프로세스 스케줄링

(= CPU Scheduling, Job Scheduling)

어떤 프로세스를
먼저 실행시킬까?

목차

용어 정리

용어 정리

스케줄링 (Scheduling)

- Wikipedia: 컴퓨팅에서 스케줄링은 작업을 수행하기 위해 자원을 할당하는 작업이다. 여기서 자원은 processors, network links 또는 expansion cards가 될 수 있다. (→ 즉 스케줄링이란 **<컴퓨터의 자원을 할당할 프로세스를 고르는 작업>**이다!) 단위가 스레드가 될 수 있지 않나..?
- 스케줄링 작업은 스케줄러에 의해 수행된다.

스케줄러 (Scheduler)

- tutorialspoint: 스케줄러는 다양한 방법을 통해 프로세스 스케줄링을 처리하는 특별한 목적의 시스템 소프트웨어이다.
- 스케줄러의 주요 업무는 시스템에 제출할 작업을 선택하고 실행할 프로세스를 결정하는 것이다.
- 스케줄러는 장기 스케줄러/중기 스케줄러/단기 스케줄러 이렇게 총 3가지 유형으로 나뉜다.

용어 정리

디스패치 (Dispatch)

- 준비 리스트 맨 앞에 있던 프로세스에 프로세서를 할당하여 실행하는 것 (상태 전이: 준비 → 실행)

디스패처 (Dispatcher)

- Wikipedia: CPU 스케줄링 기능에 참여하는 또 다른 요소는 디스패처(Dispatcher)이다. 단기 스케줄러에 의해 선택된 프로세스에 CPU의 제어권을 주는 모듈이다. 인터럽트나 시스템콜의 결과로 커널 모드에서 제어권을 받는다. 디스패처의 기능은 다음을 포함한다.
    1. 문맥 전환(Context Switch)에서 디스패처는 이전에 실행중이던 프로세스 또는 스레드의 상태를 저장한다; 디스패처는 초기 또는 이전에 저장했던 새로운 프로세스의 상태를 로드한다.
    2. 유저 모드로 전환한다.
    3. new state로 표시된 프로그램을 다시 시작하기 위해 사용자 프로그램 안에서 적절한 위치로 점프한다.

하나의 프로세스를 종료하고 다른 프로세스를 시작시키기 위해 걸리는 시간을 “dispatch latency” 라고 한다.

스케줄링의 목적과
스케줄링 시 고려 사항

스케줄링의 목적

1. **공평성**

   모든 프로세스가 자원을 공평하게 배정받아야 하며, 자원 배정 과정에서 특정 프로세스가 배제되어서는 안 된다.

2. **효율성**

   시스템 자원이 유휴 시간 없이 사용되도록 스케줄링을 하고, 유휴 자원을 사용하려는 프로세스에는 우선권을 주어야 한다.

3. **안정성**

   우선순위를 사용하여 중요 프로세스가 먼저 작동하도록 배정함으로써 시스템 자원을 점유하거나 파괴하려는 프로세스로부터 자원을 보호해야한다.

4. **확장성**

   프로세스가 증가해도 시스템이 안정적으로 작동하도록 조치해야 한다. 또한 시스템 자원이 늘어나는 경우 이 혜택이 시스템에 반영되게 해야 한다.

5. **반응 시간 보장**

   응답이 없는 경우 사용자는 시스템이 멈춘 것으로 가정하기 때문에 시스템은 적절한 시간 안에 프로세스의 요구에 반응해야 한다.

6. **무한 연기 방지**

   특정 프로세스의 작업이 무한히 연기되어서는 안 된다.

