Scheduling

태그	OS
주차	4주차

📌 목차

1. Scheduling

- 여러 프로세스가 번갈아가며 사용하는 자원을 어떤 시점에 어떤 프로세스에게 자원을 할당할 지 결정하는 것
- 시스템 성능에 직/간접적 영향을 미침
- 좋은 스케줄링은 프로세서의 효율을 높이고, 응답 시간을 최소화하여 시스템의 작업 처리 능력을 향상

1-1. 알고리즘 종류

1) 선점 스케줄링

- 프로세스 하나가 장시간동안 프로세서를 독점하는 것을 방지하기 위함
    - 장점
        - 우선순위가 높은 프로세스가 긴급 처리돼야 할 때 유용

        - 모든 프로세스에 프로세서를 할당해 줄 수 있는 기회가 생김
    - 단점
        - 잦은 프로세스 교환으로 인해(컨텍스트 스위칭) 오버헤드가 높음

2) 비선점 스케줄링

- 한 프로세스가 프로세서(CPU, 자원)을 선택했을 때 다른 프로세스가 해당 프로세서를 빼앗을 수 없도록 한 것
    - 장점
        - 실행시간이 짧은 프로세스가 먼저 자원을 가질 수 있어 가장 짧은 평균 대기 시간을 기대할 수 있음
    - 단점
        - 실행시간이 긴 프로세스는 실행시간이 짧은 프로세스에 밀려 Starvation이 올 수 있음

> 💡 **기아 현상 (Starvation)**
>
> 프로세스가 원하는 자원을 계속 할당받지 못하는 상태

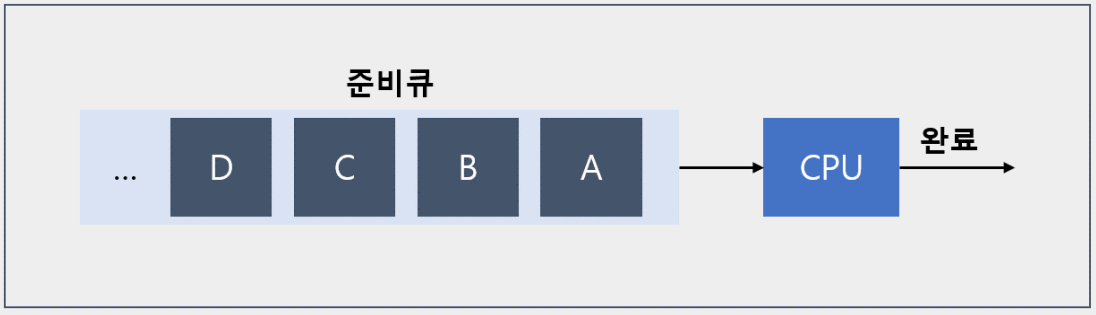
1-2. 스케줄링 목적

1. 자원 할당의 공정성
    - 모든 프로세스는 공평하게 다뤄져야 하고 어떤 프로세스도 실행이 무제한 연기(starvation)되면 안된다.
2. 단위시간당 처리량 극대화
    - 프로세서(CPU)가 최대한 쉬지 않고(유효시간 없이) 일을 하도록 하는 게 스케줄러의 목적이다.
3. 적절한 반환시간 보장
    - 어떤 input이 일어나면 적절한 시간 내에는 프로세스가 output을 낼 수 있도록 스케줄링 해야 한다.
4. 예측 가능성 보장
    - 언제 어떻게 실행해도 프로세스는 거의 비슷한 시간에 비슷한 비용으로 처리 되어야 한다.

5. 오버헤드 최소화
    - 오버헤드가 발생(ex. while(1){})하면 CPU 자원의 낭비가 심하므로 오버헤드를 줄여야 한다.
6. 자원 사용의 균형 유지
    - 최대한 프로세서 및 자원을 쉬지 않고 동작시키는 게 목표이니 유휴 상태의 자원을 사용하려는 프로세스에 우선순위를 줄 수 있다. (ex. 프린터)
7. 실행 대기 방지
    - 실행을 무한하게 연기하지 않도록 해야 한다
    - 즉, 우선순위를 적절히 주어 프로세스가 starvation 상태가 일어나지 않도록 해야 한다
    - 이는 aging 방법으로 해결할 수 있다. (대기중인 프로세스가 우선순위를 차츰 높이는 방식)
8. 우선순위
    - 모든 프로세스에 우선순위를 부여하여 스케줄링 알고리즘을 통해 우선순위가 높은 프로세스부터 먼저 실행하도록 한다

1-3. 알고리즘

1) FCFS (First Come First Served)

