Scheduling: Introduction

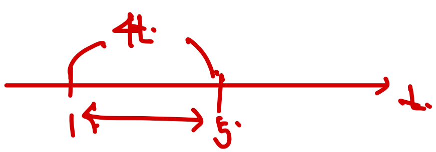
Prof. Yongtae Kim

Computer Science and Engineering
Kyungpook National University

Workload Assumptions and Scheduling Metrics

- **Now, we have to understand the high-level policies that an OS scheduler employs**
  - Before diving into the scheduling, let's make some assumption about the processes running in the system, which is called the workload

- **We make the following assumptions about the processes (jobs)**
  1) Each job run for the same amount of time
  2) All jobs arrives at the same time
  3) Once started, each job runs to completion
  4) All jobs only use the CPU (no I/O)
  5) The run-time of each job is known → 항상 문제이다



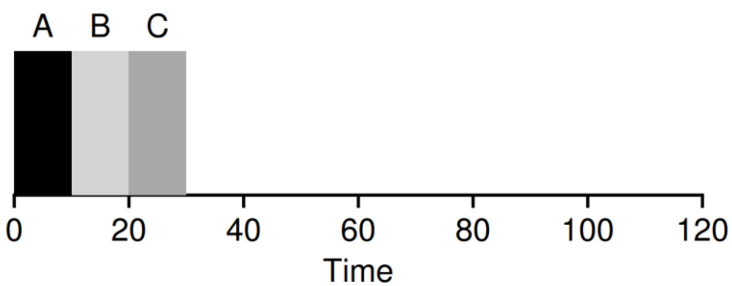
- **We need one thing to compare different scheduling policies**
  - Turnaround time is performance metric defined by $T_{turnaround} = T_{completion} - T_{arrival}$
  - In our assumptions, $T_{arrival} = 0$ and hence $T_{turnaround} = T_{completion}$
  - Another metric is fairness, which is often at odds with performance in scheduling

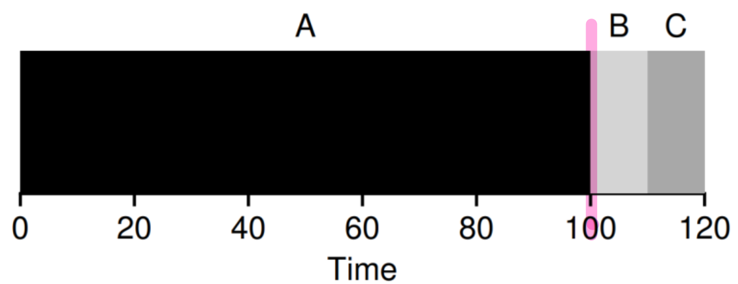
First In, First Out (FIFO)

(먼저 들어온 process 부터 먼저 나간다.)

- **The most basic algorithms is First In, First Out (FIFO) scheduling**
  - This is sometimes called First Come, First Served (FCFS)
  - It is simple and easy to implement, and works well in some cases

- **Let's do a quick example**
  1) Three jobs (A, B, C) arrive at the same time ($T_{arrival}$ = 0) and each job runs for 10; A arrived just a hair before B which did just before C
  2) What if A runs for 100 while B and C run for 10 each? (relaxing assumption #1)





1) $T_{turnaround}$ = (10+20+30)/3 = 20   2) $T_{turnaround}$ = (100+110+120)/3 = 110

- **The FCFS can cause a convoy effect**
  - A number of relatively-short potential consumers of a resource get queued behind a heavyweight resource consumer → long waiting time

FIFO FCFS

모든 작업은 실행 시간이 동일 relaxing FIFO x.

그럼 길이가 다르면 SJF

모든 작업은 동시에 도착 relaxing SJF x. → preemptive

STCF

모든 작업은 CPU에서만 동작 (I/O x) relaxing

SJF보다 turnaround 좋음

모든 작업의 실행 시간은 알고 있다 → STCF 중요함

Response time 관점

RR → turn-around time ↑ 나빠짐

time slice context-switching ↑↑

우선순위 한 번 실행되면 starving

RR 단점

relaxing STCF x. 시간정보 알 수 없음.

MLFQ → 대화형 작업. 반환 시간 최적화.

