CPU Scheduling 1

상태	완료
생성 일시	@2023년 7월 7일 오전 1:14
완성일	@2023년 7월 11일
유형	정리

프로그램이 실행되면 아래와 같은 구조로 실행이 됨

CPU를 잡고 수행하는 단계, 입출력 작업을 진행하는 단계

CPU만 쭉 잡고 실행하는 프로그램도 있겠지만 통상적으로 두 단계를 왔다 갔다함

CPU burst와 I/O burst

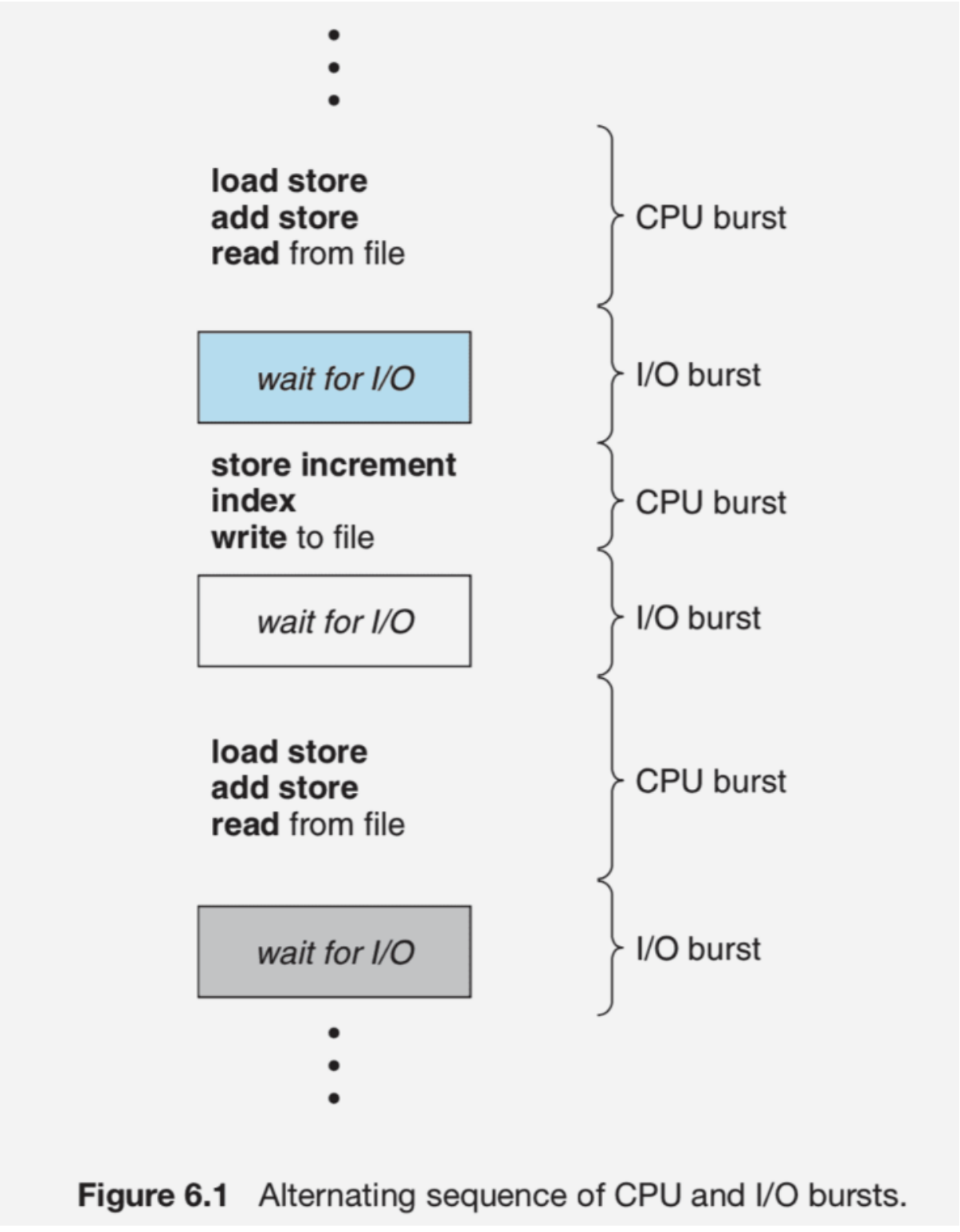


Figure 6.1 Alternating sequence of CPU and I/O bursts.

