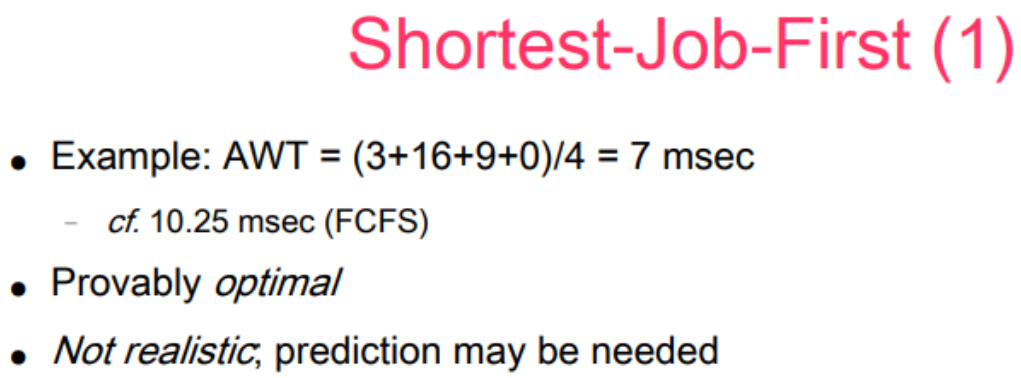
CPU 스케쥴링 알고리즘 (2)

> SJF, Priority, RR 스케쥴링

SJF (Shortest Job First)

`SJF` 는 말 그대로 가장 시간이 적게 걸릴 애를 잡아가지고 걔먼저 처리하는 것이다.

간트 차트를 한번 보자,



Process	Burst Time (msec)
P_1	6
P_2	8
P_3	7
P_4	3

보니까 P_4 가 가장 짧은데? 짧은 순으로 넣게 되면 4 => 1 => 3 => 2 로 넣게 되고,
AWT(평균 대기시간)는 7이 된다. (P_2 입장에서는 자기가 제일 기니까 앞의 3 7 6 다 기다려야해서 16이 됨.)

`Provably Optimal` ? : 증명되어진 최적이다.

다른 모든 예제를 가져 와도 `대기 시간` 의 측면에서는 FCFS보다 항상 나음.

그럼 근데 이 방법의 문제점은 뭐냐? = CPU 시간 어떤 프로세스가 얼마나 사용할지 어떻게 아냐는 문제가 있음. == `얼마나 쓸지 예측할 수 밖에 없는데 이건 비현실적임`

- 예측을 그나마 할 수는 있는데, 어떤 방식이냐면,
- 프로세스는 죽기(Terminate)전까지 큐에 갔다가, CPU 서비스 받다가, 다시 Device Queue 갔다가 다시 Ready Queue 왔다가 하는 방식으로 빙글빙글 돌다가 종료되는데?
- 반복할때마다 이번엔 CPU 시간을 얼마 쓰는가 뭐 이런식으로 일일히 기록해 뒀다가 OS가 조사한 걸 바탕으로 합리적 예측을 할 수는 있겠다만, 그러려면 뭔가 일일히 기록하고 저장해야 한다는 측면에서 Overhead(부담) 발생.

> SJF는 Preemptive / Nonpreemptive 두가지 방식으로 만들어 볼 수 있음.

Shortest-Job-First (2)

- Preemptive or Nonpreemtive
  - *cf.* Shortest-Remaining-Time-First (최소잔여시간 우선)
- Example
  - Preemptive: AWT = (9+0+15+2)/4 = 26/4 = 6.5 msec
  - Nonpreemptive: 7.75 msec

Process	Arrival Time	Burst Time (msec)
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

1 Nonpreemptive 방식은 현재 프로세스가 다 끝나고 난 다음에야 다음게 실행 되는거.

간트 차트를 보면, 위의 차트랑 다른건 위의 차트에서는 (SJF의 경우)그냥 도착시간이 똑같은 상황을 상정한거고, 여기 차트에서는 도착 시간이 다름.

