04. CPU Scheduling

▼ workload assumptions

1. 모든 process는 같은 시간 동안만 실행
2. 모든 process가 동시에 실행(process 실행시킨 시점 : PCB 생성 → ready queue에 도착한 때)
3. 실행이 되면 끝날 때까지 cpu 양보 x
4. 모든 process는 cpu 자원만 사용(ex. I/O 없이)
5. 모든 process의 실행 시간을 알 수 있음

⇒ unrealistic

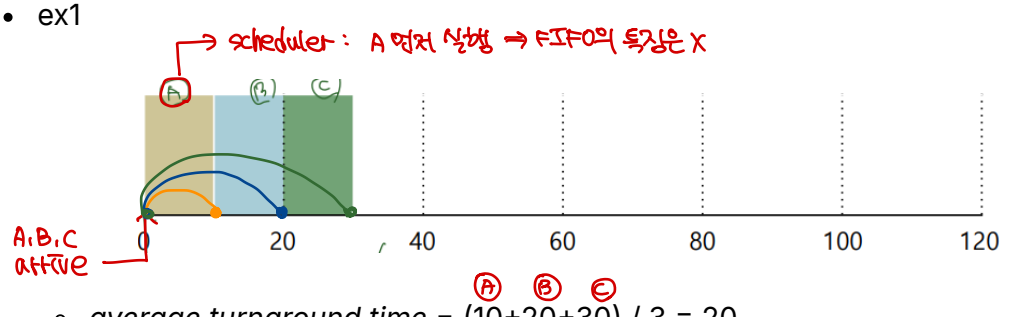
- ***turn around time***

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

▼ FIFO

first come, first served (FCFS) : simple + easy

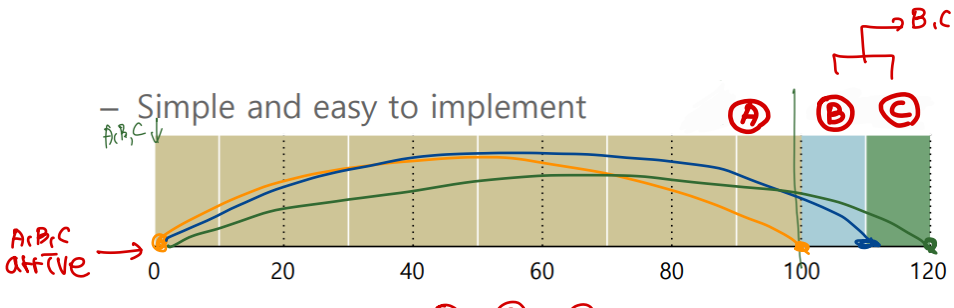
- ex1



  - *average turnaround time* = (10+20+30) / 3 = 20

→ 1번 : 모든 process가 같은 시간 동안만 실행한다는 가정 지우기.

- ex2

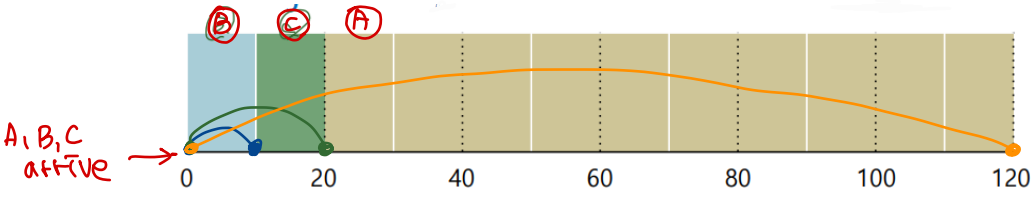


- *average turnaround time* = (100+110+120) / 3 = 110
- 실제 실행 시간은 짧지만 ready state에 있는 시간이 100임

  (turn around 길어짐)

▼ SJF(Shortest Job First)