CPU burst

CPU 버스트는 사용자 프로그램이 CPU를 직접 가지고 빠른 명령을 수행하는 단계이다. 이 단계에서 사용자 프로그램은 CPU 내에서 일어나는 명령 (ex. Add)이나 메모리(ex. Store, Load)에 접근하는 일반 명령을 사용할 수 있다.

I/O burst

I/O 버스트는 커널에 의해 입출력 작업을 진행하는 비교적 느린 단계이다. 이 단계에서는 모든 입출력 명령을 특권 명령으로 규정하여 사용자 프로그램이 직접 수행할 수 없도록 하고, 대신 운영체제를 통해 서비스를 대행하도록 한다.

이처럼 사용자 프로그램이 수행되는 과정은 CPU 작업과 I/O 작업의 반복으로 구성된다.

CPU bound process와 I/O bound process

각 프로그램마다 CPU 버스트와 I/O 버스트가 차지하는 비율이 균일하지 않다. 어떤 프로세스는 I/O 버스트가 빈번해 CPU 버스트가 매우 짧은 반면, 어떤 프로세스는 I/O를 거의 하지 않아 CPU 버스트가 매우 길게 나타난다. 이와 같은 기준에서 프로세스를 크게 I/O 바운드 프로세스와 CPU 바운드 프로세스로 나눌 수 있다.

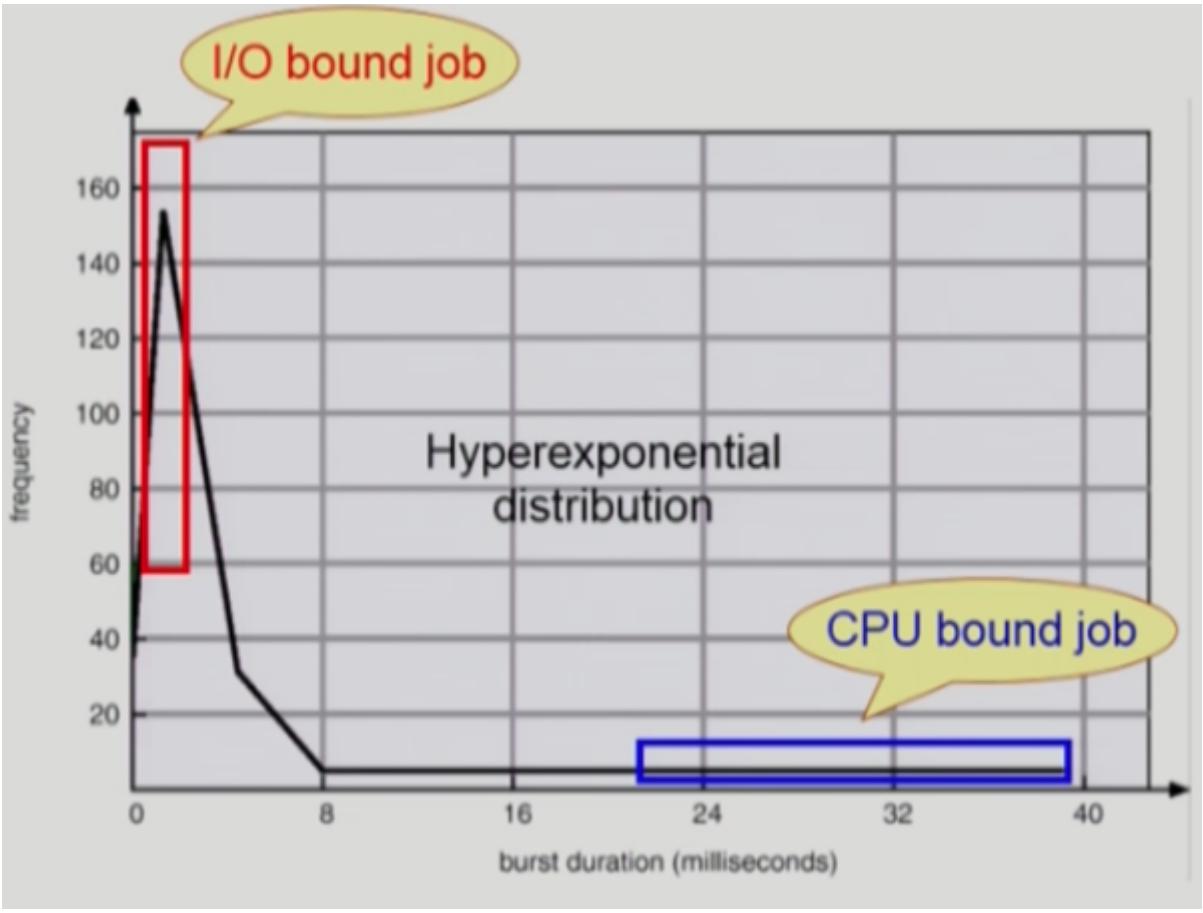
I/O 바운드 프로세스

I/O 요청이 빈번해 CPU 버스트가 짧게 나타나는 프로세스를 말한다. 주로 사용자로부터 인터랙션을 계속 받아가며 프로그램을 수행하는 대화형 프로그램이 해당된다.