우선, P_1 은 큐에 들어와보니 지밖에 없음(물론 나중에 미래 상황까지 상정해 보면 Burst Time이 더 적은 프로세스들이 존재하기는 하지만?) 그래서 실행 바로 됨.

P_1이 실행되고 1미리초가 지나면 P_2 가 들어오고, 뭐 이런식으로 진행될거임.

애초에 근데 `P₁ 이 8미리초나 하기 때문에` 얘가 한 중간쯤 실행될 시점엔 나머지 애들도 다 큐에 들어와서 기다리고 있음. 그러면 이제 큐에 P2 , P3, P4 가 들어와 있기 때문에 얘들 중 Burst Time이 짧은 애를 선택해가지고 그다음을 실행하고 이런 식임.

그러면 AWT = 7.75 라는건 어떻게 나왔냐? 우선 P1 -> P2 -> P4 -> P3 순으로 끝날거.

(1) 기본적으로 P1 이 젤 먼저 실행돼서 얜 기다린 시간이 없으니까 0 이고,

(2) P2 는 그다음 간택될건데(2 3 4 중에 제일 짧아서) 들어온 시간이 1이고, 앞에 실행이 되던 P1 이 다 끝나야만(Nonpreemptive) P2가 실행될거니까 7초 후 실행,

(3) P4 는 17-2 = 15

(4) P3 는 12-3 = 9

그래서 0 + 7 + 15 + 9 / 4 = 7.75

2 Preemptive 방식은 응급실 같은건데, 지금 진료중 환자(현재 실행중 프로세스)가 있어도 더 응급한 애가 있으면 지금 하고있는애보다 우선해서 진료(프로세스 처리) 가능한 거.

==> `Shortest Remaining Time First` 라고 부르기도 함.

이 방식은 A 프로세스가 실행되고 있는 상황에서 B 프로세스가 큐에 들어왔다면,
A 프로세스의 잔여 실행 시간과 B 프로세스의 전체적인 Burst TIme과 비교해 더 짧은애를 일단 실행시키는 방식으로 이루어짐.

대기실에 박혀있는 시간 다 합해보면 => 6.5 가 나온다는 뜻.

Priority

우선순위 : 누가 더 우선인 것인가?

일반적으로 우선순위를 정수값으로 나타냄. (숫자가 작으면 우선순위가 높다)

Priority Scheduling (1)

- Priority (우선순위): typically an integer number
  - Low number represents high priority in general (Unix/Linux)
- Example
  - AWT = 8.2 msec

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

간트 차트를 보면, Time 0 에 5개가 줄서서 기다리고 있는데? 그냥 우선순위대로 넣음.

P2 가 가장 높은데? 얘부터 들어감.

== 6 + 0 + 16 + 18 + 1 / 5 = 8.2

Priority Scheduling (2)

- Priority
  - Internal: time limit, memory requirement, i/o to CPU burst, …
  - External: amount of funds being paid, political factors, …
- Preemptive or Nonpreemptive
- Problem
  - Indefinite blocking: *starvation* (기아)
  - Solution: againg

- 우선순위는 그럼 어떻게 정하나? 내부적 요소 vs 외부적 요소
  - 내부적? : 타임 리미트 (짧은애를 먼저 해라)
  - memory requirement (메모리 적게 차지하는애 먼저 해라)..etc

    - 외부적? : 어느 학과가 서비스 이용하는데 돈 많이 내냐..? 뭐 이런 외부적 요인들이 기준이 될 수 있음.
- 기아? 굶어 죽는건데.. 어떤 우선순위 낮은 애는? 이 5개 말고도 막 새로 굴러들어오는 애가 기다리는애보다 우선순위 높으면 얜 영영 기다려야함.
- 에이징 : 오래 기다리면 우선순위 좀 올려주는 방식으로 해결 가능.