예측

사용 하는 재개

Reset

MLFQ

우선순위 높으면

작업 우선순위 같 → RR

새로운 작업 최고 우선순위

time slice 다 쓰면 ↓

조건 오래된 작업 가이드라인

대화형 ↑ batch ↓

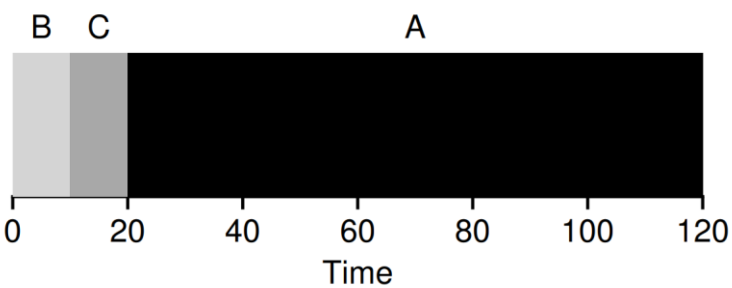
#hos

Shortest Job First (SJF)

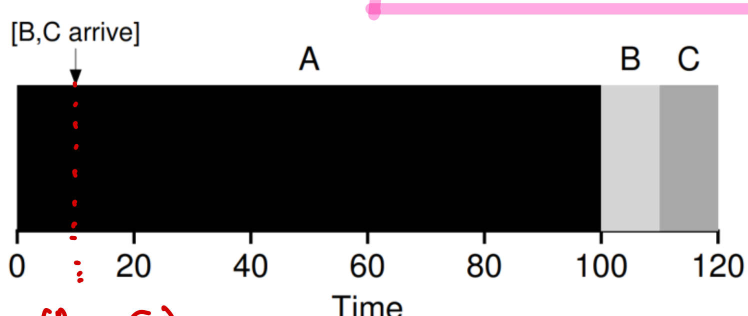
- **Shortest Job First (SJF) algorithm runs the shortest job first, then the next shortest, and so on**
  - This is a very simple approach to avoid the convoy effect

- **Let's take the same example but with SJF**
  1) SJF reduces average turnaround time from 110 to 50
  2) What if A arrives at t=0 and B and C arrive at t = 10? (relaxing assumption #2)

A가 수행 하는 중에 (B,C)가 옴.



1) $T_{turnaround} = (10+20+120)/3 = 50$



(B와 C)

2) $T_{turnaround} = (100+(110-10)+(120-10))/3 = 103.33$

<110

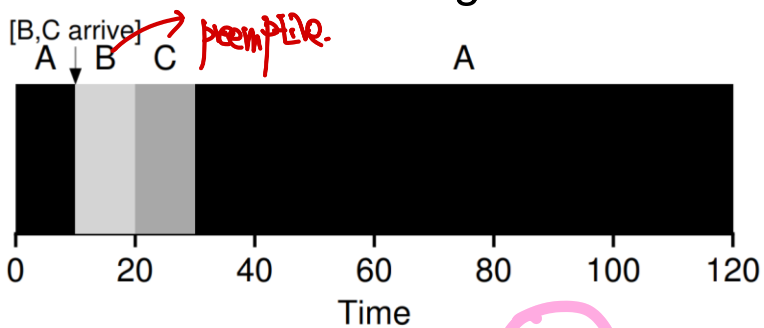
- **SJF is an optimal scheduling if all jobs arrive at the same time**
  - Otherwise, it can also suffer from the same convoy problem as shown in 2)

Shortest Time-to-Completion First (STCF)

- **Shortest Time-to-Completion First (STCF) algorithm adds the preemption on top of SJF**
  - This is also known as Preemptive Shortest Job First (PSJF) scheduler
  - Anytime a new job enters, the STCF scheduler determines which of remaining jobs (including the new job) has the least time left, and schedules that one

(선점형) (비선점형)

- **Preemptive vs Non-preemptive scheduling**
  - Preemptive: can stop a job to run another, require the context switch
  - Non-preemptive: run a job to completion (can't stop a job until finishing)
- **Let's take the same example but with STCF**
  - STCF reduces average turnaround time from 103.3 to 50



preemptive.

$$T_{turnaround} = (\ (120-0)+(20-10)+(30-10)\ )\ /\ 3$$
$$= 50$$

가정하에서 turnaround 관점에서 최적.

- **STCF is provably optimal given our new assumptions**

A New Metric: Response Time

- **The turnaround time is a good metric for early batch systems**
  - However, the introduction of time-shared machines changed all that
  - Now users would sit a terminal and demand interactive performance as well

(batch-system) A B C

- **A new metric was born: response time** → 도착해서 작업을 시작하는 데에 걸리는 시간.
  - We define response time as time from when the job arrives to the first time it scheduled, more formally: $T_{response} = T_{firstrun} - T_{arrival}$

Queue에 들어가는 시간.