CPU 바운드 프로세스

I/O 작업을 거의 수행하지 않아 CPU 버스트가 길게 나타나는 프로세스를 말한다. 주로 프로세스 수행의 상당 시간을 입출력 작업 없이 CPU 작업에 소모하는 계산 위주의 프로그램이 해당된다.

CPU-burst Time의 분포



프로그램이 수행되는 구조를 보면 I/O 바운드 프로세스는 짧은 CPU 버스트를 많이 가지고 있지만, CPU 바운드 프로세스는 소수의 긴 CPU 버스트로 구성된다는 것을 알 수 있다.

즉 빈도를 살펴본 결과 I/O가 중간에 끼어드는 경우가 매우 많은 것을 확인할 수 있음. CPU만 진득하게 사용하는 경우는 굉장히 낮았음, 이런 식으로 CPU는 짧게 쓰고 중간에 I/O가 끼어드는 이런 종류의 작업을 I/O bound job이라고 부르며 CPU만 굉장히 오랫동안 쓰고 이런 프로그램을 CPU bound job이다. 라고 함

** 여러 종류의 job(=process)이 섞여 있기 때문에 CPU 스케줄링이 필요하다.

- interactive(상호작용을 하는) job에게 적절한 response 제공 요망
- CPU와 I/O 장치 등 시스템 자원을 골고루 효율적으로 사용

이 표를 보고 대부분의 CPU 시간은 I/O bound job이 다 쓰는구나! 라고 생각하면 안 됨

I/O bound job은 I/O가 너무 많이 끼어들어서 CPU가 난도질당해 빈도가 높았던 것이고

반면의 CPU bound job은 CPU를 굉장히 오랫동안 쓰기 때문에 빈도는 적을 수밖에 없는 것임

그래서 이 그림을 보고 CPU는 I/O bound job이 많이 쓰는구나! 라고 생각하면 안 됨 실제로 CPU는 CPU bound job이 많이 사용함 I/O bound job은 CPU를 짧게 쓰는데 빈도가 잦은 것이다 요 정도로 해석하는 게 타당하다 결론적으로는 job의 종류가 섞여 있다.

CPU 스케줄링

컴퓨터 시스템 내에서 수행되는 프로세스의 CPU 버스트를 분석해보면 대부분의 경우 짧은 CPU 버스트를 가지며, 극히 일부분만 긴 CPU 버스트를 갖는다. 이는 다시 말해서 CPU를 한 번에 오래 사용하기보다는 잠깐 사용하고 I/O 작업을 수행하는 프로세스가 많다는 것이다. 즉 대화형 작업을 많이 수행해야 하는데, 사용자에 대한 빠른 응답을 위해서는 해당 프로세스에게 우선적으로 CPU를 할당하는 것이 바람직하다. 만약 CPU 바운드 프로세스에게 먼저 CPU를 할당한다면 그 프로세스가 CPU를 다 사용할 때까지 수많은 I/O 바운드 프로세스는 기다려야할 것이다. 이러한 이유로 CPU 스케줄링이 필요해졌다.

요약하자면 I/O 바운드 프로세스는 CPU 바운드 프로세스처럼 CPU를 한번 잡으면 길게 사용하는 프로세스가 있을 때 CPU를 못 잡고 기다려야 하는 상황이 발생하고 그 결과로 사용자가 답답함을 느낄 수 있기 때문에 CPU 스케줄링이 필요한 것이다.

CPU 스케줄러

CPU 스케줄러는 준비 상태에 있는 프로세스들 중 어떠한 프로세스에게 CPU를 할당할 지 결정하는운영체제의 코드이다.

CPU 스케줄러가 필요한 경우

- Running → Blocked (ex. I/O 요청하는 시스템 콜)
- Running → Running (ex. 할당 시간 만료로 인한 타이머 인터럽트)
- Blocked → Ready (ex. I/O 완료 후 인터럽트(디바이스 컨트롤러가 인터럽트를 걸어서 CPU를 얻을 수 있는 권한을 주는 것임 Ready로 만들어서))
- Terminated(ex. 종료 됨)

1, 4번째의 스케줄링은 강제로 빼앗지 않고 자진 반납하는 `nonpreemptive` 방식이고, 그 외의 스케줄링은 CPU를 강제로 빼앗는 `preemptive` 방식이다. 전자를 비선점형 스케줄링, 후자를

선점형 스케줄링이라고 부른다. 참고로 3번째 스케줄링은 Context Switch가 발생할 때만 해당한다.

