Scheduling: Introduction

Operating System (OS)

안인규

2025.03.13

강의자료는 KAIST 원유집 교수님 OS 강의 자료를 참고했습니다.

Scheduling?

- Scheduling?
  - OS에서 실행할 프로세스 중, **어떤 프로세스를 언제 실행할지** 결정하는 과정 (based on the high-level **policies** (=disciplines))

Scheduling: Introduction

- Workload assumptions:
  1. Each job runs for the **same amount of time.**
  2. All jobs **arrive** at the same time.
  3. Once started, each job runs to completion
  4. All jobs only use the **CPU** (i.e., they perform no I/O).
  5. 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.

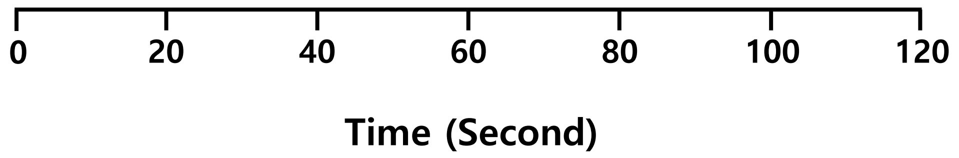
$$T_{turnaround} = T_{completion} - T_{arrival}$$

- Another metric is fairness
  - 프로세스들이 공평하게 CPU를 할당
  - Performance와 fairness는 종종 상충된다
  - Performance는 최적화되었지만, fairness는 감소할 수 있음

First In, First Out (FIFO)

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

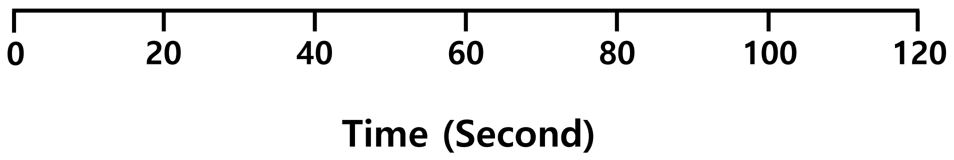
- First Come, First Served (FCFS)
  - Very simple and easy to implement
- Example:
  - 'A' arrived just before 'B' which arrived just before 'C'.
  - Each job runs for 10 seconds.



Convoy effect

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

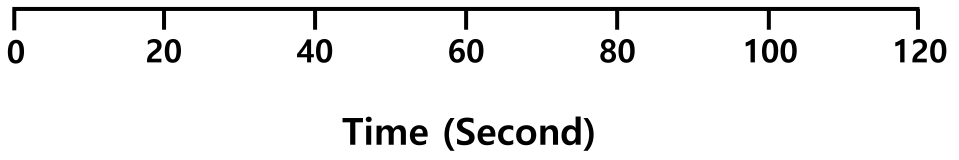
- Let's relax assumption 1: Each job **no longer** runs for the same amount of time.
- Example (FIFO):
  - 'A' arrived just before 'B' which arrived just before 'C'.
  - 'A' runs for 100 seconds, 'B' and 'C' run for 10 each.



Shortest Job First (SJF)

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

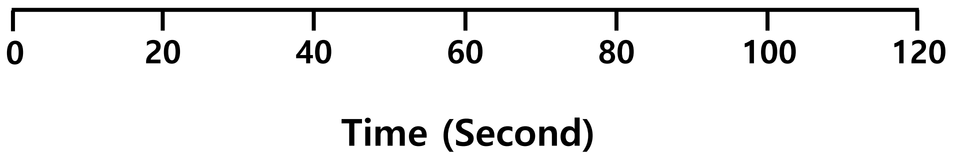
- Run the shortest job first (SJF), then the next shortest, and so on
  - Non-preemptive (비선점형) scheduler
- Example:
  - 'A' arrived just before 'B' which arrived just before 'C'.
  - 'A' runs for 100 seconds, 'B' and 'C' run for 10 each.



Late Arrivals from B and C

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

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



Shortest Time-to-Completion First (STCF)

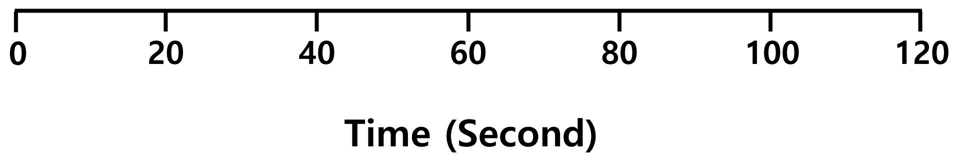
- Add preemption to SJF
  - Also knows as Preemptive Shortest Job First (PSJF)
- Let's relax assumption 3: Once started, each job runs to completion
- A new job enters the system:
  - Determine of the remaining jobs and new job
  - Schedule the job which has the lest time left

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

STCF

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

- 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



New scheduling metrics: 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?