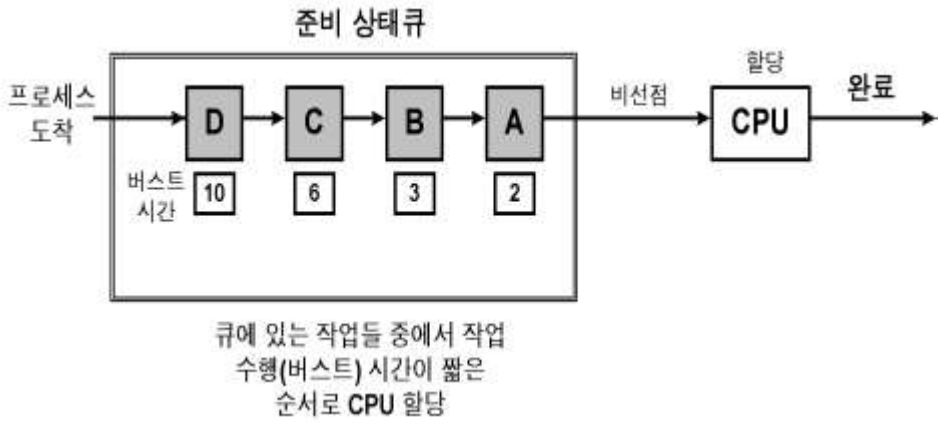


FCFS 스케줄링 알고리즘

- 비선점 스케줄링
- 이미 할당된 CPU를 다른 프로세스가 강제로 빼앗아 사용할 수 없는 스케줄링 기법
- 짧은 작업이 긴 작업을 기다리는 경우가 발생할 수 있음

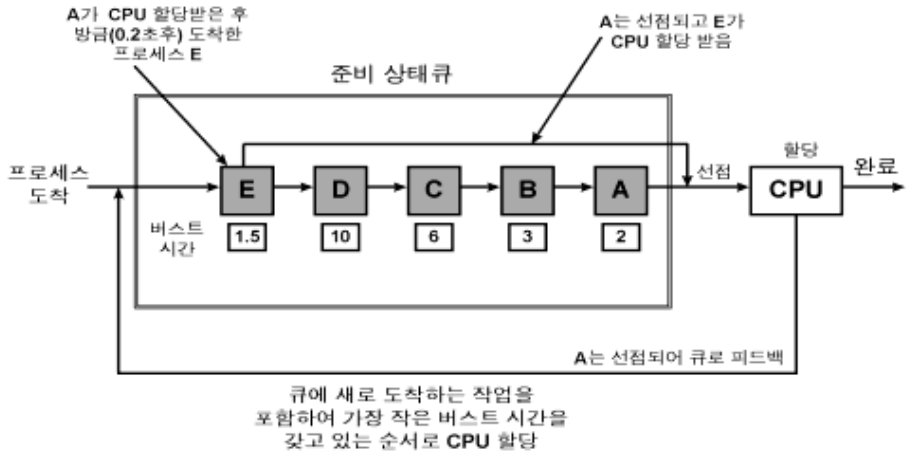
- 프로세스 응답 시간의 예측이 용이하며, **일괄 처리 방식 (Batch Processing)** 에 적합
    - Batch Processing : 여러 개의 프로그램을 읽어놓고, 한 번에 하나의 프로그램만 실행
- 먼저 도착한 프로세스가 먼저 CPU를 선점
- 단점
    - Convoy Effect: 실행시간이 짧은 프로세스들이 실행시간이 긴 프로세스를 계속해서 기다리며 효율성이 저하됨

2) SJF (Shortest Jop First)



- **비선점형 스케줄링**
- 실행시간(Burst Time)이 가장 짧은 프로세스부터 실행
- 이미 긴 프로세스가 실행중이라면 새로 도착한 짧은 프로세스는 기다리긴 해야 함
- 단점
    - Starvation: 실행시간이 긴 프로세스가 영원히 CPU를 할당받을 수 없게됨

