scheduler
- Example:
  - A arrived just before B which arrived just before C
  - A runs for 100 seconds, B and C run for 10 each



$$Average\ turnaround\ time = \frac{10 + 20 + 120}{3} = 50\ sec$$

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



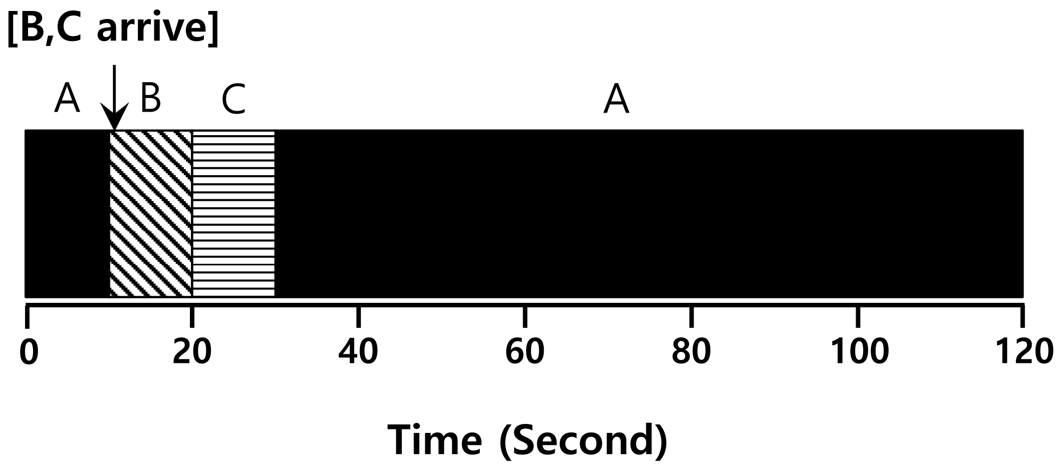
$$\text{Average turnaround time} = \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33 \text{ sec}$$

Shortest Time-to-Completion First (STCF)

- Add preemption to SJF
  - Jobs are not available simultaneously (Assumption 2 relaxed)
  - Preemptive version of SJF (Assumption 3 relaxed)
  - Also knows as Preemptive Shortest Job First (PSJF) or Shortest Remaining Time First (SRTF)
- A new job enters the system:
  - Determine 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



$$Average\ turnaround\ time = \frac{(120-0)+(20-10)+(30-10)}{3} = 50\ sec$$

SJF Implementation

- Determining the lengths of next CPU bursts time is required

- Estimation by using the length of previous CPU burst times (exponential averaging)

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$ ➔ 적응의 속도
4. Define : $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n$

SJF Implementation (Cont.)

- Next CPU burst time

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

  - $\alpha = 0$
    - ➢ $\tau_{n+1} = \tau_n$
    - ➢ Recent history has no effect
  - $\alpha = 1$
    - ➢ $\tau_{n+1} = t_n$
    - ➢ Only the most recent CPU burst matters