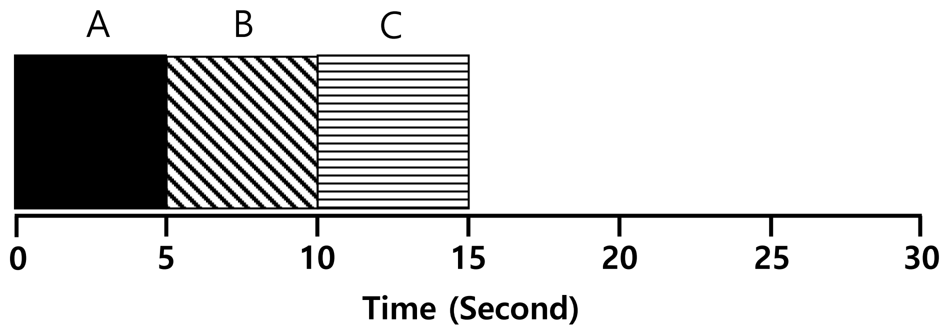
Round Robin (RR) Scheduling

- 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

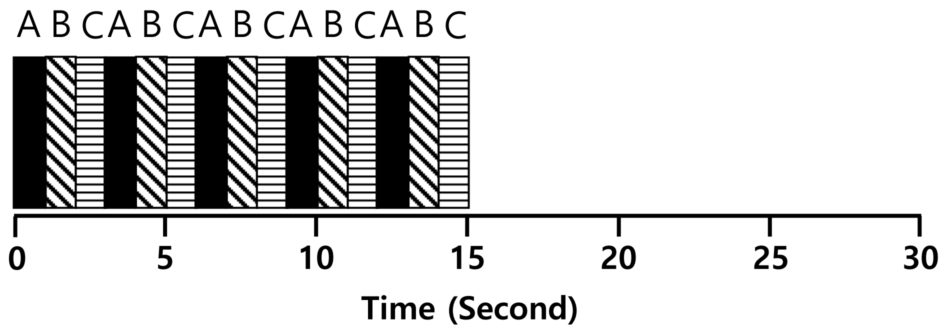
RR Scheduling Example

- 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
  - <u>The cost of context switching will dominate overall performance</u>

- The longer time slice
  - Reduce the cost of switching
  - Worse response time

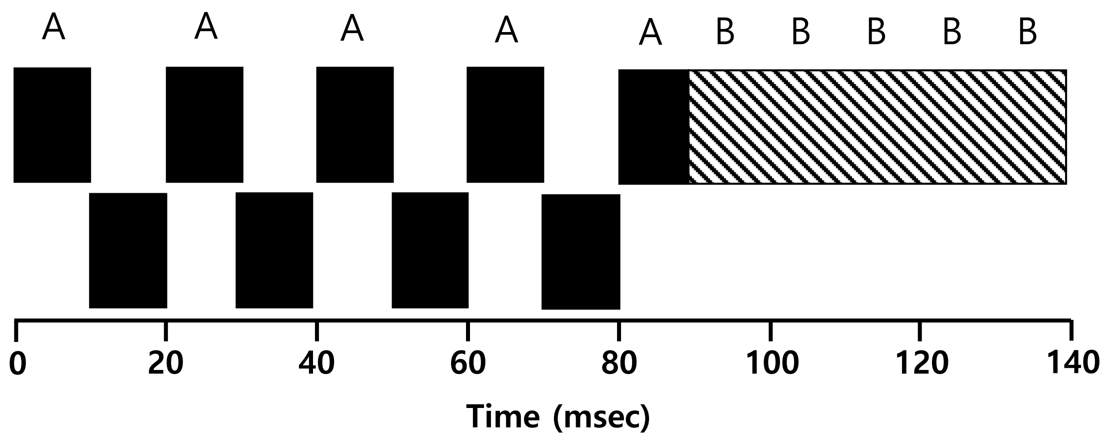
Deciding on the length of the time slice presents a trade-off to a system designer

Incorporating I/O

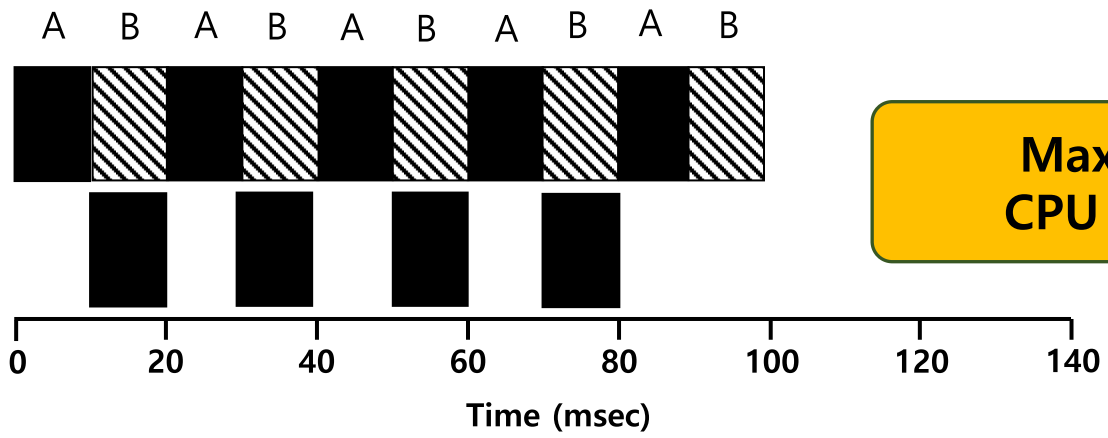
1. ~~Each job runs for the **same amount of time.**~~
2. ~~All jobs **arrive** at the same time.~~
3. ~~Once started, each job runs to completion~~
4. ~~All jobs only use the **CPU** (i.e., they perform no I/O).~~
5. The **run-time** of each job is known.

- Let's relax assumption 4: 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.)



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 (예: Disk Controller가 “데이터 읽기 완료” 신호를 보냄) → Interrupt Handler 실행
  - Interrupt Handler가 어떤 process 의 I/O 요청이 완료되었는지 확인
  - 작업이 완료된 process 를 Blocked → Ready 상태로 변경 (Ready process list 에 추가)

No More Oracle

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

- Let's relax assumption 5: The run-time of each job is known
  - The OS usually knows very little about the length of each job
  - How can we build an approach that behaves like SJF/STCF without such *a priori* knowledge?