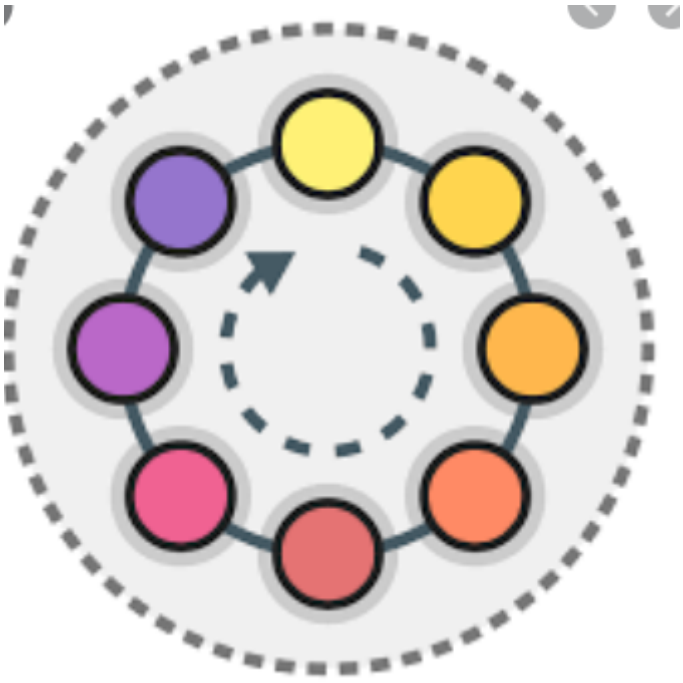
Round Robin

Round-Robin (1)

- Time-sharing system (시분할/시공유 시스템)
- Time *quantum* 시간양자 = time *slice* (10 ~ 100msec)
- Preemptive scheduling
- Example
  - Time Quantum = 4msec
  - AWT = 17/3 = 5.66 msec

Process	Burst Time (msec)
P_1	24
P_2	3
P_3	3

> 뱅글 뱅글 돌면서 스케쥴링



Time Sharing System 에서 많이 사용되는데,

P1 P2 P3 / P1 P2 P3 / P1 P2 P3 => 이런 식으로 하니까 당연히 preemptive 가 됨.

타임 퀀텀을 두고 (4 밀리초라고 할때?) => 걍 스위치 시간이 4 밀리초라고 생각.

P1 이 실행되고, 4밀리초 지나면 스위칭 될건데, 4밀리초 꼭 다 쓸필요는 없고 남은 Burst Time 이 끝나면 스위칭 됨.

뭐 타임퀀텀 (델타)를 짧게 바꾸면 AWT같은 척도가 달라지게 될 것.

=> 그래서 `타임 퀀텀`에 굉장히 의존적이다!!

- 만약 타임 퀀텀을 무한대로 둔다면??? => 스위칭이 안됨 프로세스 끝날때까지.

=> 그래서 그냥 선착순 FCFS랑 똑같게 됨.

- 타임 퀀텀을 0으로 두면? 프로세스 쉐어링 이라고 하는데 스위칭이 하도 빈번해서 3개가 한꺼번에 도는 것처럼 보이게 됨. 0으로 수렴시키면 근데 context switching overhead 발생!!
- 너무 빈번하게 스위치 하면 dispatcher가 PCB 값 바꾸고 하는 역할을 하는데 이게 너무 자주 발생해서 부담이 증가하게 된다는 뜻.

아래 예제는 타임 퀀텀에 따라 성능이 달라지는다는 것을 계산해 둔 표.

Round-Robin (2)

- Performance depends on the size of the time quantum
  - $\Delta \to \infty$ FCFS
  - $\Delta \to 0$ Processor sharing (* context switching overhead)
- Example: *Average turnaround time (ATT)*
  - ATT = 11.0 msec (Δ = 1), 12.25 msec (Δ = 5)

Process	Burst Time (msec)
P_1	0
P_2	3
P_3	1
P_4	7