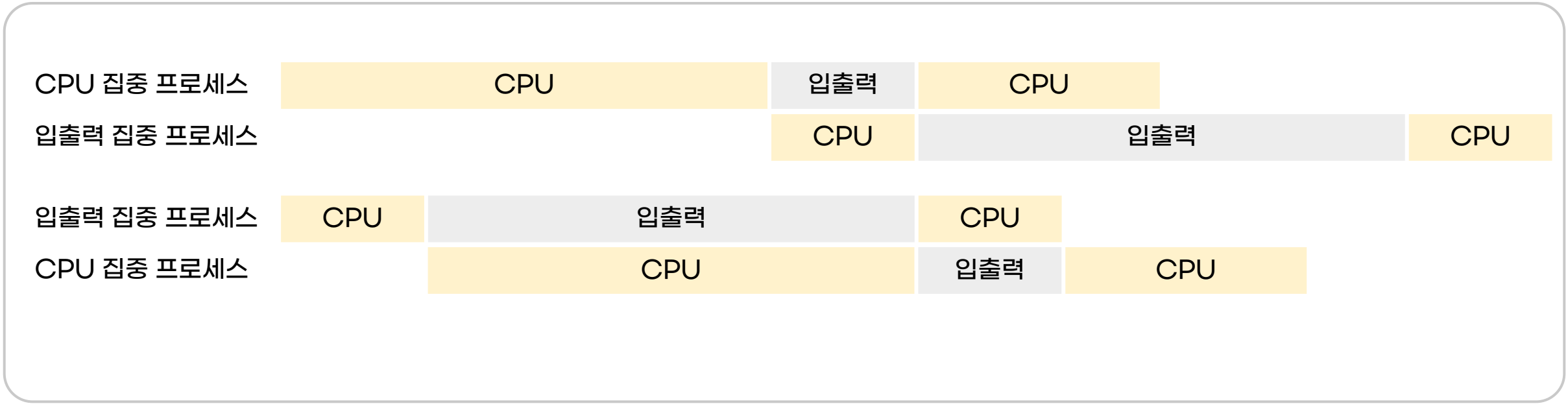
스케줄링 시 고려사항

1. **선점형 스케줄링 vs 비선점형 스케줄링 (CPU를 빼앗을 수 있느냐 없느냐)**
   - 선점형 스케줄링: 어떤 프로세스가 CPU를 할당받아 실행중이더라도 운영체제가 CPU를 강제로 빼앗을 수 있는 스케줄링 방식
   - 비선점형 스케줄링: 어떤 프로세스가 CPU를 할당받아 실행중이면 운영체제가 CPU를 강제로 뺏을 수 없는 스케줄링 방식
   - 선점형 스케줄링은 한 프로세스가 CPU를 독점할 수 없어서 대화형이나 시분할 시스템에 적합하다. 문맥 교환시 오버헤드가 발생한다는 단점이 있다.
   - 비선점형 스케줄링은 일괄 처리 방식 스케줄러에 사용한다, CPU 스케줄러의 작업량이 적고 문맥 교환의 오버헤드가 적다는 장점이 있지만, 기다리는 프로세스가 많아 처리율이 떨어진다.
2. **프로세스 우선순위 (커널 프로세스 vs 일반 프로세스)**
   - 우선순위가 높다 = 더 빨리 자주 실행된다.
   - 커널과 관련된 중요한 작업은 우선적으로 처리되어야 하기 때문에, 우선순위를 고려하는 것이 필요하다.
   - 일반 프로세스들의 우선순위도 서로 다른데, 사용자가 조정이 가능하다. (리눅스에서는 `renice`)
3. **전면 프로세스 vs 후면 프로세스**
   - 전면 프로세스: GUI를 사용하는 운영체제에서 화면의 맨 앞에 놓인 프로세스. 현재 입력과 출력을 사용하는 프로세스로, 사용자와 상호작용이 가능하여 상호작용 프로세스라고 한다.
   - 후면 프로세스: 사용자와 상호작용이 없는 프로세스. 압축 프로그램처럼 사용자의 입력 없이 작동하기 때문에, 일괄 작업 프로세스라고도 한다.
   - 전면 프로세스는 사용자의 요구에 즉각 반응해야 하지만, 후면 프로세스는 상호작용이 없기 때문에, 전면 프로세스의 우선순위가 후면 프로세스보다 높다.

스케줄링 시 고려사항

4. CPU 집중 프로세스 vs 입출력 집중 프로세스

- CPU를 할당받아 실행하는 작업을 CPU 버스트(CPU burst), 입출력 작업을 입출력 버스트(I/O burst)라고 한다.
- 위의 작업 형태에 따라 CPU 집중 프로세스와 입출력 집중 프로세스로 나눌 수 있다.
- CPU 집중 프로세스: 수한 연산과 같이 CPU를 많이 사용하는 프로세스 (CPU 버스트가 많은 프로세스)
- 입출력 집중 프로세스: 저장장치에서 데이터를 복사하는 일과 같이 입출력을 많이 사용하는 프로세스 (입출력 버스트가 많은 프로세스)
- 입출력 집중 프로세스는 CPU를 많이 안쓰기 때문에, 먼저 처리하면 시스템의 효율을 높일 수 있다.



스케줄링 알고리즘

스케줄링 알고리즘

비선점형 알고리즘	FCFS 스케줄링, SJF 스케줄링, HRN 스케줄링
선점형 알고리즘	라운드 로빈 스케줄링, SRT 스케줄링, 다단계 큐 스케줄링, 다단계 피드백 큐 스케줄링
둘 다 가능	우선순위 스케줄링

선택 기준

- **CPU 사용률**: 전체 시스템의 동작 시간 중 CPU가 사용된 시간을 측정하는 방법
- **처리량**: 시스템이 정상적으로 작동한다면 일정 시간 후 작업이 끝난다. 처리량은 단위 시간당 작업을 마친 프로세스의 수로, 이 수치가 클수록 좋은 알고리즘이다.
- **대기 시간**: 작업을 요청한 프로세스가 작업을 시작하기까지 대기하는 시간. 짧을수록 좋다.
- **응답 시간**: 프로세스 시작 후 첫 번째 출력 또는 반응이 나올 때까지 걸리는 시간. 짧을수록 좋다.
- **반환 시간**: 대기 시간 + 실행 시간. 프로세스가 생성된 후 종료되어 사용하던 자원을 모두 반환하는데까지 걸리는 시간

스케줄링 알고리즘의 성능을 비교할 때는 주로 평균 대기 시간을 본다. 평균 대기 시간은 모든 프로세스의 대기 시간을 합한 후 프로세스의 수로 나눈 값이다.