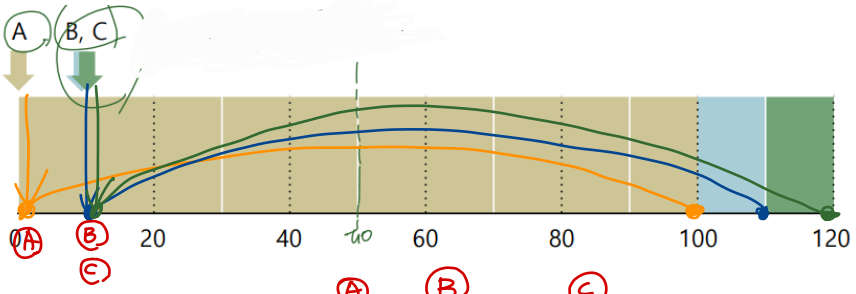
- ex1



  - *average turnaround time* = (10+30 + 120) / 3 = 50
  - 모든 process가 동시에 도착한다면 optimal

→ 2번 : 모든 process가 동시에 도착한다는 가정 지우기.

- ex2



  - *average turnaround time* = (100+(110-10)+(120-10)) / 3 = 103.33
  - 시작하면 끝날 때까지 멈추지 않음

▼ STCF(Shortest Time-to-Completion First) = preemptive SJF(PSJF)

→ 3번 : 한 번 시작되면 실행이 끝날 때까지 CPU 양보하지 않는다는 가정 지우기

- Non-Preemptive Scheduler
  - FIFO, SJF

- 새로운 프로세스가 실행되는지 고려하기 전에 각 프로세스를 실행(SJF) ⇒ 옛날 Batch computing

- Preemptive Scheduler ⇒ OS에서 현재하는 모든 scheduler (STCF, RR)
  - 다른 프로세스를 실행하면 한 프로세스 실행 중지
  - job이 arrive할 때마다 scheduler 실행되어야 함(남은 시간, 실행 시간 비교)

  - scheduling matrix
    - *turnaround time*

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

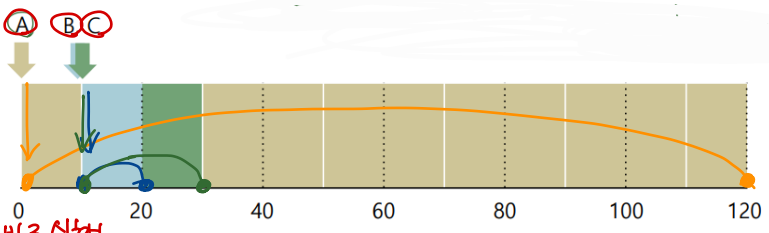
    - *response time* : 시작해서 처음으로 running 상태가 되는 시간

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

⇒ response time이 짧기를 원함. (다른 job이 실행되는데까지)

  - context switch

- ex1 → Job이 arrive할 때마다 scheduler 실행되어야 함. (계속 남은시간, 실행 시간 비교)



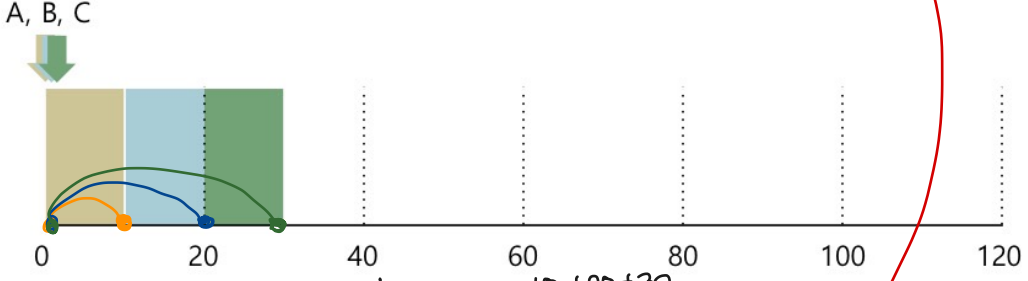
Ⓐ, Ⓑ 바로 실행
Ⓒ 10 < 20 ⇒ 10초 후!

  - *average turnaround time* = (120 + (20-10) + (30-10)) / 3 = 50

⇒ overhead → 줄었지만 0은 아님

  - *average response time* = (0 + 0 + 10) / 3 = 3.33...

- ex2 → good turnaround, bad response and interactivity
  ⇒ Ⓐ : 실행시간 100이라고 가정



  - average turn around time = $\frac{10+20+30}{3} = 20$
  - average response time = $\frac{0+10+20}{3} = 10$ Bad!