3) SRTF (Shortest Remaining Time First), SRT



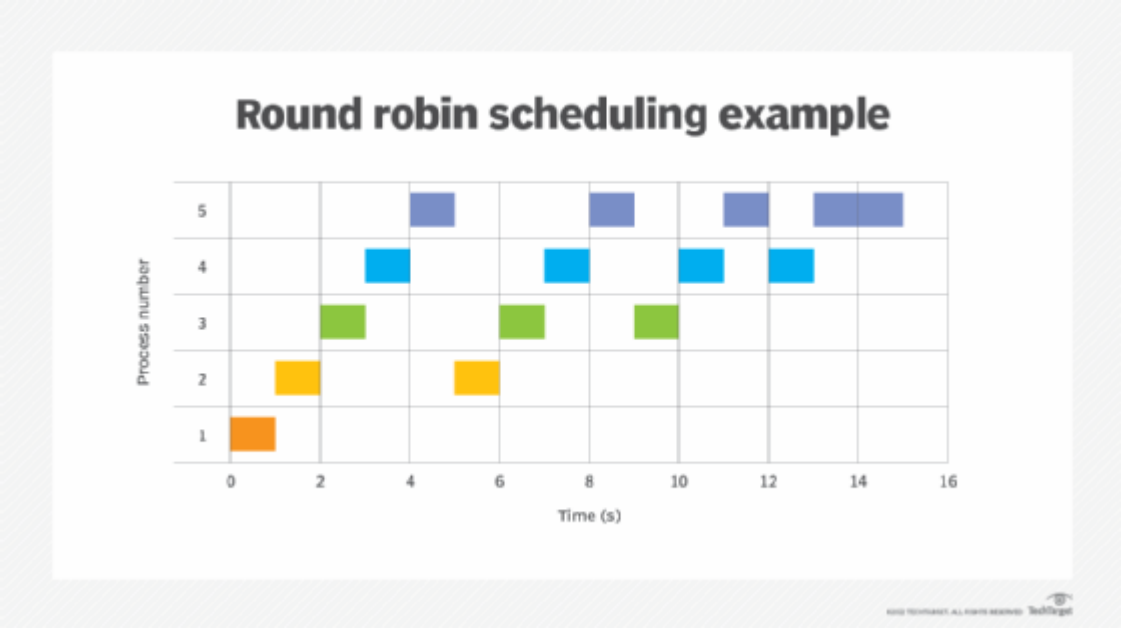
- **선점형 스케줄링**
- 현재 실행 중인 프로세스보다 우선순위가 더 높은 프로세스가 도착하면 cpu를 뺏김
- 새로운 프로세스가 도착할 때마다 새롭게 스케줄링을 진행
- 새로운 프로세스가 들어온 시점에서 현재까지 남은 실행시간이 가장 적은 프로세스를 먼저 실행
    - 새로 들어온 가장 짧은 프로세스가 오자마자 실행될 수 있음
- 단점
    - Starvation : SJF 와 같은 이유로 실행시간이 긴 프로세스가 영원히 CPU 할당받을 수 없음
    - 새로운 프로세스가 올 때마다 스케줄링을 다시하므로, 프로세스의 정확한 CPU Burst Time을 측정할 수 없음

4) Priority Scheduling

- **선점, 비선점 스케줄링 방식 모두 사용 가능**
    - 선점형 : 더 높은 우선순위의 프로세스가 도착하면 현재 실행중인 프로세스에게서 CPU를 뺏음
    - 비선점형 : 더 높은 우선순위의 프로세스가 도착하면 Ready Queue에 넣고 바로 다음에 실행되게 함
- 우선순위가 높은 프로세스가 CPU를 선점하도록 하는 스케줄링
    - 우선순위는 숫자가 작을수록 높음
- 단점

    - Starvation : 우선순위가 낮은 프로세스는 계속해서 우선순위가 높은 프로세스에게 밀려 실행될 수 없음
    - **무기한 봉쇄(Indefinite blocking)** : 우선순위가 높은 프로세스가 Blocking 되어 있어 CPU가 계속해서 대기해야하는 상황
- 해결
    - **Aging:** 프로세스의 대기 시간이 증가함에 따라 우선순위를 높여주는 기법. Starvation 예방 가능

5) RR (Round Robin)



- 각 프로세스는 동일한 할당 시간 **(Time Quantum)** 을 가짐
- CPU를 할당 받고 할당 시간이 지나면 Ready 상태로 돌아가 Ready Queue의 Tail로 들어감
- 프로세스들이 작업을 완료할 때까지 계속해서 순회
- 장점
    - `Response time`이 빨라짐
        - n 개의 프로세스가 ready queue 에 있고 할당시간이 q(time quantum)인 경우 각 프로세스는 q 단위로 CPU 시간의 1/n 을 얻음
        - 즉, 어떤 프로세스도 (n-1)q time unit 이상 기다리지 않음
    - 모든 프로세스가 공정하게 CPU를 할당받을 수 있음을 보장

- 주의할 점
  - 설정한 `time quantum`이 너무 커지면 `FCFS`와 같아짐
  - 너무 작아지면 Context Switching으로 인한 Overhead가 증가함

정리) 선점형 VS 비선점형

- 선점형
  - RR, SRT
- 비선점형
  - FCFS, SJF
- 둘 다 가능
  - Priority