용어정리

`nonpreemptive` = 강제로 빼앗지 않고 자진 반납

`preemptive` = 강제로 빼앗음

디스패처 (Dispatcher)

CPU를 누구한테 누구한테 줄 지 결정했으면 해당 프로세스에게 넘겨야 하는데, 디스패처가 이 역할을 수행한다. 이 과정이 Context Switch에 해당한다. 디스패처도 운영체제의 코드이다.

- CPU 스케줄러는 누구한테 CPU를 줄지 결정하는 거고 디스패처는 실제로 CPU를 주는 코드임
- 문맥 교환은 디스패처의 역할

스케줄링의 성능 평가 (Scheduling Criteria)

어떤 스케줄링이 좋은지 평가하는 방법을 말함, 크게 두 가지로 나눌 수 있음

시스템 입장

- CPU utilization (이용률)
  - 전체 시간 중에서 CPU가 일을 한 시간의 비율(놀지않고)
  - keep the CPU as busy as possible(시스템 입장에서는 CPU가 가능한 일을 많이 하는게 좋은 것임)
- Throughput (처리량)
  - 주어진 시간동안 준비 큐에서 기다리고 있는 프로세스 중 몇 개를 끝마쳤는지 나타낸다.
  - 즉 CPU의 서비스를 원하는 프로세스 중 몇 개가 원하는 만큼의 CPU를 사용하고 이번 CPU 버스트를 끝내어 준비 큐를 떠났지 측정한 것이다.

    - 더 많은 프로세스들이 CPU 작업을 완료하기 위해서는 CPU 버스트가 짧은 프로세스에게 우선적으로 CPU를 할당하는 것이 유리하다.
    - number of processes that complete their execution per time until

프로세스 입장

CPU를 이용하는 고객 입장에서는 빨리 끝나면 좋은 것임 그래서 시간과 관련된 개념으로 이루어짐

- Turnaround time (소요 시간, 반환 시간)
    - 프로세스가 CPU를 요청한 시점부터 자신이 원하는 만큼 CPU를 다 쓰고 CPU 버스트가 끝날 때까지 걸린 시간을 뜻한다.
    - (준비 큐에서 기다린 시간) + (실제로 CPU를 사용한 시간)
    - amount of time to execute a particular process
- Waiting time (대기 시간)
    - CPU 버스트 기간 중 프로세스가 준비 큐에서 CPU를 얻기 위해 기다린 시간의 (총)합을 뜻한다.
    - 시분할 시스템의 경우, 한 번의 CPU 버스트 중에도 준비 큐에서 기다린 시간이 여러 번 발생할 수 있다.
    - 이때 대기 시간은 CPU 버스트가 끝나기까지 준비 큐에서 기다린 시간의 합을 뜻하게 된다.
    - amount of time a process has been waiting in the ready queue
- Response time (응답 시간)
    - 프로세스가 준비 큐에 들어온 후 첫 번째 CPU를 획득하기까지 기다린 시간(첫번째 반응)을 뜻한다.
    - 응답 시간은 대화형 시스템에 적합한 성능 척도로서, 사용자 입장에서 가장 중요한 성능 척도라고 할 수 있다.

여기서 이야기하는 건 프로세스가 시작해서 종료되는 그 시간을 말하는 게 아님 CPU를 사용하는 burst를 이야기하는 것이라 프로세스별 각각의 CPU 버스트를 따지는 것임 그래서 반환 시간을 보자면 프로세스가 시작해서 종료되는 시간이 아니고 하나의 프로세스가 CPU를 쓰러 들어와서 I/O를 하러 나갈 때까지 걸린 그 시간을 의미함