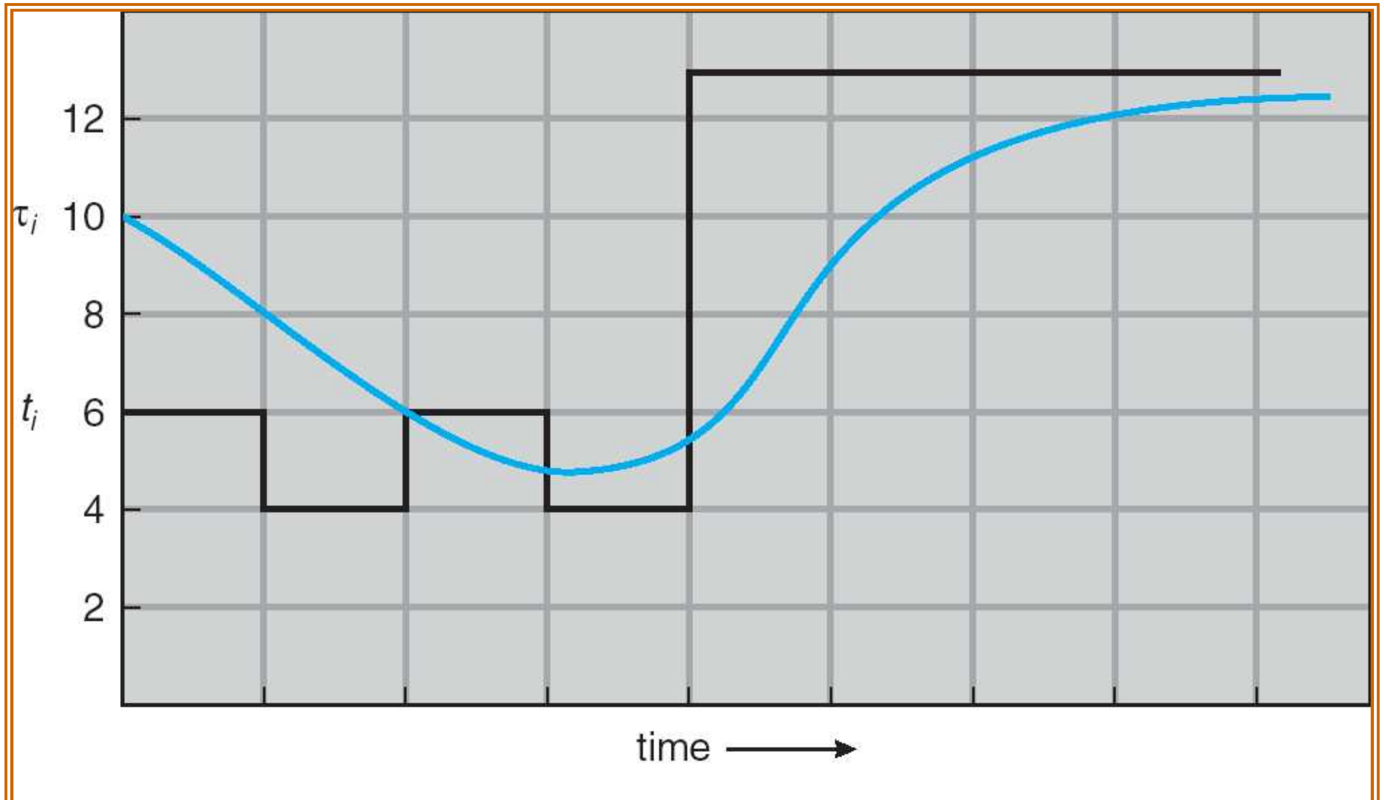
- If we expand the formula, we get

$$\begin{aligned}\tau_{n+1} &= \alpha\, t_n + (1-\alpha)\alpha\, t_{n-1} + \cdots \\ &\quad + (1-\alpha)^j \alpha\, t_{n-j} + \cdots \\ &\quad + (1-\alpha)^{n+1} \tau_0\end{aligned}$$

  - Since both α and $(1-\alpha)$ are less than or equal to 1,
  - Each successive term has less weight than its predecessor

SJF Implementation (Cont.)

- Prediction of the length of the next CPU burst time



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

New scheduling metric: Response time

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

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

  – STCF and related disciplines are not particularly good for response time

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

Round Robin (RR) Scheduling

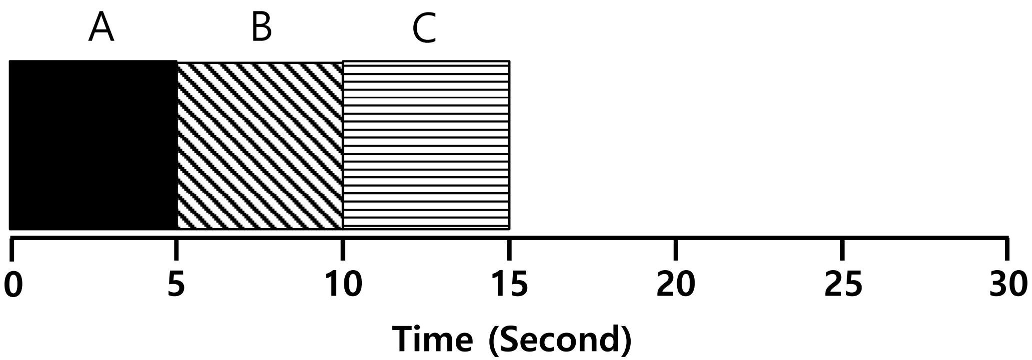
- Time slicing Scheduling
  - Run a job for a time slice and then switch to the next job in the **run queue** until the jobs are finished
    - ➢ Time slice is sometimes called a scheduling quantum
  - It repeatedly does so until the jobs are finished
  - The length of a time slice must be *a multiple of* the timer-interrupt period

RR is fair, but performs poorly on metrics such as turnaround time

response time에 좋지만
turnaround에서 안좋아.

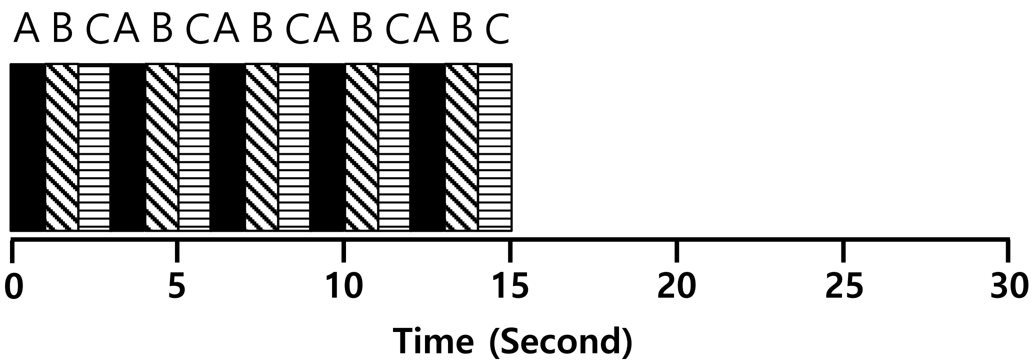
RR Scheduling Example

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



SJF (Bad for Response Time)

$$T_{average\ \ response} = \frac{0+5+10}{3} = 5sec$$



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

$$T_{average\ \ response} = \frac{0+1+2}{3} = 1sec$$

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

Deciding on the length of the time slice presents a trade-off to a system designer