▼ Round-Robin(RR)

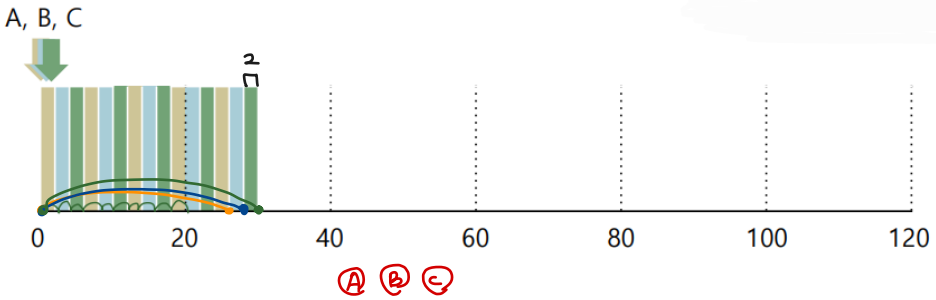
현재의 OS : Round Robin + 이를 실제에 사용.

⇒ response time Good, turn around time Bad.

= Time slicing : scheduling quantum ⇒ Job을 실행

= timer interrupt까지 관여되는 scheduler

- ex1



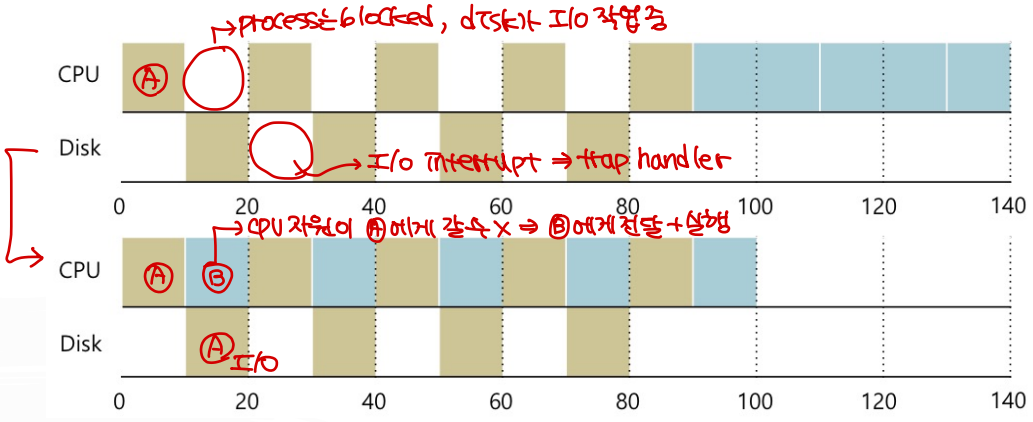
Ⓐ Ⓑ Ⓒ

  - *average response time* = (0+2+4) /3 = 2
  - *turnaround time* → worst! = (16+18+20) /3 = 18
  - time slice = 2 ⇒ 해당 값 설정이 굉장히 어려움.
    - timer의 간격이 time slice와 동일 → 이 때마다 process 돌아가면서 실행
    - time slice 값과 context switch(overhead) 비례함 ⇒ cost trade-off

→ 4번 : 모든 작업은 오로지 CPU만 사용한다는 가정 지우기

- Incorporating I/O(OS가 계속 간섭)
  - r/w system call이 return하기 전에 scheduler 호출
  - process가 넘어가는 과정에도 scheduler 호출 (ex. Ⓐ→Ⓑ)
  - disk : 어떤 process가 요청하는지는 모름 → cpu와 disk 사이의 간섭은 x

Scheduler
: I/O 작업 시작할 때 결정해야 하는 것들
① Job : I/O작업 끝날 때까지 기다렸다가 block
② Scheduler
: I/O 언제 끝날지 결정해야함.



→ 5번 : 모든 processes의 실행 시간을 알수 있다는 가정 지우기

- 우리가 process에 대한 knowledge 없이 SJF/STCF와 같은 방법을 생각할 수 있을까?
  ⇒ 실행시간 예측하면서 어떤 순서로 실행?
  But interactive한 응용 SW 多 ⇒ 그래서 고민한 것 좀 알아보자!
  ⇒ next chapter!