생활 속에서의 비유를 통한 스케줄링의 성능 평가 이해

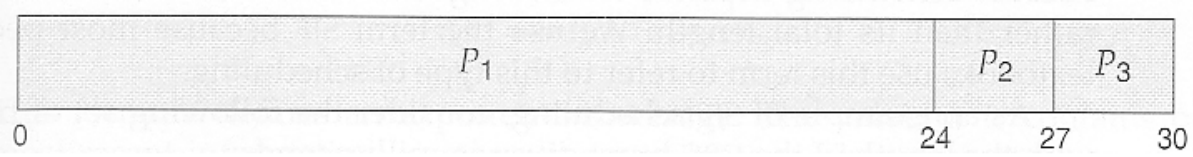
- 중국집에 주방장과 손님이 있다.
- 이용률, 처리률 → 중국집 입장에서의 척도
  - 이용률: 전체 시간 중 주방장이 일한 시간의 비율
  - 처리률: 주방장이 정해진 시간 동안 요리를 만들어 준 손님의 수
- 소요 시간, 대기 시간, 응답 시간 → 손님 입장에서의 척도
  - 소요 시간: 손님이 중국집에 들어와서 주문한 음식을 다 먹고 나가기까지 소요된 총 시간
  - 대기 시간: 음식을 먹은 시간을 제외한 순수하게 기다린 시간 (음식이 조금씩 여러 번 나왔더라도 중간 중간 기다린 시간을 다 합쳐야 함 ex 코스요리)
  - 응답 시간: 최초의 음식이 나오기까지 기다린 시간

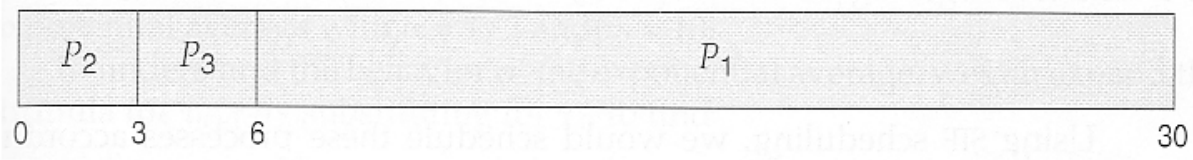
스케줄링 알고리즘 (Scheduling Algorithms)

비선점형 스케줄링(CPU를 강제로 빼앗지 않음), 선점형 스케줄링(CPU를 강제로 빼앗음)

선입 선출 스케줄링 (FCFS, First-Come First-Served)

- 먼저 온 순서대로 처리하는 방식 (`nonpreemptive`) = (비선점)
- CPU를 오래 쓰는 프로세스가 먼저 와서 CPU를 할당 받으면, 나머지 프로세스들은 전부 기다려야하므로 효율적이지 않음





- 위 사진처럼 어떤 프로세스가 먼저 실행되느냐에 따라 전체 대기 시간에 상당한 영향을 미침

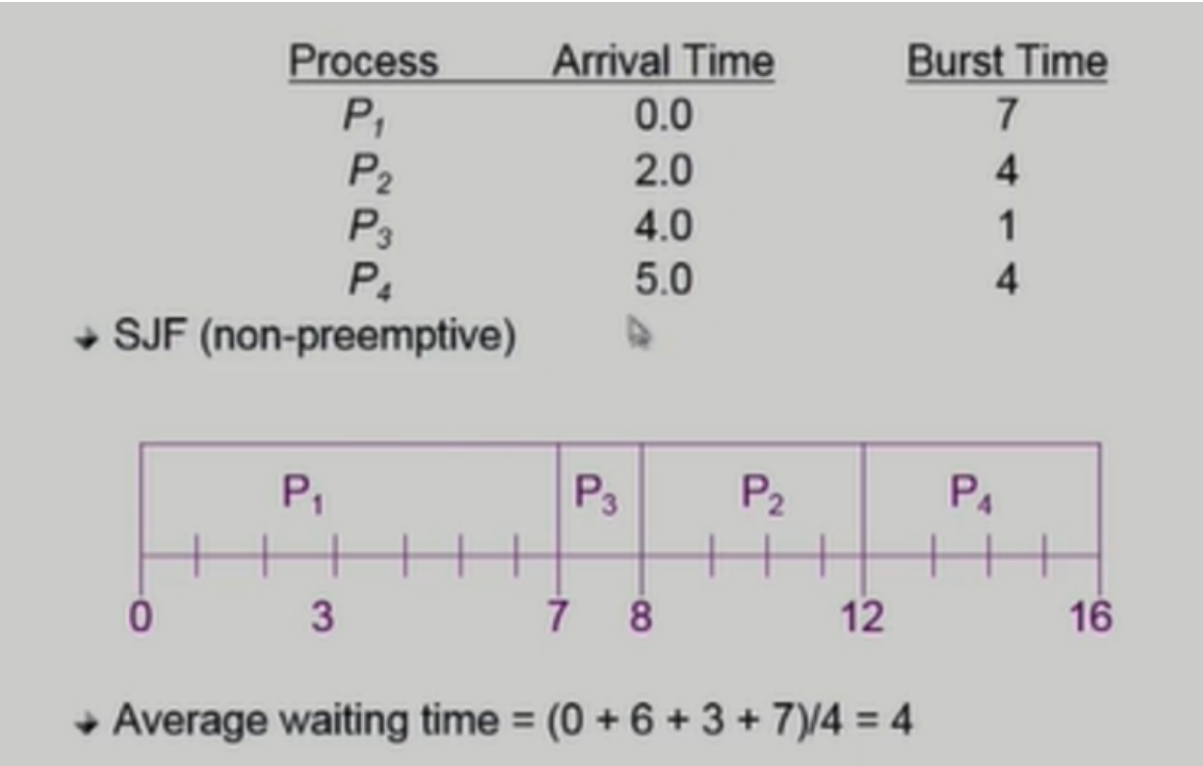
    - 1번
        - 대기 시간: P1 = 0, P2 = 24, P3 = 27
        - 평균 대기 시간: (0 + 24 + 27) / 3 = 17
    - 2번
        - 대기 시간: P1 = 6, P2 = 0, P3 = 3
        - 평균 대기 시간: (6 + 0 + 3) / 3 = 3
- 긴 프로세스 하나 때문에 짧은 프로세스(CPU를 짧게 쓰는) 여러 개가 기다리는 현상을 `Convoy effect` 라 부름
    - FCFS는 앞에 어떤 프로세스가 버티고 있냐에 따라 기다리는 시간의 상당한 영향을 줌
    - ex 중국집에서 먼저 온 사람이 5시간 걸리는 요리 시키면 뒤에 자장면 먹으러 온 사람은 기다려야 함