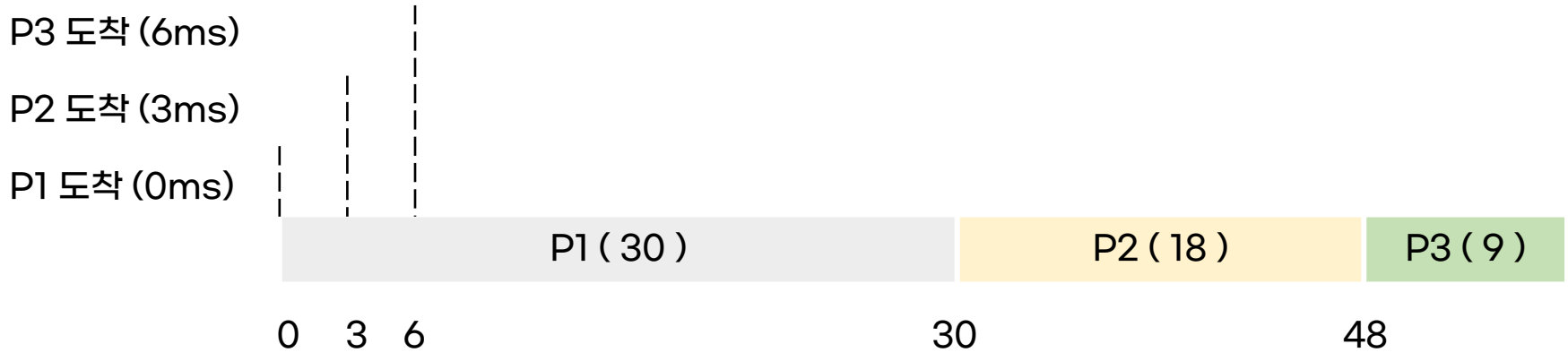
1. FCFS (First Come First Served) 스케줄링

- FIFO(First In First Out) 라고도 하는데, 이는 큐를 가리키는 말이기 때문에, 이와 구분하기 위해 스케줄링 알고리즘에서는 FCFS라고 부른다.
- 스케줄링 준비 큐에 도착한 순서대로 CPU를 할당하는 비선점형 방식으로 큐가 하나라 모든 우선순위가 동일하다.

[단점]

- 처리 시간이 긴 프로세스가 있다면 이를 처리하느라 다음 프로세스의 대기 시간이 길어질 수 있다. 이처럼 앞의 작업이 오래 걸려 뒤의 작업이 지연되는 현상을 콘보이 효과(convoy effect) 또는 호위 효과라고 한다.
- 현재 작업 중인 프로세스가 입출력 작업을 요청하는 경우 CPU가 작업하지 않고 쉬는 시간이 늘어난다.

도착 순서	도착 시간	작업 시간
P1	0	30
P2	3	18
P3	6	9



$$\text{평균 대기 시간} = \frac{0+(30-3)+(48-6)}{3} = \frac{0+27+42}{3} = 23\ (ms)$$

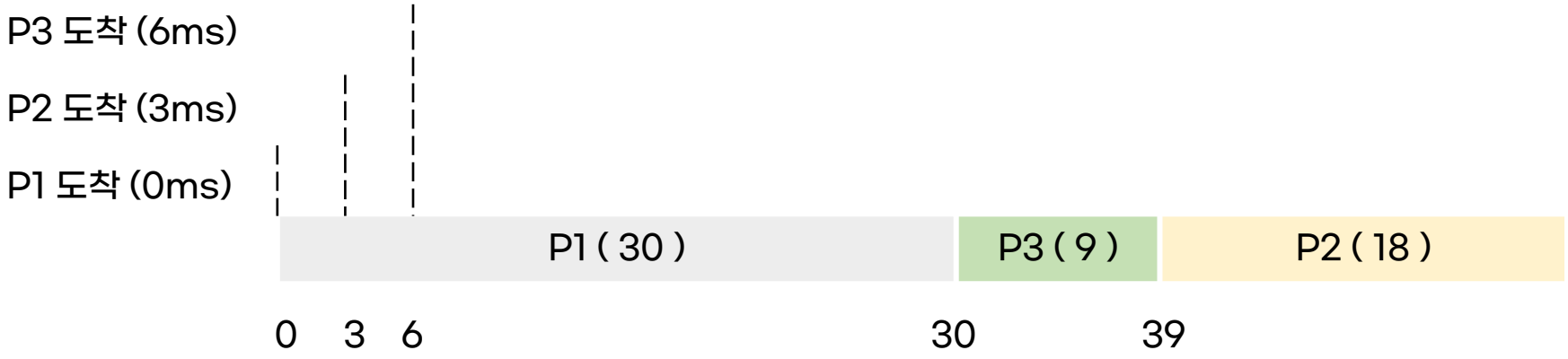
2. SJF (Short Job First) 스케줄링

- SPF(Shortest Process First) 또는 최단 프로세스 우선 스케줄링이라고 한다.
- 실행 시간이 가장 짧은 작업부터 CPU를 할당하는 비선점형 방식이다.
- 콘보이 효과를 완화하여 시스템의 효율을 높인다.