Incorporating I/O

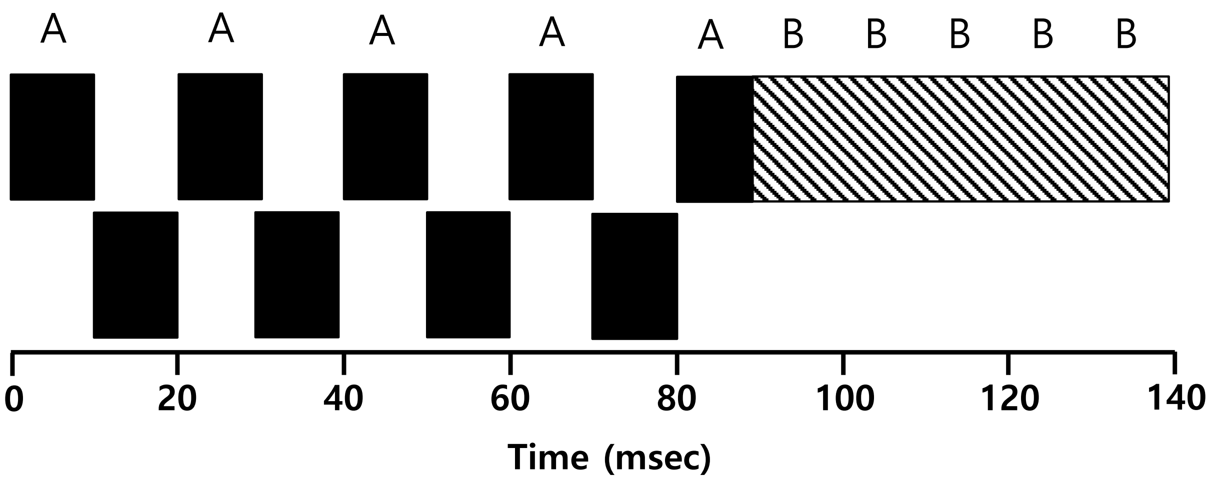
- Let's relax assumption 3: All programs perform I/O

- Example:
  - A and B need 50ms of CPU time each
  - A runs for 10ms and then issues an I/O request
    - ➢I/Os each take 10ms
  - B simply uses the CPU for 50ms and performs no I/O
  - The scheduler runs A first, then B after

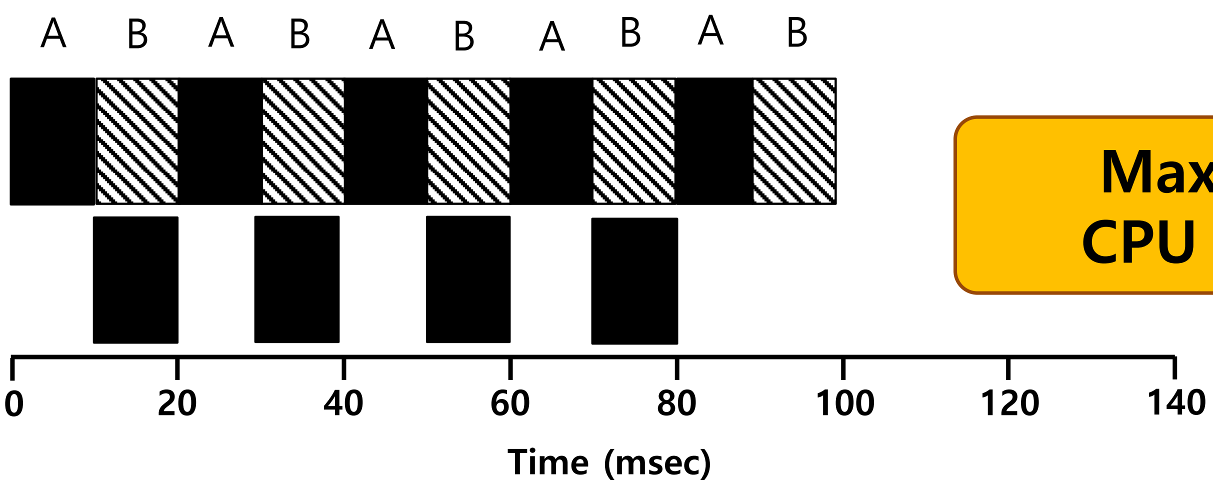
Incorporating I/O (Cont.)

CPU

I/O



Poor Use of Resources



Overlap Allows Better Use of Resources

Maximize the CPU utilization

Incorporating I/O (Cont.)

- 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