최단 작업 우선 스케줄링 (SJF, Shortest-Job-First)

- SJF는 CPU 버스트가 가장 짧은 프로세스에게 제일 먼저 CPU를 할당하는 방식
- 평균 대기 시간을 가장 짧게 하는 최적 알고리즘
- 두 가지 방식
    - nonpreemptive (비선점형)
        - 일단 CPU를 잡으면 더 짧은 프로세스가 들어와도 CPU 버스트가 완료될 때까지 CPU를 선점당하지 않음.(일단은 보장해줌)
    - preemptive (선점형)
        - SRTF (Shortest-Remaining-Time-First)
        - CPU를 잡았다 하더라도 더 짧은 프로세스가 들어오면 CPU를 빼앗김.
        - nonpreemptive보다 더 평균 대기 시간을 가장 짧게 하는 최적 알고리즘
            - 더 짧은 프로세스가 들어오면 CPU를 빼앗길 때 전체적인 대기 시간이 짧아짐
- 문제점
    - Starvation (기아): 짧은 프로세스로 인해 긴 프로세스가 영원히 CPU를 잡지 못할 수 있음
    - CPU 버스트 시간을 미리 알 수 없음: 과거 CPU 사용 시간을 통해 추정만 가능

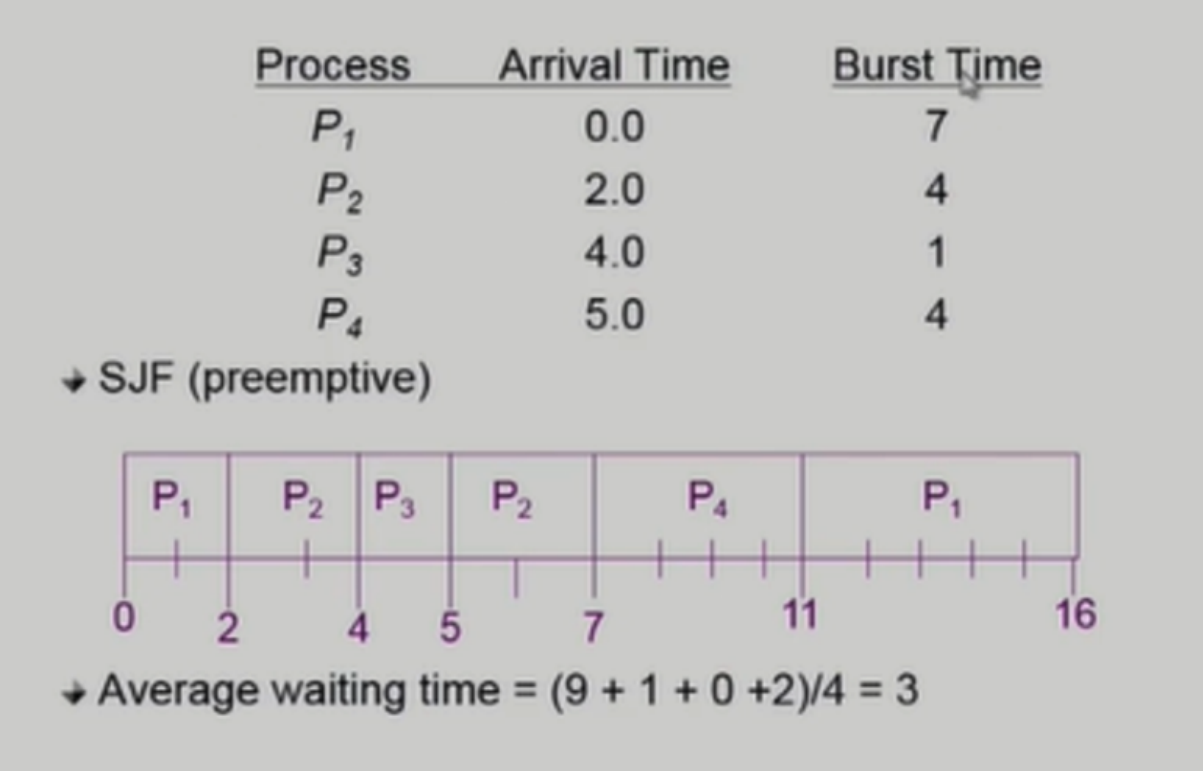
- 매번 CPU를 사용하려고 들어와서 내가 이번에 얼마나 쓰고 나갈지를 알 수가 없음
- 그래서 실제 시스템에서 SJF는 추정(과거 CPU 사용 시간을 통해)을 통해 구현함