[단점]

- 운영체제가 프로세스의 종료 시간을 정확하게 예측하기 어렵다. 특히 현대의 프로세스는 사용자와 상호작용이 많기 때문에 예측이 어렵다. 이는 프로세스가 자신의 작업 시간을 운영체제에 알려주면 해결이 가능하지만, 프로세스 또한 자신의 작업 시간을 정확히 알기 어렵다.
- 먼저 들어왔음에도 실행 시간이 짧은 프로세스에 의해 계속 뒤로 밀릴 수 있다. (아사(starvation) 현상 또는 무한 봉쇄(infinite blocking) 현상) 이는 에이징(aging)으로 완화할 수 있다. 에이징은 프로세스가 양보할 수 있는 상한선을 정하는 방식이다.

도착 순서	도착 시간	작업 시간
P1	0	30
P2	3	18
P3	6	9



평균 대기 시간 = $\frac{0+(30-6)+(39-3)}{3} = \frac{0+24+36}{3} = 20\ (ms)$

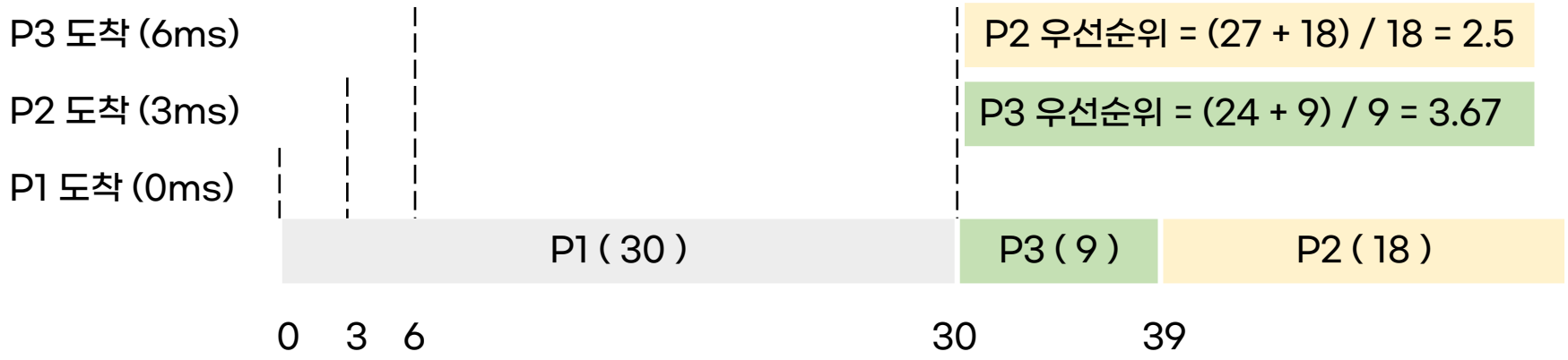
3. HRN (Highest Response Ration Next) 스케줄링

- 아사 현상을 해결하기 위해 만들어진 비선점형 알고리즘으로 최고 응답률 우선 스케줄링이라고도 한다.
- HRN 스케줄링에서는 `(대기 시간 + CPU 사용 시간) / CPU 사용 시간` 으로 우선순위를 결정한다.
- 대기 시간을 고려함으로써 아사 현상을 완화시킨다. 해당 스케줄링에서는 숫자가 클수록 우선순위가 높다.

[단점]