- **STCF and related disciplines are not particularly good for response time**
  - Imagine sitting at a terminal, typing, and having to wait 10 seconds to see a response from the system; not too pleasant

- **Then, how can we build a scheduler sensitive to response time?**

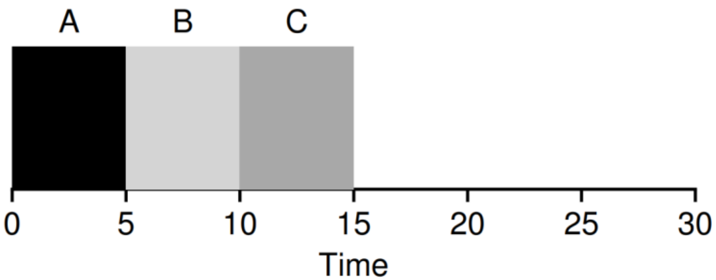
Round Robin (RR)

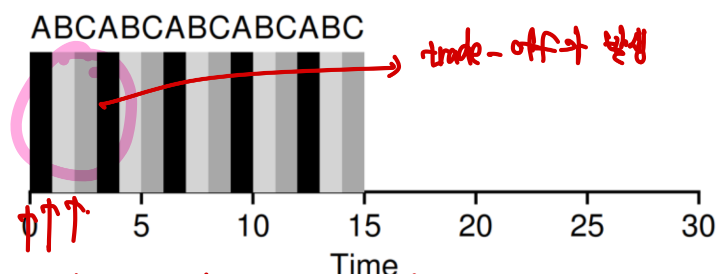
- **To solve this, we introduce Round Robin (RR) scheduling**
  - Instead of running jobs to completion, RR runs a job for a time slice (scheduling quantum, time quantum) and then switches to the next job in the run queue
  - It repeatedly does so until the jobs are finished; called time-slicing
  - The length of a time slice must be a multiple of the timer-interrupt period

context-switching이 너무 많이 일어남.

time-slicing

- **Let's look at an example**
  - Three jobs (A, B, C) arrive at the same time, each runs for 5, time slice is 1





trade-off가 발생

turnaround time

SJF: $T_{response} = (0+5+10)/3 = 5$

RR: $T_{response} = (0+1+2)/3 = 1$

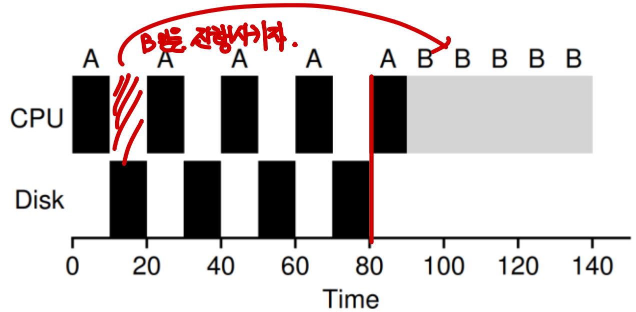
- **As can be seen, the length of the time slice is critical for RR**
  - The short it is, the better the response time; too short → context switch cost ↑
  - Deciding on the length presents a trade-off to a system designer

Incorporating I/O

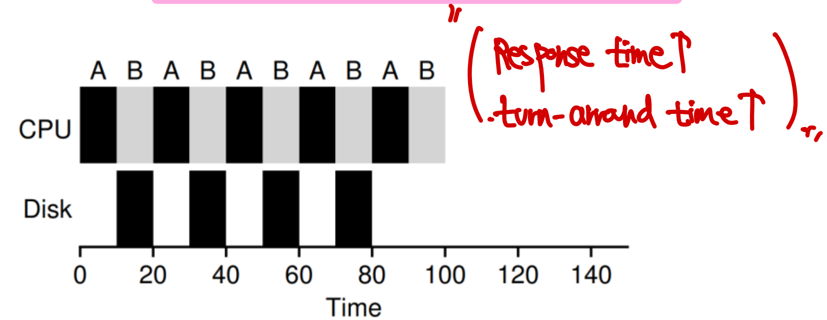
(printf)

- **Obviously, all programs perform I/O and let's relax assumption #4**
  - A scheduler has a decision when a job has an I/O request and I/O complete

- **Let's consider an example**
  - Two jobs (A, B), which each needs 50 ms of CPU time; A runs for 10 ms and then requests I/O (takes 10 ms) and B uses the CPU for 50 ms with no I/O



Waiting for I/O completion:
Poor use of resources



Scheduling for another job:
Better use of resources

- **For better utilizing the processor, the scheduler should make**
  - the CPU-intensive jobs (job B) run while the interactive jobs (job A) are performing I/O