비선점형 방식 예시



p1이 7초 동안 CPU를 사용할 때 다 도착해 있고, 그중 P3가 CPU 버스트가 가장 짧음

선점형 방식 예시



p1 먼저 사용하다가 2초 뒤에 버스트 타임 4인 p2가 들어옴 그러면 p1은 5초 더 처리해야 하지만 뒤로 밀림

p2는 2초 사용하다가 갑자기(4초 시점에) p3가 버스트 타임이 더 짧아 뒤로 밀림

5초 시점에는 p4까지 도착한 시점인데 남은 시간을 비교했을 때 p1 5초, p2 2초, p4 4초라 해당 순서에 맞게 처리됨

평균 대기시간이 3초가 나왔는데 preemptive SJF는 평균 대기 시간을 가장 짧게 하는 최적 알고리즘이기 때문에 이후 설명할 어떤 알고리즘보다도 평균 대기시간이 짧음

우선 순위 스케줄링 (Priority Scheduling)

- 우선 순위가 제일 높은 프로세스에게 CPU를 할당
- 일반적으로 우선 순위 값 (priority number)가 작을 수록 높은 우선 순위를 갖는다.
- 두 가지 방식
  - nonpreemptive (비선점형)
    - 일단 CPU를 잡으면 더 높은 우선 순위를 가진 프로세스가 들어와도 CPU 버스트가 완료될 때까지 CPU를 선점당하지 않음.
  - preemptive (선점형)