- 아사 현상은 완화되지만, 여전히 공평성이 위배되어 많이 사용되지 않음

도착 순서	도착 시간	작업 시간
P1	0	30
P2	3	18
P3	6	9

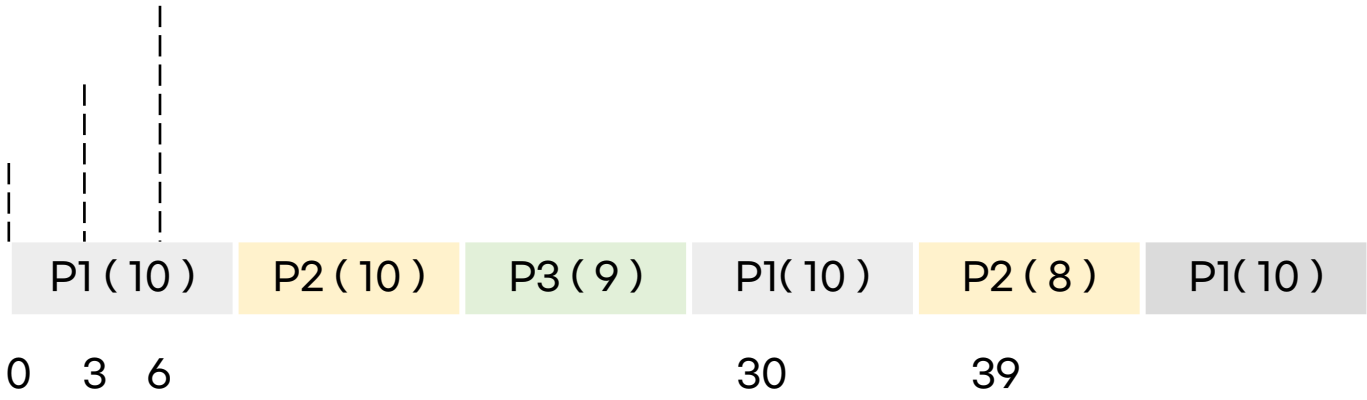


평균 대기 시간 = $\frac{0+(30-6)+(39-3)}{3} = \frac{0+24+36}{3} = 20\ (ms)$

4. 라운드 로빈 (=순환 순서 방식) 스케줄링

- 한 프로세스가 할당받은 시간(타임슬라이스)동안 작업을 하다가 작업을 완료하지 못하면 준비 큐의 맨 뒤로 가서 자기 차례를 기다리는 방식
- 선점형 알고리즘 중 우선순위가 적용되지 않은 가장 단순하고 대표적인 방식으로, 프로세스들이 작업을 완료할 때까지 계속 순환하면서 실행된다.
- 앞의 긴 작업을 무작정 기다리는 콘베이 효과가 줄어든다.
- 문맥 교환에 따른 추가 시간을 고려하여 타임 슬라이스를 적절히 설정해야한다.
- 타임 슬라이스가 큰 경우 하나의 작업이 끝난 뒤 다음 작업이 시작되는 것 처럼 보이며 이 경우 FCFS 스케줄링과 다를 게 없다.
- 타임 슬라이스가 작은 경우 여러 프로그램이 동시에 실행되는 것처럼 느껴질 수 있지만, 너무 작게 설정하면 문맥 교환이 너무 자주 일어나 문맥 교환에 걸리는 시간이 실제 작업 시간보다 상대적으로 커질 수 있다.
- 유닉스 운영체제에서는 타임 슬라이스가 대략 100밀리초이다. 타임슬라이스는 고정되어 있지 않고 10~200밀리초 사이에서 조정될 수 있다.

도착 순서	도착 시간	작업 시간
P1	0	30
P2	3	18
P3	6	9



평균 대기 시간 = $\frac{0(P1)+7(P2)+14(P3)+19(P1)+19(P2)+8(P1)}{3} = \frac{67}{3} = 22.33(ms)$

5. SRT (Shortest Remaining Time) 스케줄링

- SJF 스케줄링과 라운드 로빈 스케줄링을 혼합한 방식으로 최소 잔류 시간 우선 스케줄링이라고도 한다. 쉽게 말해 SJF 스케줄링의 선점형 버전이라고 할 수 있다.
- CPU를 할당받을 프로세스를 선택할 때 남아 있는 작업 시간이 가장 적은 프로세스를 선택한다.
- 라운드 로빈 스케줄링이 큐에 있는 순서대로 CPU를 할당한다면, SRT 스케줄링은 남은 시간이 적은 프로세스에 CPU를 먼저 할당한다.

[단점]

- 현재 실행 중인 프로세스와 큐에 남아 있는 프로세스의 남은 시간을 주기적으로 계산하고 남은 시간이 더 적은 프로세스와 문맥 교환을 해야한다는 단점이 있다.
- SJF 스케줄링과 마찬가지로 운영체제가 프로세스의 종료 시간을 예측하기 어렵고 아사 현상이 일어나기 때문에 잘 사용하지 않는다.

6. 우선순위 스케줄링

- 중요도에 따라 우선순위를 정하고 이러한 우선순위를 반영한 스케줄링 알고리즘이다.
- 어떤 기준으로 우선순위를 정하느냐에 따라 다양하게 구현할 수 있다.

[단점]

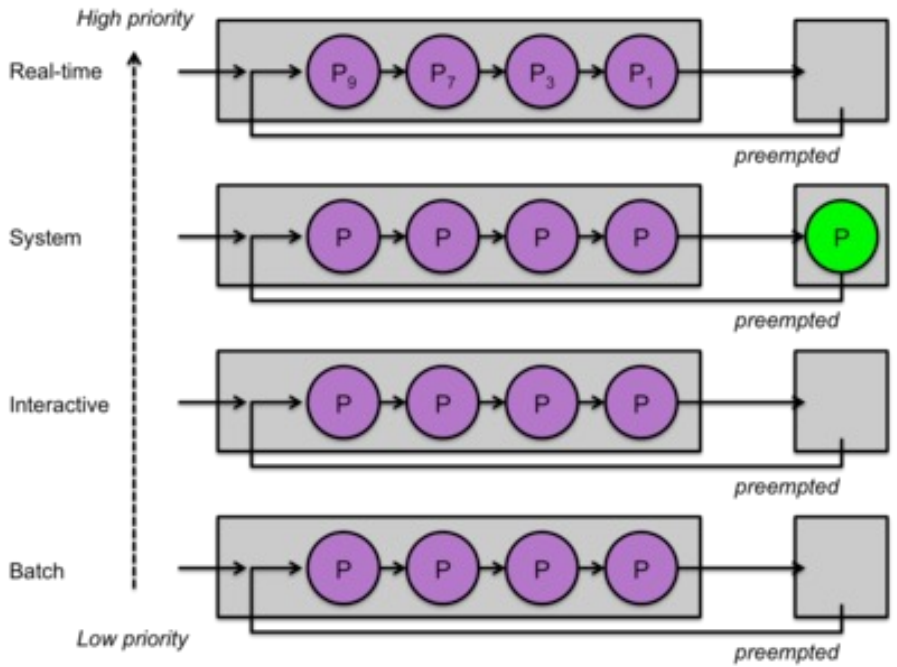
- 준비 큐에 있는 프로세스의 순서를 무시하고 프로세스의 우선순위를 매번 바꿔야하기 때문에 오버헤드가 발생하여 시스템의 효율성을 떨어뜨린다.

7. 다단계 큐 스케줄링 (Multiple Queue Scheduling)

- 우선순위에 따라 큐를 나눠 스케줄링한다.
- 우선순위는 고정형 우선순위를 사용하며, 상단의 큐에 있는 모든 프로세스가 실행되어야 다음 우선순위 큐의 작업이 시작된다.

[단점]

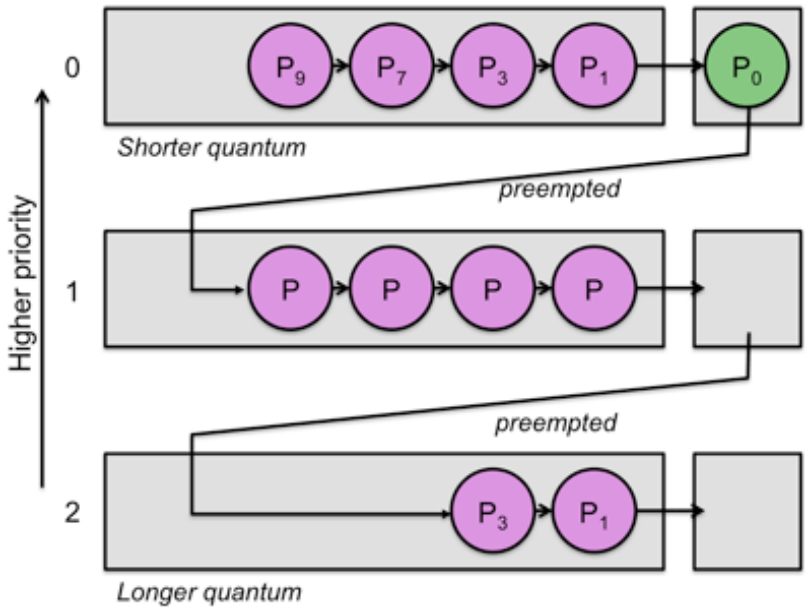
- 우선순위가 높은 프로세스 때문에 우선순위가 낮은 프로세스의 작업이 연기되는데 이러한 문제를 해결하기 위해 제안된 것이 다단계 피드백 큐 스케줄링이다.



이미지 출처: https://people.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html

8. 다단계 피드백 큐 스케줄링 (Multilevel Feedback Queue Scheduling)

- CPU를 사용하고 난 프로세스의 우선순위가 낮아진다.
- 우선순위에 따라 타임 슬라이스의 크기가 다르다.
- 우선순위가 낮은 프로세스의 실행 기회를 확대하려고 하지만, 그렇다고 해도 우선순위가 낮은 프로세스가 우선순위가 높은 프로세스보다 CPU를 얻을 확률은 여전히 낮다. 따라서 어렵게 얻은 CPU를 좀 더 오랜동안 사용할 수 있도록 우선순위가 낮은 큐의 타임 슬라이스를 크게 설정한다.
- 다단계 피드백 큐 스케줄링은 오늘날의 운영체제가 CPU 스케줄링을 위해 일반적으로 사용하는 방식으로, 변동 우선순위 알고리즘의 전형적인 예이다.



이미지 출처: https://people.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html

운영체제별 사용하는 스케줄링 알고리즘

운영체제별 사용하는 알고리즘 - 요약

Operating System	Preemption	Algorithm
Amiga OS	Yes	Prioritized round-robin scheduling
FreeBSD	Yes	Multilevel feedback queue
Linux kernel before 2.6.0	Yes	Multilevel feedback queue
Linux kernel 2.6.0–2.6.23	Yes	O(1) scheduler
Linux kernel after 2.6.23	Yes	Completely Fair Scheduler
classic Mac OS pre-9	None	Cooperative scheduler
Mac OS 9	Some	Preemptive scheduler for MP tasks, and cooperative for processes and threads
macOS	Yes	Multilevel feedback queue
NetBSD	Yes	Multilevel feedback queue
Solaris	Yes	Multilevel feedback queue
Windows 3.1x	None	Cooperative scheduler
Windows 95, 98, Me	Half	Preemptive scheduler for 32-bit processes, and cooperative for 16-bit processes
Windows NT (including 2000, XP, Vista, 7, and Server)	Yes	Multilevel feedback queue