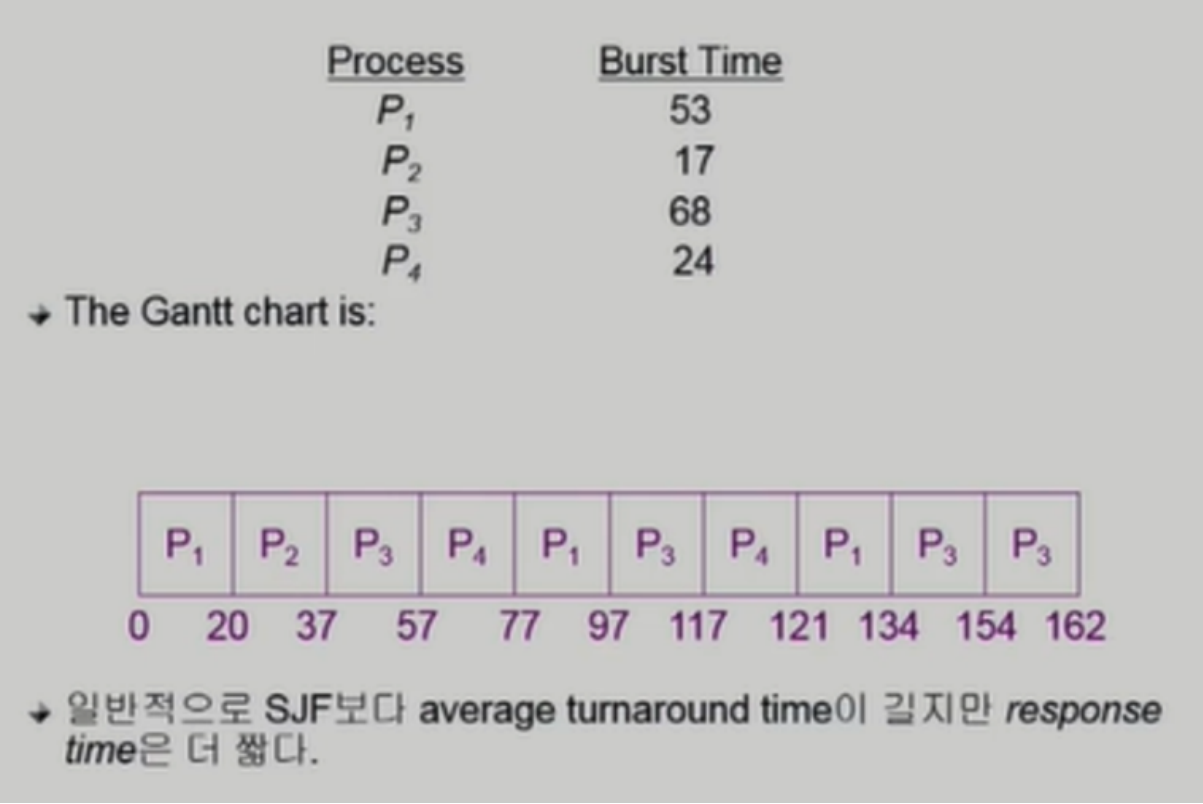
        - CPU를 잡았다 하더라도 더 높은 우선 순위를 가진 프로세스가 들어오면 CPU를 빼앗김
- SJF는 일종의 우선 순위 스케줄링이라고 볼 수 있음 (우선 순위 = 예상되는 다음 CPU 버스트 시간)
- 문제점
    - Starvation (기아): 우선 순위가 낮은 프로세스는 영원히 CPU를 잡지 못할 수 있음
- 해결 방안
    - Aging (노화): 아무리 우선 순위가 낮은 프로세스라 하더라도 시간이 오래 지나면 우선 순위를 높여주는 기법.
        - SJF도 일종의 우선순위 스케줄링이라고 볼 수 있으니 이 방법으로 해결할 수 있음
        - 우선순위 낮더라도 노인 공경

라운드 로빈 스케줄링 (RR, Round Robin)

현대적인 컴퓨터 시스템의서 사용하는 CPU 스캐줄링은 라운드 로빈의 기반함

- 각 프로세스는 동일한 크기의 할당 시간인 `time quantum` 을 가짐
- 할당 시간이 지나면 프로세스는 CPU를 빼앗기고 Ready Queue 맨 뒤에 가서 줄을 서게 됨(선점형)
- 짧은 응답 시간을 보장함
    - 조금씩 CPU를 줬다 뺏었다를 반복하기 때문에 CPU를 최초로 얻기까지 걸리는 시간이 짧음
    - n개의 프로세스가 Ready Queue에 있고, 할당 시간이 q time unit인 경우 어떤 프로세스도 `(n - 1) * q time unit` 이상 기다리지 않음 (자기 자신을 제외한 나머지 프로세스가 사용하는 할당 시간이 지나면 자기 차례가 돌아옴)
    - CPU를 할당받기 위해 기다리는 시간이 CPU 버스트에 비례
        - RR특징 중 재미있는 부분 CPU를 길게 쓰는 프로세스는 기다리는 시간도 길어지고 CPU를 짧게 쓰는 프로세스는 대기시간도 짧아짐 `time quantum`를 계속 도니까
- 성능
    - q가 커질 수록 FCFS에 가까워짐

    - q가 작을 수록 context switch 오버헤드가 증가함 (너무 짧게주면 오버헤드가 커져서 시스템 전체 성능이 나빠질 수 있음)
    - 보통 적당한 시간이 10에서 100millisecond 정도라고 함
- 일반적으로 SJF보다 평균 turnaround time (소요 시간), Waiting time이 길어질수 있지만, 응답 시간은 짧음
- 시간이 오래 걸리는 job과 짧게 걸리는 job이 섞여 있을 때는 효율적이지만, 모든 시간이 동일한 job만 있을 때는 비효율적이다. (RR은 기본적으로 CPU를 짧고, 길게 쓰는 프로세스들이 얼마나 쓸지 알 수 없는 상황에서 마구잡이로 섞여 있을 때 쓰기 좋은 스케줄링임)



위와 같이 라운드 로빈은 일정 할당 시간(20)을 공평하게 할당받아서 CPU 사용이 가능하다.