이미지 출처: Wikipedia / Scheduling (Computing)

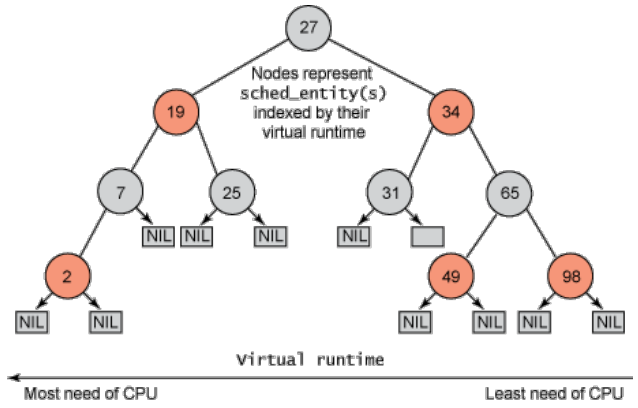
리눅스 커널 스케줄링 (2.6.23 이전)

- 리눅스에서는 프로세스와 스레드를 동일한 것으로 간주하여 프로세스 및 스레드 스케줄링을 통합적으로 처리한다. (task_struct 가 하나의 수행단위로 실행됨)
- 리눅스에서는 run queue에서 다음에 실행할 태스크를 선택하는데, run queue는 active array와 expired array로 구성된다.
- 주어진 time slice를 다 사용하면 expired array로 빠지게 되고, 만약 실행중 다른 프로세스에게 뺏겨서 time slice를 다 사용하지 못했다면 active array에 다시 추가한다.
- active array에 있는 프로세스를 모두 수행하면 expired array가 active array가 된다. (expired array와 active array 교체)
- 따라서 리눅스에서 프로세스가 CPU를 차지하기 위해서는 남은 time slice가 0보다 크면서 우선순위가 높아야한다.

버전	사용한 알고리즘
1.2	• 라운드 로빈 스케줄링 정책에 따라 작동하는 순환형 큐 사용 • 단순하며 빠르지만, 다수의 프로세서나 하이퍼 스레딩이 포함된 대형 아키텍처를 전혀 고려하지 않는다.
2.2	• 스케줄링 클래스라는 개념이 채택되면서 실시간 작업, 우선 순위가 없는 작업 및 비실시간 작업에 대한 스케줄링 정책을 사용할 수 있다. • SMP(Symmetric Multiprocessing)에 대한 지원도 포함되어 있다.
2.4	• O(n) Scheduler 사용(우선순위 범위: 0 ~ 140) • 시간은 에포크(Epoch) 단위로 나누며 모든 작업은 해당 시간 조각동안 실행할 수 있다. 작업이 주어진 time slice를 다 사용하지 못하고 끝나면 남은 시간의 절반을 추가하여 다음 에포크에서 더 길게 실행할 수 있도록 했다. • 다음에 실행할 태스크를 찾기 위해서는 대기하고 있는 모든 프로세스의 PCB를 확인해봐야 했기 때문에, O(n)만큼의 시간복잡도를 가진다.
2.6 ~ 2.6.22	• O(1) Scheduler 사용 • 우선 순위 레벨을 두고 우선 순위 레벨별로 두 개의 실행 큐를 사용했다. 따라서 다음 작업을 가져오기 위해서는 우선 순위에 해당하는 큐에 가서 작업을 가져오기만 하면 된다.

리눅스 커널 스케줄링 (2.6.23 부터 그 이후) - Completely Fair Scheduler (CFS)

- O(1) 스케줄러의 작성자인 Ingo Molnar은 Con Kolivas의 Staircase scheduler에 포함되어 있던 RDSL(Rotating Stairecase Deadline Scheduler)에서 영감을 받아 Completely Fair Scheduler(CFS)를 만들었다.
- CFS의 기본 개념은 프로세서 시간을 제공할 때 공평성을 유지하는 것이다.
- CFS 에서는 밸런스를 결정하기 위해 “가상 런타임”이라는 지정된 작업에 제공된 시간의 양을 관리하며 이는 프로세서에 엑세스할 수 있도록 허용된 시간을 의미한다.
- 가상 런타임이 작을수록 더 많은 프로세서 시간이 필요하다. 또한 대기자 공평성이라는 개념을 통해 현재 실행할 수 없는 작업이 나중에 프로세서가 필요할 때 대기했던 시간에 상응하는 프로세서 시간을 받을 수 있도록 보장한다.
- 이전 Linux 스케줄러와 달리 실행 큐에서 작업을 관리하지 않고 가상 런타임순으로 정렬된 red-black 트리를 유지한다.
- CFS에서는 우선 순위를 직접 사용하지 않는 대신 작업에 허용된 실행 시간(가상 런타임)에 대한 지연 인수로 사용한다. 우선순위가 낮을수록 지연 인수가 높은 작업이며, 우선 순위가 높을수록 지연 인수가 낮은 작업이다.



Red-Black 트리 추가 설명

- red-black 트리는 스스로 밸런스를 조절한다는 특성이 있다.
- 가상 런타임이 가장 낮은 작업, 즉 가장 왼쪽에 있는 작업을 다음에 실행할 노드로 선택한다.

리눅스 커널 우선순위

- 리눅스에서는 숫자가 낮을수록 우선순위가 높다.
- 리눅스에서 우선순위 범위는 0 ~ 140이고, 이 범위는 2개로 나눠진다. 하나는 nice 값을 위한 공간이며, 하나는 실시간 우선순위를 위한 공간이다.
- nice 값은 -20 에서 19의 범위를 가지며 기본값은 0이다. (-20: 높은 우선순위, 19: 낮은 우선순위)
- 실시간 우선순위(real-time priority)에 대한 범위는 기본적으로 1부터 99이며, 100부터 139는 user-space를 위한 공간이다.
- 우선순위를 포함하여 프로세스 확인하는 명령어: ps -el
- 프로세스 실행 전 우선순위 변경하는 명령어(기존 우선순위에 N만큼을 더해 우선순위를 낮춘다.): nice -n [N] [프로세스명]
- 실행중인 프로세스의 우선순위를 변경하는 명령어: renice [N] PID

```
Total number of priorities = 140
Real time priority range(PR or PRI):  0 to 99
User space priority range: 100 to 139
```

Nice value range (NI): -20 to 19

```
PR = 20 + NI
PR = 20 + (-20 to + 19)
PR = 20 + -20  to 20 + 19
PR = 0 to 39 which is same as 100 to 139.
```

PRI와 NI

- PRI는 운영체제에서 참고하는 우선순위로 시스템 상황에 따라 알아서 부여되기 때문에 사용자가 조작할 수 없다.
- NI는 nice 값으로 사용자가 조작 가능한 우선순위이다.

이미지 출처: https://www.tecmint.com/set-linux-process-priority-using-nice-and-renice-commands/

퀴즈

1. 프로세스와 스레드 대한 설명으로 알맞지 않은 것은? (1개 이상)

1️⃣ 스레드는 자원을 공유하고 있고, 프로세스에 비해 가볍기 때문에 Lightweight process라고도 한다.

2️⃣ Process의 상태에는 New, Ready, Runnable, Run, Wait, Terminated가 있다.

3️⃣ 프로세스는 운영체제로부터 자원을 할당받는 단위이다.

4️⃣ 프로세스가 실행(Run)이 되다가 주어진 시간이 끝나면 대기(Wait) 상태로 전환된다.

2. 스레드에 스택을 독립적으로 할당하는 이유는?

3. 프로세스 제어 블록에 저장되는 정보로 올바르지 않은 것은?

1️⃣ 프로세스 상태 2️⃣ Thread control block리스트 3️⃣ 형제 프로세스 아이디 4️⃣ 프로세스 우선순위

퀴즈

1. 퀴즈 1

1️⃣ 스레드는 자원을 공유하고 있고, 프로세스에 비해 가볍기 때문에 Lightweight process라고도 한다.

2️⃣ **Process의 상태에는 New, Ready, Runnable, Run, Wait, Terminated가 있다.**

→ **Runnable은 Java의 Thread 에서 지원하는 Thread 상태 종류 중 하나이다.**

3️⃣ 프로세스는 운영체제로부터 자원을 할당받는 단위이다.

4️⃣ **프로세스가 실행(Run)이 되다가 주어진 시간이 끝나면 대기(Wait) 상태로 전환된다.**

→ **대기가 아니라 준비큐에 삽입되고 준비(Ready) 상태로 전환된다.**

2. 퀴즈 2

스택은 함수와 관련된 값과 돌아갈 위치가 저장되어 있는 메모리 공간이기 때문에, 독립적인 실행 흐름을 추가하기 위해 스택을 독립적으로 할당한다.

3. 퀴즈 3

1️⃣ 프로세스 상태 2️⃣ Thread control block리스트 3️⃣ **형제 프로세스 아이디** 4️⃣ 프로세스 우선순위

참고

1. 프로세스 스케줄링의 전반적인 개념 - [책] 쉽게 배우는 운영체제 | 한빛아카데미
2. 용어 정리 > 스케줄링, 디스패처
   https://en.wikipedia.org/wiki/Scheduling_(computing)
3. 용어 정리 > 스케줄러
   https://www.tutorialspoint.com/operating_system/os_process_scheduling.htm
4. 리눅스 커널 스케줄링
   - “[Linux Kernel] 14. Kernel-Scheduling(커널 스케쥴링)”: https://coder-in-war.tistory.com/entry/Embedded-21-Linux-Kernel-Kernel-Scheduling%EC%BB%A4%EB%84%90-%EC%8A%A4%EC%BC%80%EC%A5%B4%EB%A7%81
   - ”리눅스 커널 스케줄러”: http://cloudrain21.com/linux-kernel-scheduler

5. CFS
   https://dataonair.or.kr/db-tech-reference/d-lounge/technical-data/?mod=document&uid=236896

[참고하려 했던 자료(읽어보면 좋을 듯함)]
“Linux Schedling”: https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf