Chapter 04

CPU 스케줄링

Contents

학습목표

- CPU 스케줄링의 의미를 이해하고 단계와 목적을 알아본다.
- 스케줄링 시 고려할 사항을 알아본다.
- 준비 상태와 대기 상태에서 다중 큐가 어떻게 운영되는지 알아본다.
- 스케줄링 알고리즘의 종류와 각 방식의 장단점을 파악한다.

1-1 식당 관리자의 스케줄링

식당 관리자의 역할

- 예약 관리
- 좌석 관리
- 주문 관리
- 조리 순서 관리
- 손님 요청 관리

그림 4-1 식당 관리자의 역할

1-1 식당 관리자의 스케줄링

CPU 스케줄러

- 운영체제에서 식당 관리자의 역할을 담당
- 여러 프로세스의 상황을 고려하여 CPU와 시스템 자원의 배정을 결정

그림 4-2 조리 순서 변경

1-2 스케줄링의 단계

고수준 스케줄링

- 시스템 내의 전체 작업 수를 조절하는 것
- 어떤 작업을 시스템이 받아들일지 또는 거부할지를 결정
- 시스템 내에서 동시에 실행 가능한 프로세스의 총개수가 정해짐
- 장기 스케줄링, 작업 스케줄링, 승인 스케줄링이라고도 함

저수준 스케줄링

- 어떤 프로세스에 CPU를 할당할지, 어떤 프로세스를 대기 상태로 보낼지 등을 결정
- 아주 짧은 시간에 일어나기 때문에 단기 스케줄링이라고도 함

1-2 스케줄링의 단계

중간 수준 스케줄링

- 중지와 활성화로 전체 시스템의 활성화된 프로세스 수를 조절하여 과부하를 막음
- 일부 프로세스를 중지 상태로 옮김으로써 나머지 프로세스가 원만하게 작동하도록 지원
- 저수준 스케줄링이 원만하게 이루어지도록 완충하는 역할

1-2 스케줄링의 단계

정리

고수준 스케줄링	전체 시스템의 부하를 고려하여 작업을 시작할지 말지를 결정
중간 수준 스케일링	시스템에 과부하가 걸려서 전체 프로세스 수를 조절해야 한다면 이미 활성화된 프로세스 중 일부를 보류 상태로 보냄
저수준 스케줄링	실제 작업을 수행

1-2 스케줄링의 단계

정리

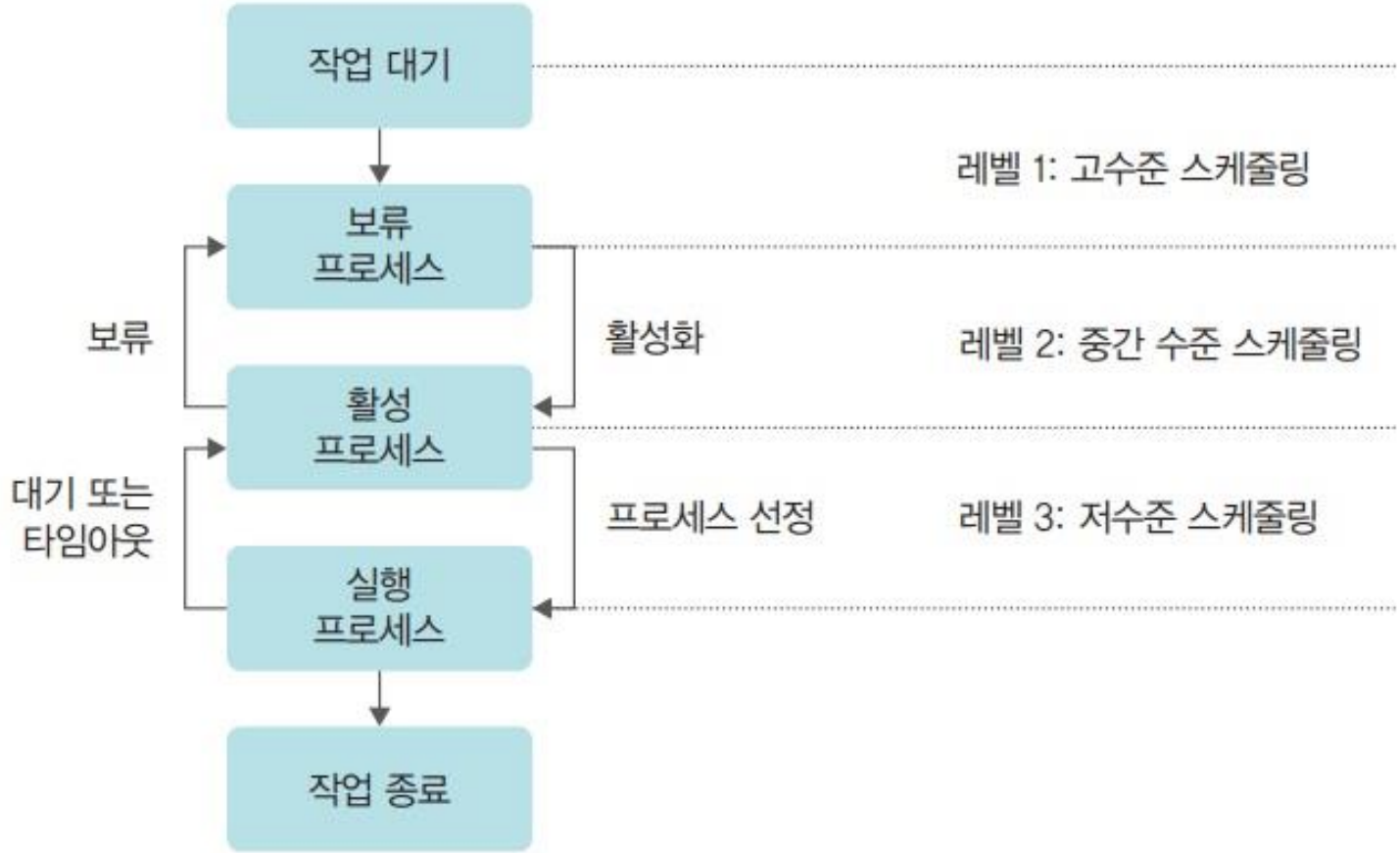


그림 4-3 스케줄링의 단계

1-3 스케줄링의 목적

CPU 스케줄링의 목적

공평성	모든 프로세스가 자원을 공평하게 배정받아야 하며, 자원 배정 과정에서 특정 프로세스가 배제되어서는 안 됨
효율성	시스템 자원이 유휴 시간 없이 사용되도록 스케줄링을 하고, 유휴 자원을 사용하려는 프로세스에는 우선권을 주어야 함
안정성	우선순위를 사용하여 중요 프로세스가 먼저 작동하도록 배정함으로써 시스템 자원을 점유하거나 파괴하려는 프로세스로부터 자원을 보호해야 함
확장성	프로세스가 증가해도 시스템이 안정적으로 작동하도록 조치해야 하며 시스템 자원이 늘어나는 경우 이 혜택이 시스템에 반영되게 해야 함
반응 시간 보장	응답이 없는 경우 사용자는 시스템이 멈춘 것으로 가정하기 때문에 시스템은 적절한 시간 안에 프로세스의 요구에 반응해야 함
무한 연기 방지	특정 프로세스의 작업이 무한히 연기되어서는 안 됨

2-1 선점형 스케줄링과 비선점형 스케줄링

선점형 스케줄링

- 운영체제가 필요하다고 판단하면 실행 상태에 있는 프로세스의 작업을 중단시키고 새로운 작업을 시작할 수 있는 방식
- 하나의 프로세스가 CPU를 독점할 수 없기 때문에 빠른 응답 시간을 요구하는 대화형 시스템이나 시분할 시스템에 적합
- 대부분의 저수준 스케줄러는 선점형 스케줄링 방식을 사용

비선점형 스케줄링

- 어떤 프로세스가 실행 상태에 들어가 CPU를 사용하면 그 프로세스가 종료되거나 자발적으로 대기 상태에 들어가기 전까지는 계속 실행되는 방식
- 선점형 스케줄링보다 스케줄러의 작업량이 적고 문맥 교환에 의한 낭비도 적음
- CPU 사용 시간이 긴 프로세스 때문에 CPU 사용 시간이 짧은 여러 프로세스가 오랫동안 기다리게 되어 전체 시스템의 처리율이 떨어짐
- 과거의 일괄 작업 시스템에서 사용하던 방식

2-1 선점형 스케줄링과 비선점형 스케줄링

표 4-1 선점형 스케줄링과 비선점형 스케줄링의 비교

구분	선점형	비선점형
작업 방식	실행 상태에 있는 작업을 중단시키고 새로운 작업을 실행할 수 있다.	실행 상태에 있는 작업이 완료될 때까지 다른 작업이 불가능하다.
장점	프로세스가 CPU를 독점할 수 없어 대화형이나 시분할 시스템에 적합하다.	CPU 스케줄러의 작업량이 적고 문맥 교환의 오버헤드가 적다.
단점	문맥 교환의 오버헤드가 많다.	기다리는 프로세스가 많아 처리율이 떨어진다.
사용	시분할 방식 스케줄러에 사용된다.	일괄 작업 방식 스케줄러에 사용된다.
중요도	높다.	낮다.

2-2 프로세스 우선순위

프로세스 우선순위

- 커널 프로세스의 우선순위가 일반 프로세스보다 높음
- 시스템에는 다양한 우선순위의 프로세스가 공존하며 우선순위가 높은 프로세스가 CPU를 먼저, 더 오래 차지
- 시스템에 따라 높은 숫자가 높은 우선순위를 나타내기도 하고, 낮은 숫자가 높은 우선순위를 나타내기도 함

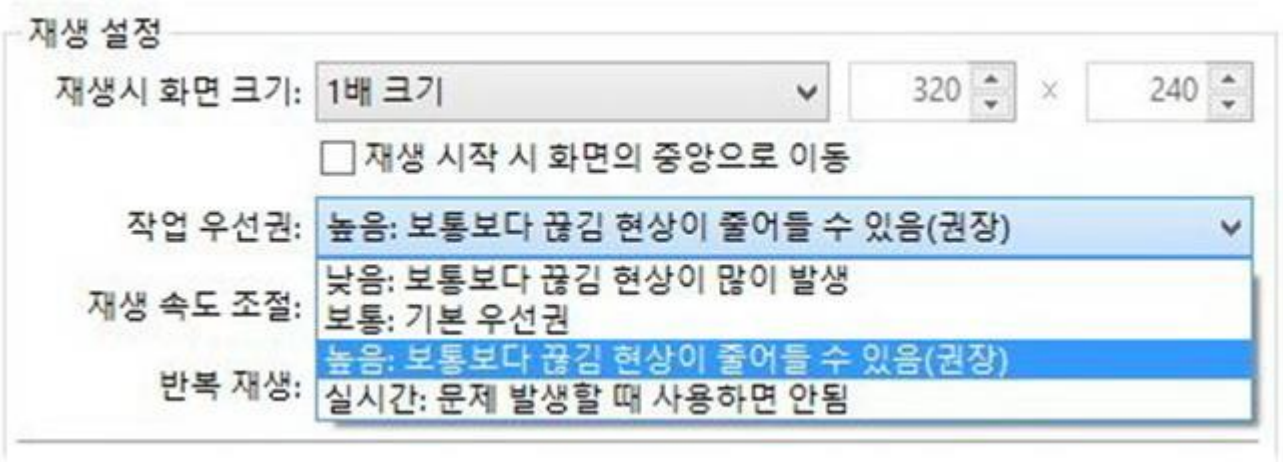


그림 4-4 윈도우의 우선 순위 조절

2-3 CPU 집중 프로세스와 입출력 집중 프로세스

CPU 집중 프로세스

- 수학 연산과 같이 CPU를 많이 사용하는 프로세스로 CPU 버스트가 많은 프로세스

입출력 집중 프로세스

- 저장장치에서 데이터를 복사하는 일과 같이 입출력을 많이 사용하는 프로세스로 입출력 버스트가 많은 프로세스

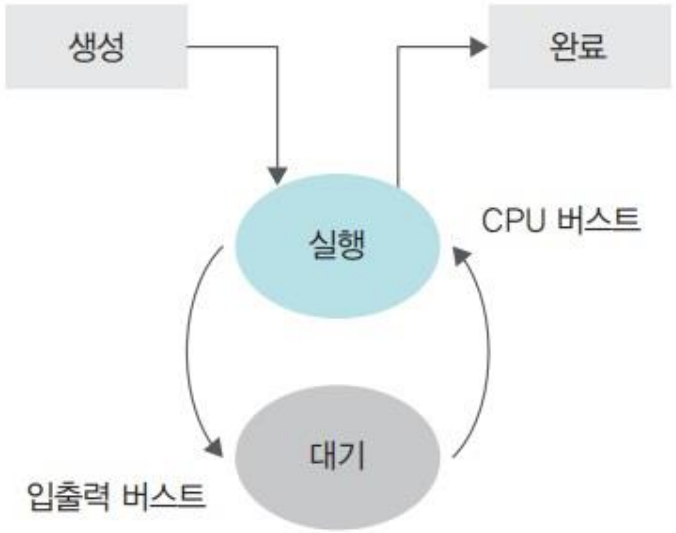
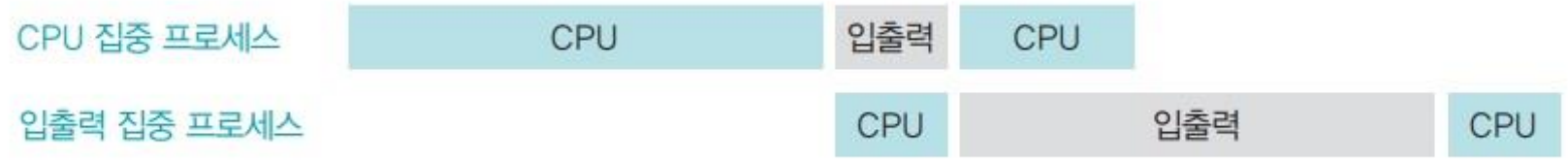


그림 4-5 CPU 버스트와 입출력 버스트

2-3 CPU 집중 프로세스와 입출력 집중 프로세스

우선 배정

- 스케줄링을 할 때 입출력 집중 프로세스의 우선순위를 CPU 집중 프로세스보다 높이면 시스템의 효율이 향상



(a) CPU 집중 프로세스 우선 배정

(b) 입출력 집중 프로세스 우선 배정

그림 4-6 CPU 집중 프로세스와 입출력 집중 프로세스의 우선 배정 결과

2-4 전면 프로세스와 후면 프로세스

전면 프로세스

- GUI를 사용하는 운영체제에서 화면의 맨 앞에 놓인 프로세스
- 현재 입력과 출력을 사용하는 프로세스
- 사용자와 상호작용이 가능하여 상호작용 프로세스라고도 함

후면 프로세스

- 사용자와 상호작용이 없는 프로세스
- 사용자의 입력 없이 작동하기 때문에 일괄 작업 프로세스라고도 함
- 전면 프로세스의 우선순위가 후면 프로세스보다 높음

2-4 전면 프로세스와 후면 프로세스

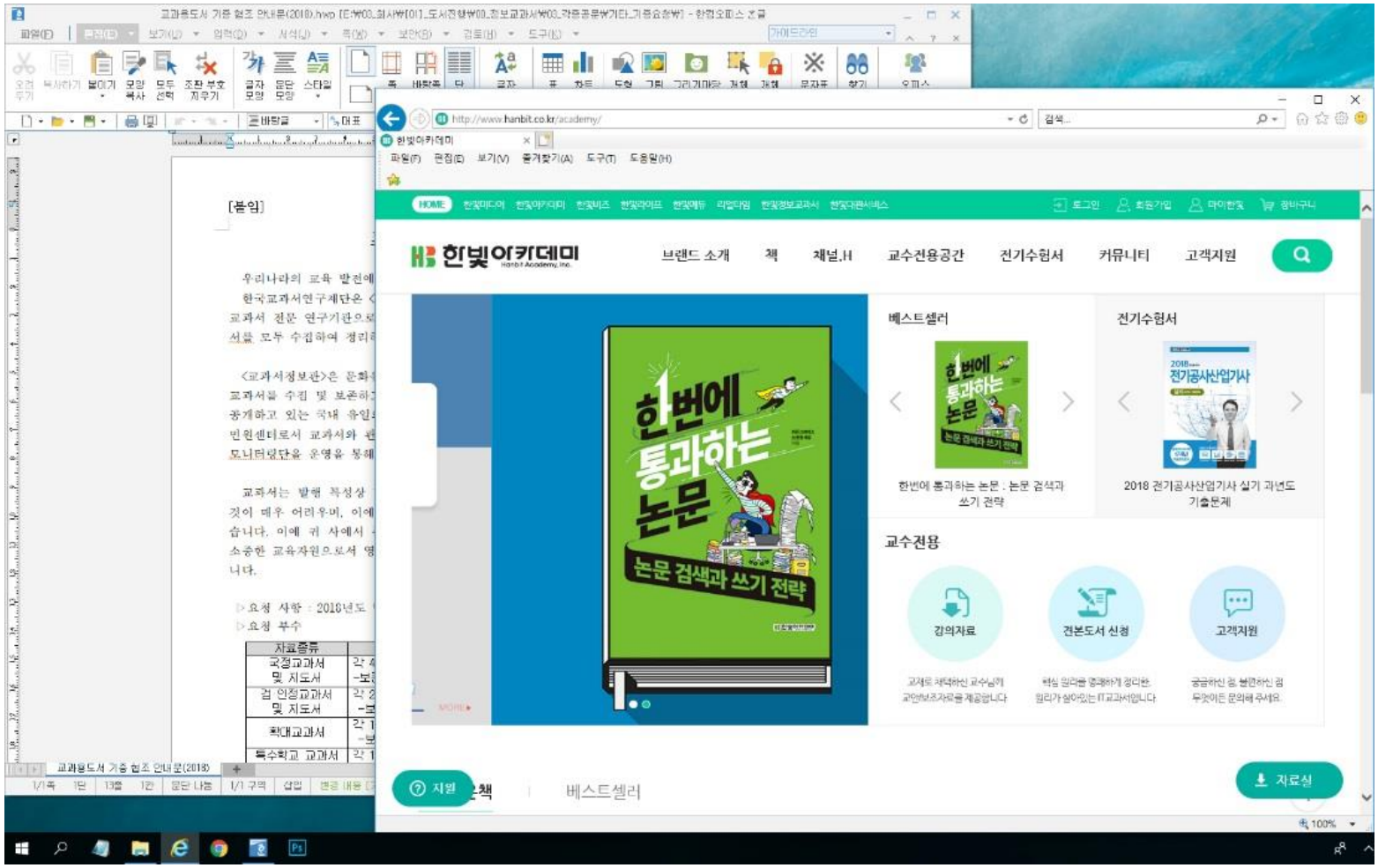


그림 4-7 **전면 프로세스와 후면 프로세스의 예**

2-5 정리

CPU 스케줄링 시 고려사항

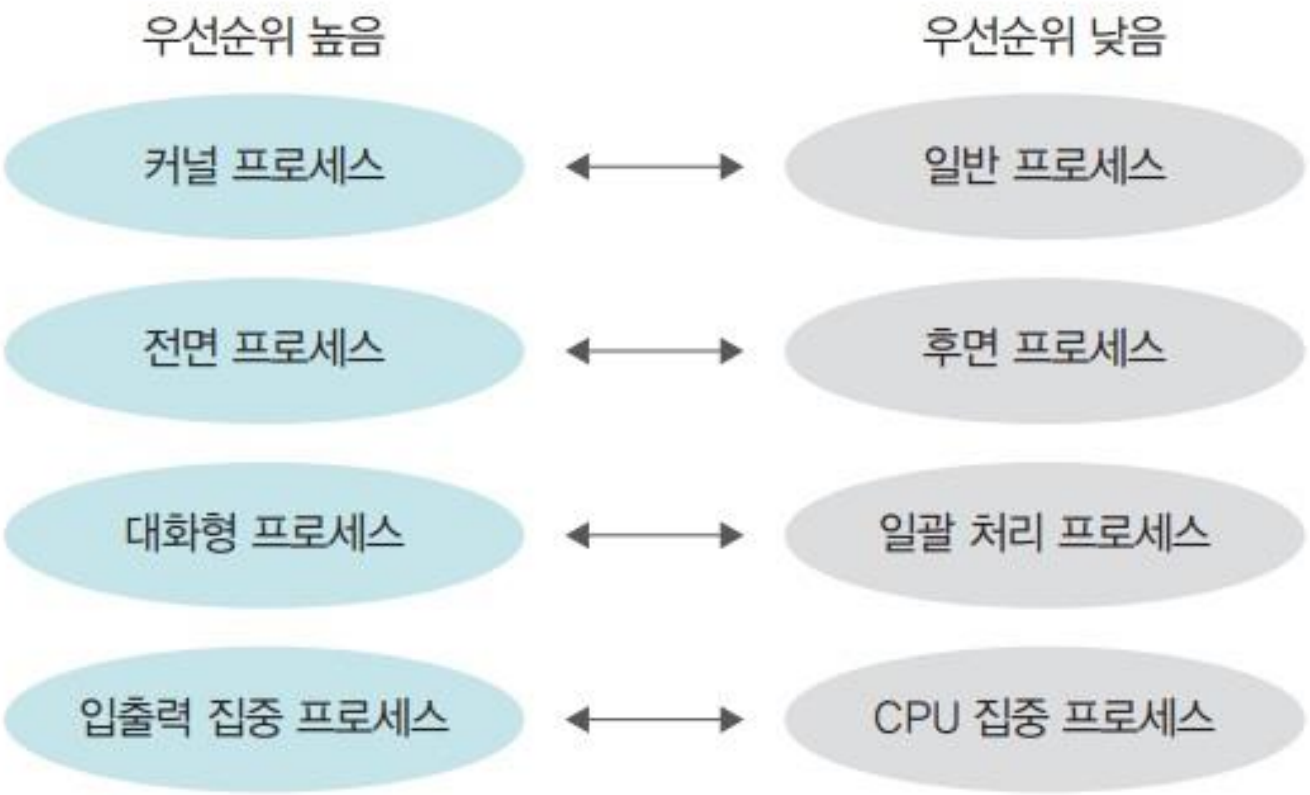


그림 4-8 CPU 스케줄링 시 고려 사항

3-1 준비 상태의 다중 큐

준비 상태의 다중 큐

- 프로세스는 준비 상태에 들어올 때마다 자신의 우선순위에 해당하는 큐의 마지막에 삽입
- CPU 스케줄러는 우선순위가 가장 높은 큐(0번 큐)의 맨 앞에 있는 프로세스 6에 CPU 할당

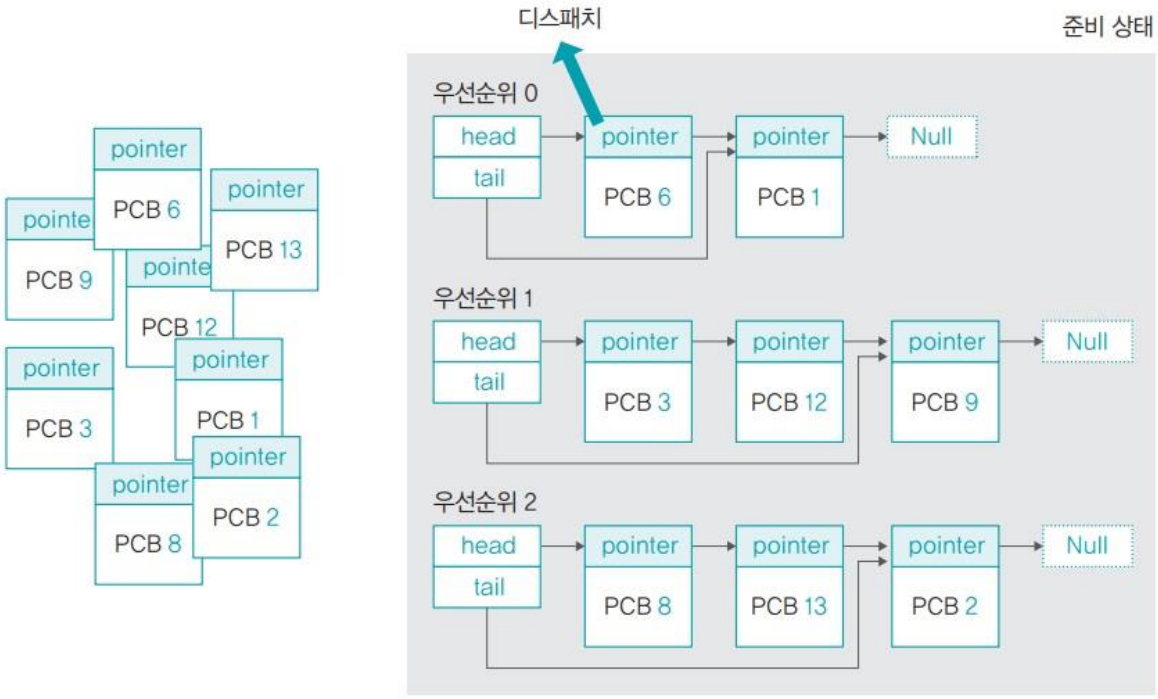


그림 4-9 **준비 상태의 다중 큐**

3-1 준비 상태의 다중 큐

프로세스의 우선순위를 배정하는 방식

- 고정 우선순위 방식
  - 운영체제가 프로세스에 우선순위를 부여하면 프로세스가 끝날 때까지 바뀌지 않는 방식
  - 프로세스가 작업하는 동안 우선순위가 변하지 않기 때문에 구현하기 쉽지만, 시스템의 상황이 시시각각 변하는데 우선순위를 고정하면 시스템의 변화에 대응하기 어려워 작업 효율이 떨어짐

- 변동 우선순위 방식
  - 프로세스 생성 시 부여받은 우선순위가 프로세스 작업 중간에 변하는 방식
  - 구현하기 어렵지만 시스템의 효율성을 높일 수 있음

3-2 대기 상태의 다중 큐

대기 상태의 다중 큐

- 시스템의 효율을 높이기 위해 대기 상태에서는 같은 입출력을 요구한 프로세스끼리 모아 놓음

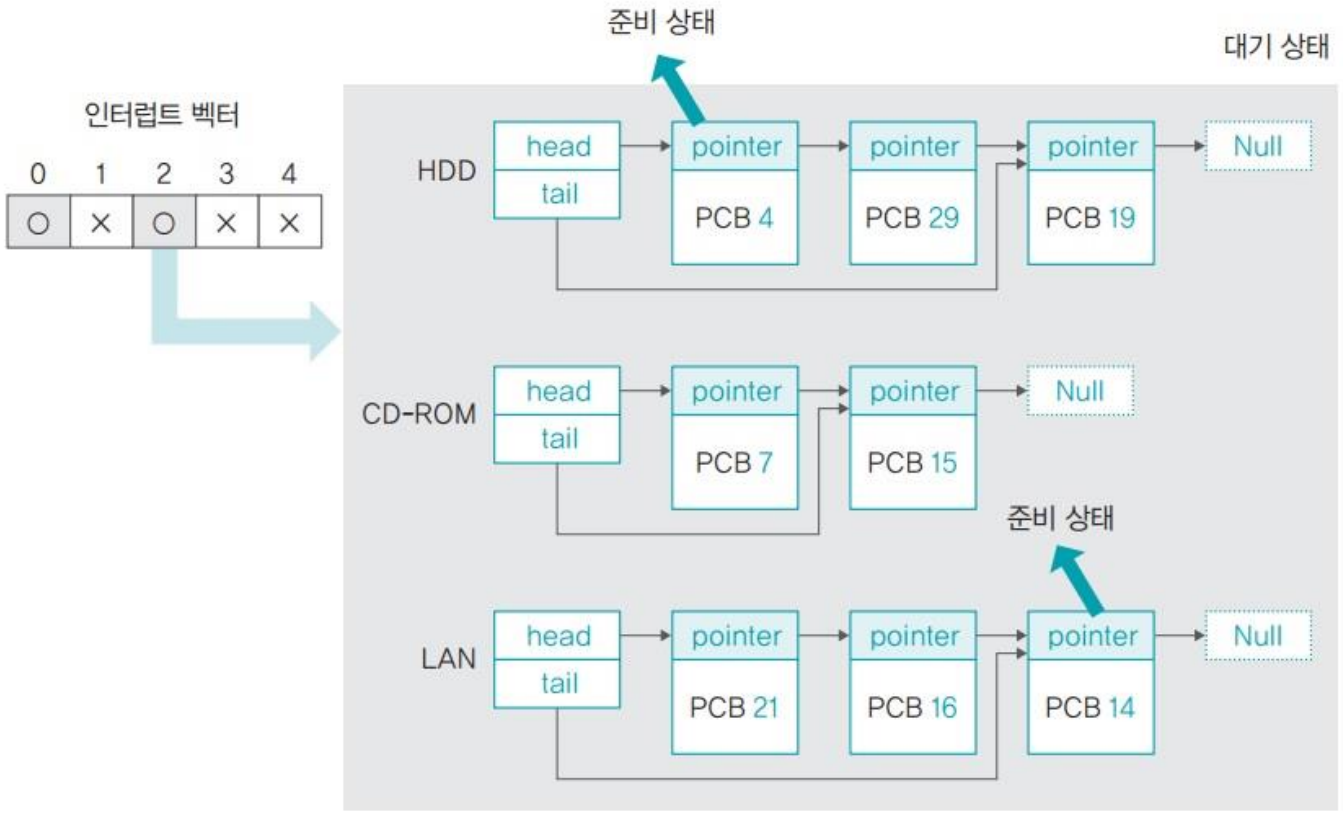


그림 4-10 대기 상태의 다중 큐

3-2 대기 상태의 다중 큐

다중 큐 비교

- 준비 큐
  - 한 번에 하나의 프로세스를 꺼내어 CPU를 할당
- 대기 큐
  - 여러 개의 프로세스 제어 블록을 동시에 꺼내어 준비 상태로 옮김
  - 대기 큐에서 동시에 끝나는 인터럽트를 처리하기 위해 인터럽트 벡터라는 자료 구조 사용

3-2 대기 상태의 다중 큐

다중 큐

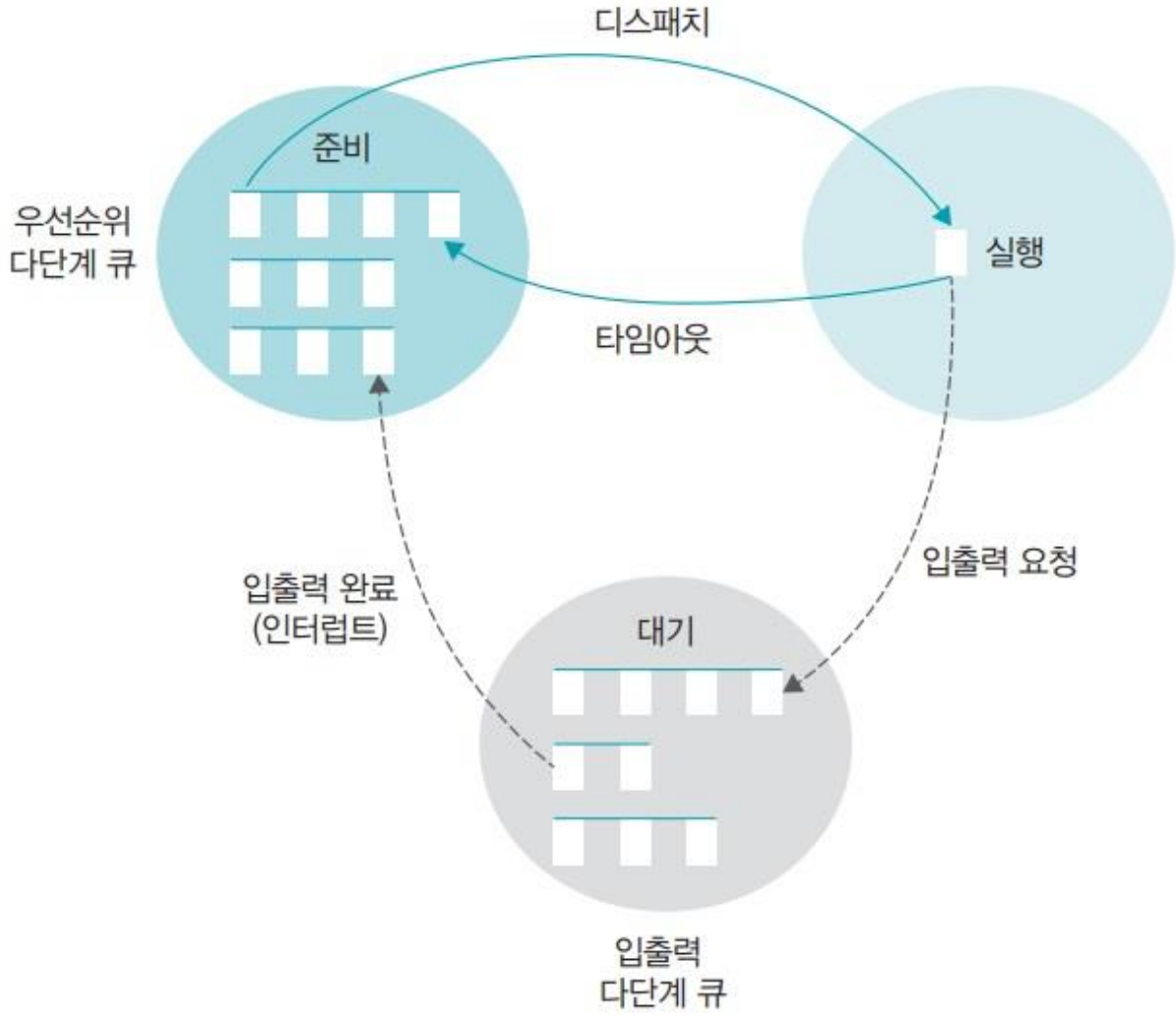


그림 4-11 **프로세스 상태와 다중 큐**

4-1 스케줄링 알고리즘의 선택 기준

표 4-2 **스케줄링 알고리즘의 종류**

구분	종류
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

4-1 스케줄링 알고리즘의 선택 기준

스케줄링 알고리즘의 평가 기준

- CPU 사용률
  - 전체 시스템의 동작 시간 중 CPU가 사용된 시간을 측정하는 방법
  - 가장 이상적인 수치는 100%이지만 실제로는 여러 가지 이유로 90%에도 못 미침
- 처리량
  - 단위 시간당 작업을 마친 프로세스의 수
  - 이 수치가 클수록 좋은 알고리즘임

4-1 스케줄링 알고리즘의 선택 기준

스케줄링 알고리즘의 평가 기준

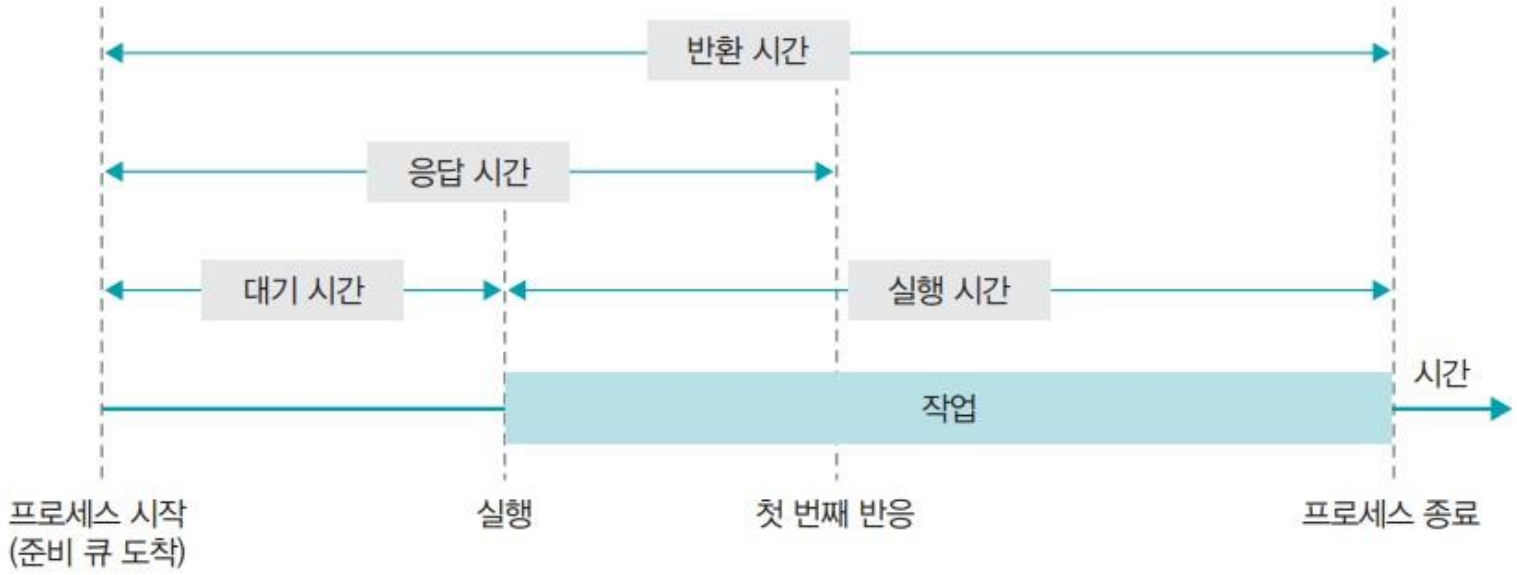


그림 4-12 대기 시간, 응답 시간, 실행 시간, 반환 시간의 관계

- **대기 시간** : 프로세스가 생성된 후 실행되기 전까지 대기하는 시간
- **응답 시간** : 첫 작업을 시작한 후 첫 번째 출력(반응)이 나오기까지의 시간
- **실행 시간** : 프로세스 작업이 시작된 후 종료되기까지의 시간
- **반환 시간** : 대기 시간을 포함하여 실행이 종료될 때까지의 시간

4-1 스케줄링 알고리즘의 선택 기준

평균 대기 시간

- 모든 프로세스의 대기 시간을 합한 뒤 프로세스의 수로 나눈 값

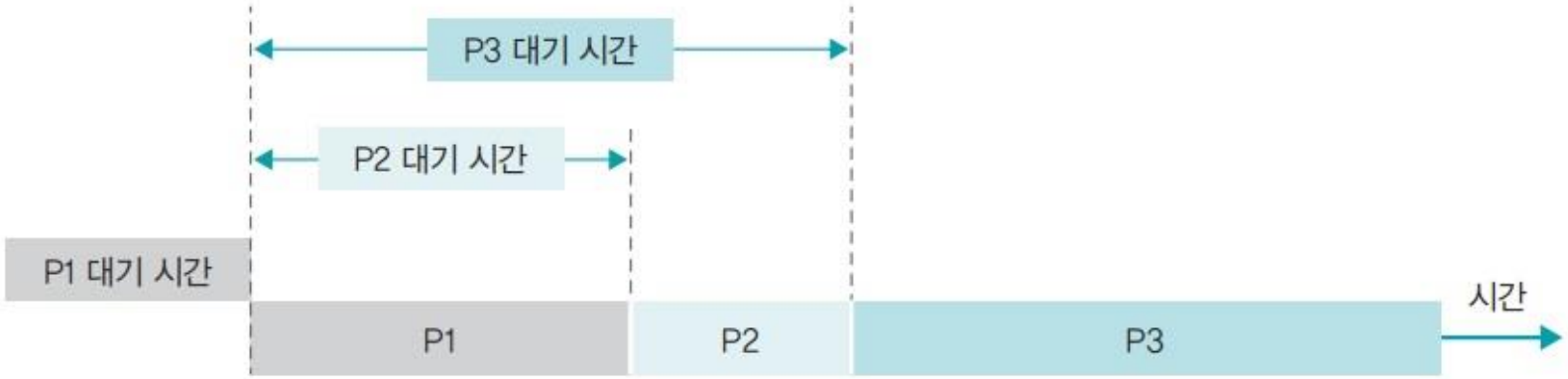


그림 4-13 **평균 대기 시간**

4-2 FCFS 스케줄링

FCFS 스케줄링의 동작 방식

- 준비 큐에 도착한 순서대로 CPU를 할당하는 비선점형 방식
- 한 번 실행되면 그 프로세스가 끝나야만 다음 프로세스를 실행할 수 있음
- 큐가 하나라 모든 프로세스는 우선순위가 동일

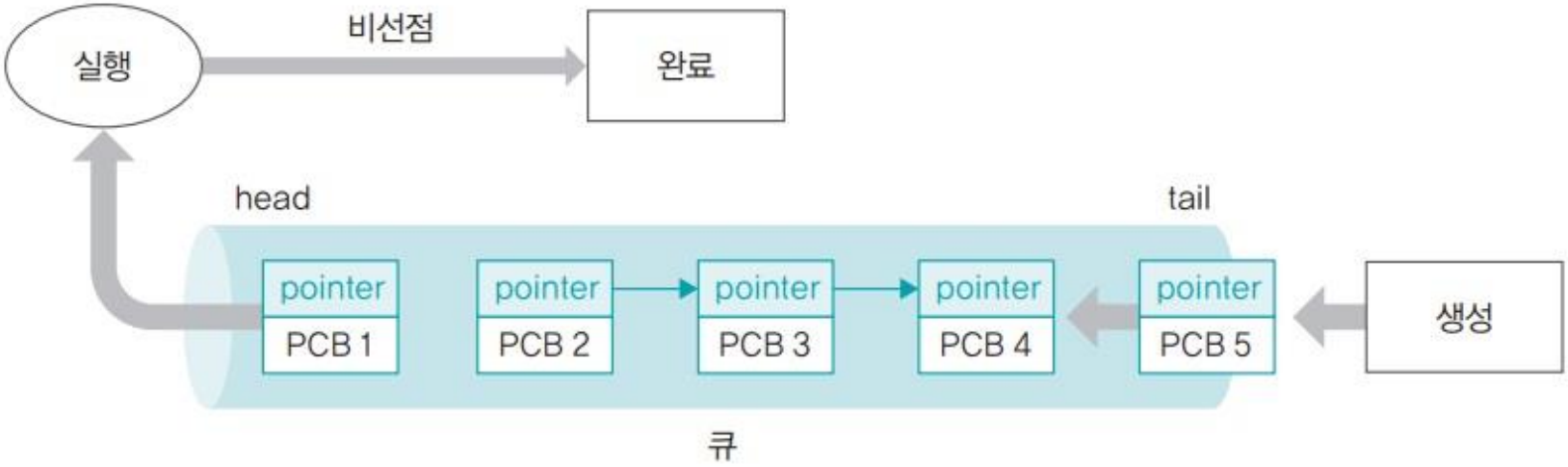


그림 4-14 FCFS 스케줄링의 동작

4-2 FCFS 스케줄링

FCFS 스케줄링의 성능

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

※ 평균 대기 시간
(0+27+42)÷3=23밀리초

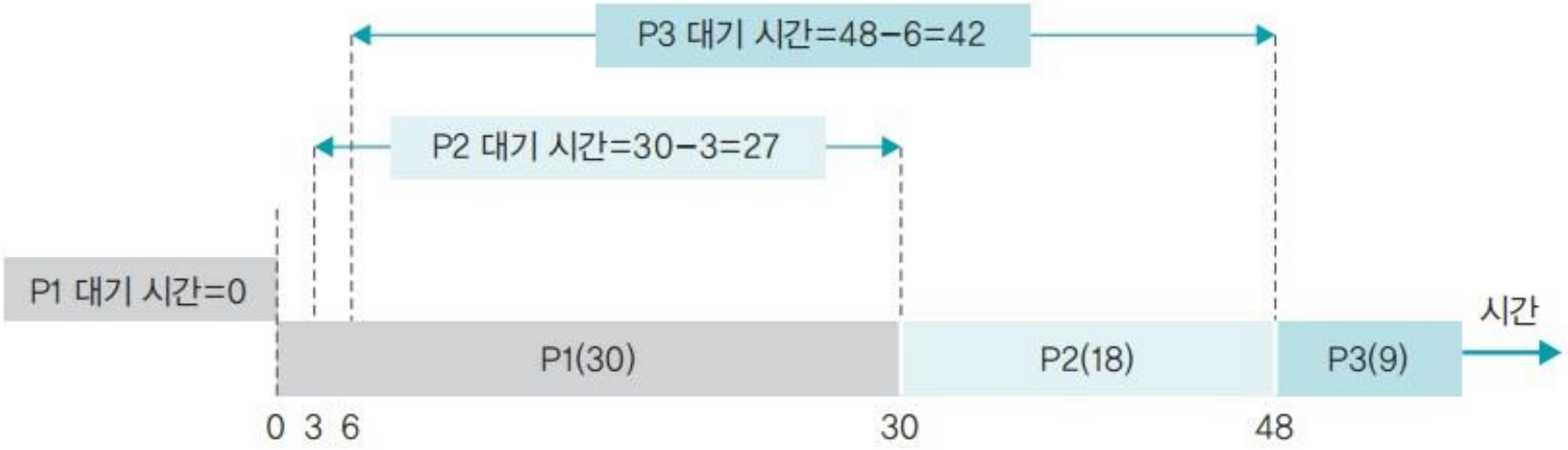


그림 4-15 FCFS 스케줄링의 평균 대기 시간

4-2 FCFS 스케줄링

FCFS 스케줄링의 평가

- 처리 시간이 긴 프로세스가 CPU를 차지하면 다른 프로세스들은 하염없이 기다려 시스템의 효율성이 떨어짐
- 특히 현재 작업 중인 프로세스가 입출력 작업을 요청하는 경우 CPU가 작업하지 않고 쉬는 시간이 많아져 작업 효율이 떨어짐

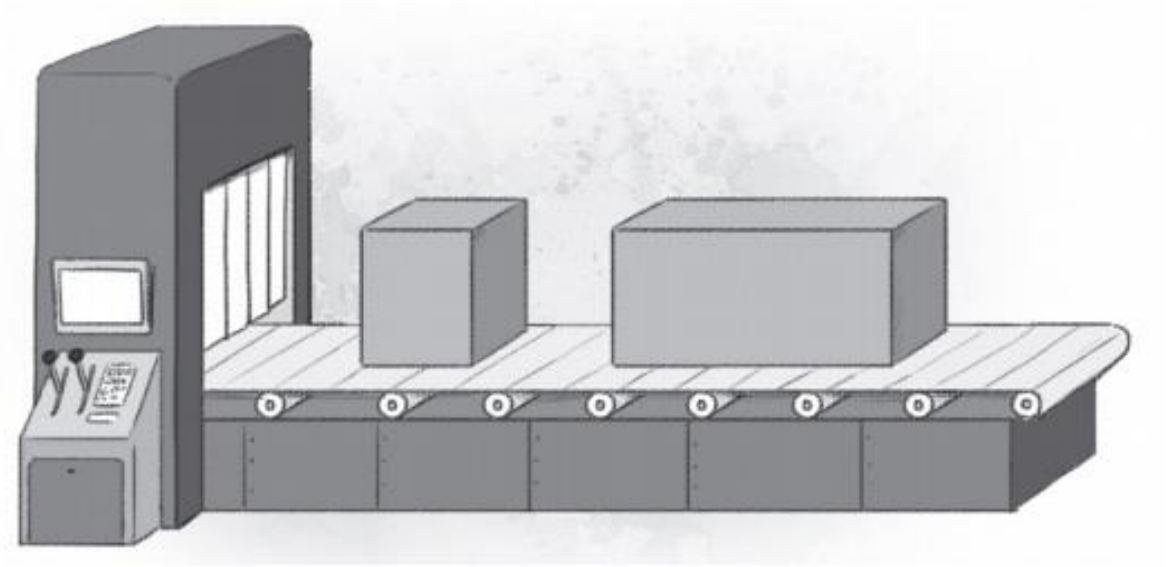


그림 4-16 콘보이 효과

4-3 SJF 스케줄링

SJF 스케줄링의 동작 방식

- 준비 큐에 있는 프로세스 중에서 실행 시간이 가장 짧은 작업부터 CPU를 할당하는 비선점형 방식
- 최단 작업 우선 스케줄링이라고도 함
- 콘보이 효과를 완화하여 시스템의 효율성을 높임

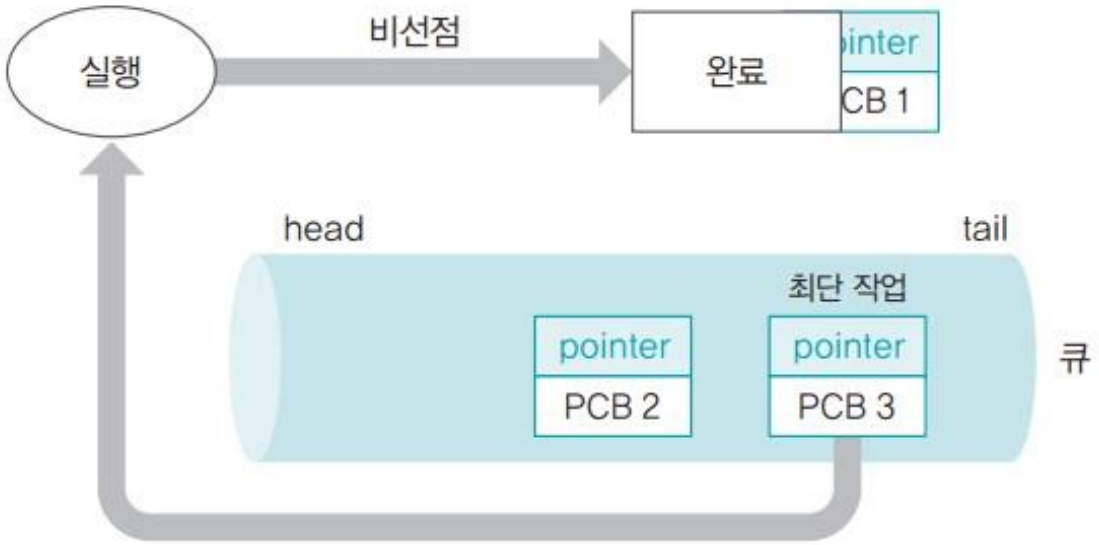


그림 4-17 SJF 스케줄링의 동작

4-3 SJF 스케줄링

SJF 스케줄링의 성능

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

※ 평균 대기 시간
(0+24+36)÷3=20밀리초

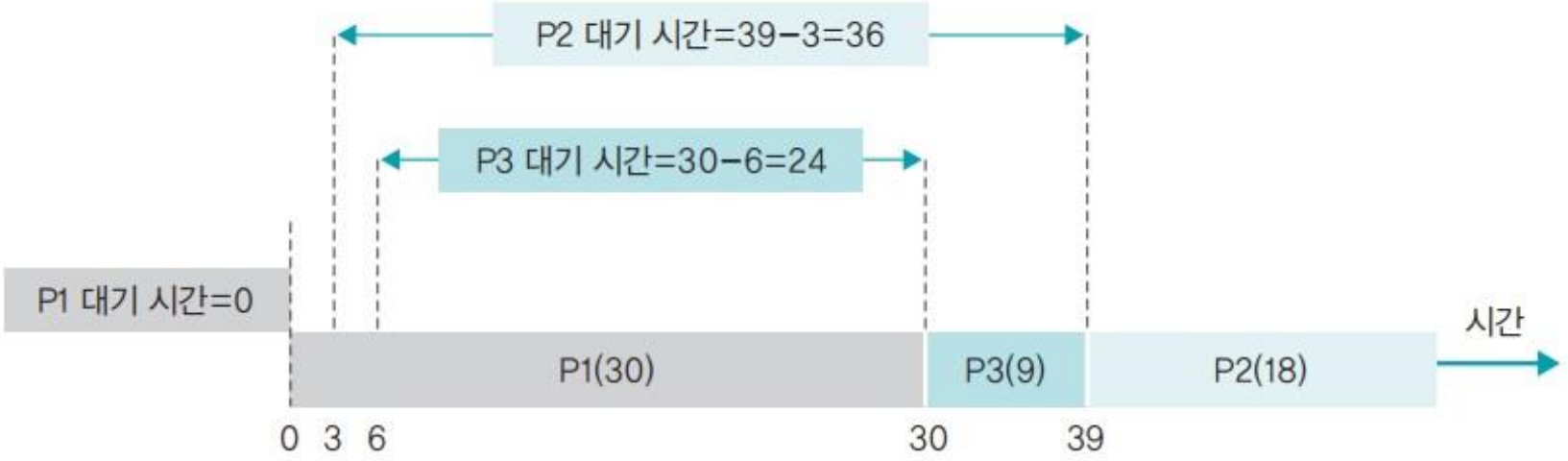


그림 4-18 SJF 스케줄링의 평균 대기 시간

4-3 SJF 스케줄링

SJF 스케줄링의 평가

- 운영체제가 프로세스의 종료 시간을 정확하게 예측하기 어려움
- 작업 시간이 길다는 이유만으로 계속 뒤로 밀려 공평성이 현저히 떨어짐. 이를 아사(starvation) 현상 이라 부름

에이징(나이 먹기)

- 아사 현상의 완화 방법
- 프로세스가 양보할 수 있는 상한선을 정하는 방식
- 프로세스가 자신의 순서를 양보할 때마다 나이를 한 살씩 먹어 최대 몇 살까지 양보하도록 규정하는 것

4-4 HRN 스케줄링

HRN 스케줄링의 동작 방식

- SJF 스케줄링에서 발생할 수 있는 아사 현상을 해결하기 위해 만들어진 비선점형 알고리즘
- 최고 응답률 우선 스케줄링이라고도 함
- 서비스를 받기 위해 기다린 시간과 CPU 사용 시간을 고려하여 스케줄링을 하는 방식
- 프로세스의 우선순위를 결정하는 기준

$$\text{우선순위} = \frac{\text{대기 시간} + \text{CPU 사용 시간}}{\text{CPU 사용 시간}}$$

4-4 HRN 스케줄링

HRN 스케줄링의 성능

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

※ 평균 대기 시간
(0+24+36)÷3=20밀리초

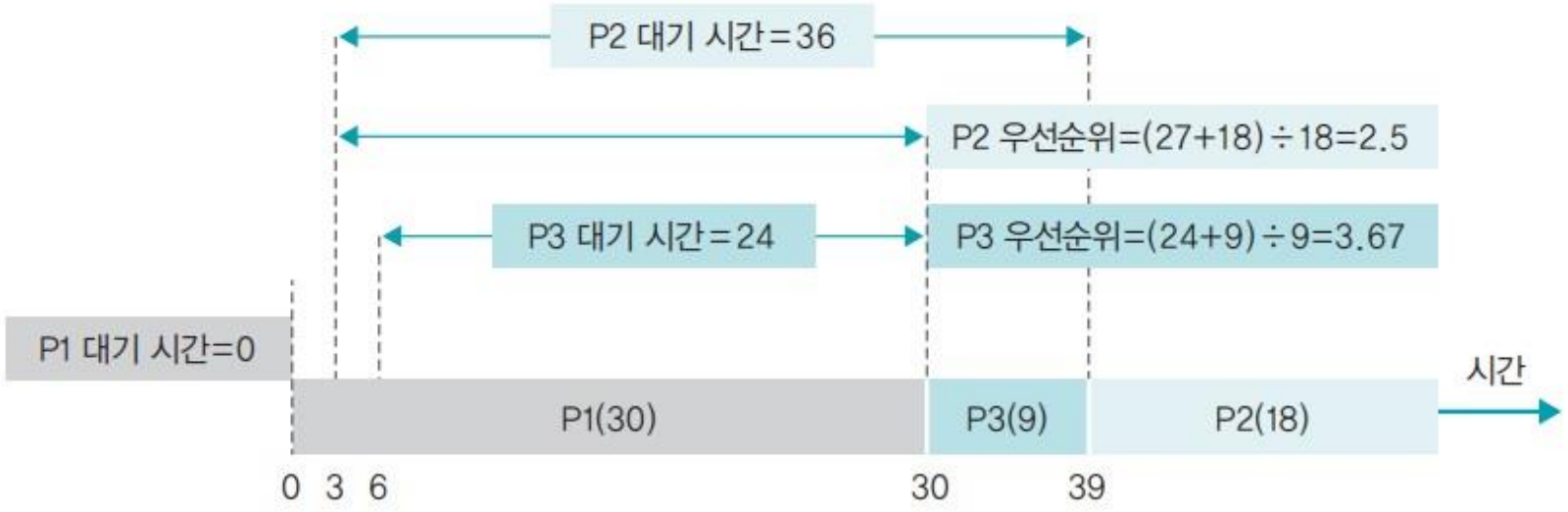


그림 4-19 HRN 스케줄링의 평균 대기 시간

4-4 HRN 스케줄링

HRN 스케줄링의 평가

- 실행 시간이 짧은 프로세스의 우선순위를 높게 설정하면서도 대기 시간을 고려하여 아사 현상을 완화
- 대기 시간이 긴 프로세스의 우선순 위를 높임으로써 CPU를 할당받을 확률을 높임
- 여전히 공평성이 위배되어 많이 사용되지 않음

4-5 라운드 로빈 스케줄링

라운드 로빈 스케줄링의 동작 방식

- 한 프로세스가 할당받은 시간(타임 슬라이스) 동안 작업을 하다가 작업을 완료하지 못하면 준비 큐의 맨 뒤로 가서 자기 차례를 기다리는 방식
- 선점형 알고리즘 중 가장 단순하고 대표적인 방식
- 프로세스들이 작업을 완료할 때까지 계속 순환하면서 실행

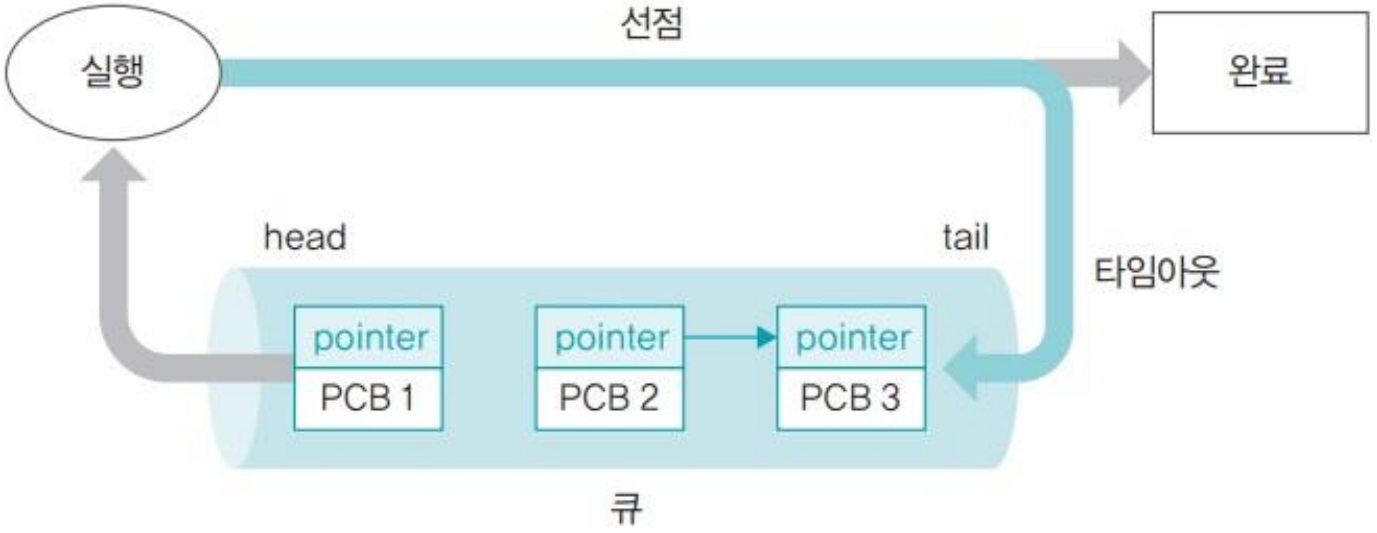


그림 4-20 라운드 로빈 스케줄링의 동작

4-5 라운드 로빈 스케줄링

라운드 로빈 스케줄링의 성능

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

※ 총 대기 시간
0(P1)+7(P2)+14(P3)+19(P1)+19(P2)+8(P1)=67밀리초

※ 평균 대기 시간
67÷3=22.33밀리초

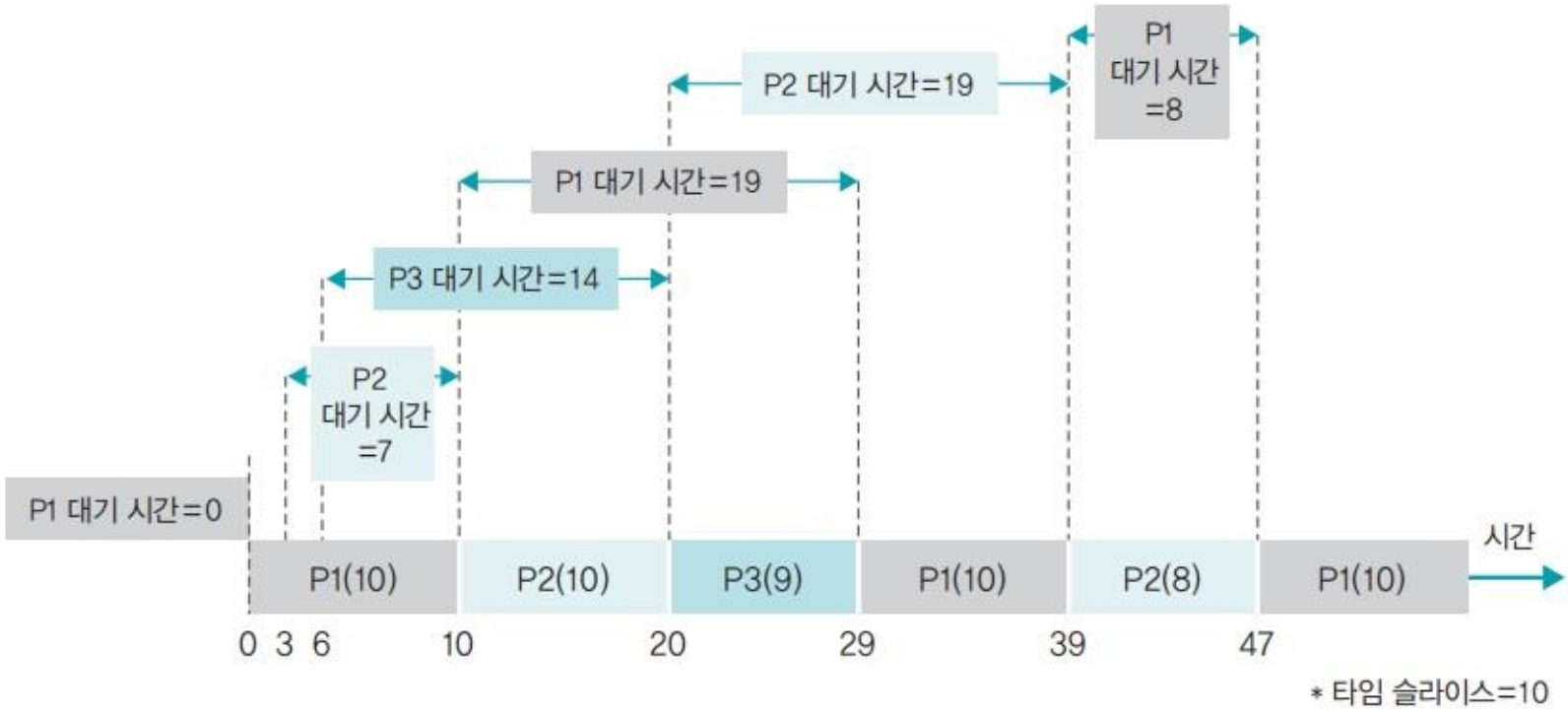


그림 4-21 라운드 로빈 스케줄링의 평균 대기 시간

4-5 라운드 로빈 스케줄링

타임 슬라이스의 크기와 문맥 교환

- 라운드 로빈 스케줄링이 효과적으로 작동하려면 문맥 교환에 따른 추가 시간을 고려하여 타임 슬라이스를 적절히 설정해야 함

타임 슬라이스가 큰 경우

- 하나의 작업이 끝난 뒤 다음 작업이 시작되는 것처럼 보여 FCFS 스케줄링과 다를 게 없음

타임 슬라이스가 작은 경우

- 문맥 교환이 너무 자주 일어나 문맥 교환에 걸리는 시간이 실제 작업 시간보다 상대적으로 커지며, 문맥 교환에 많은 시간을 낭비하여 실제 작업을 못하는 문제가 발생

4-5 라운드 로빈 스케줄링

정리

- 타임 슬라이스는 되도록 작게 설정하되 문맥 교환에 걸리는 시간을 고려하여 적당한 크기로 하는 것이 중요
- 유닉스 운영체제에서는 타임 슬라이스가 대략 100 밀리초

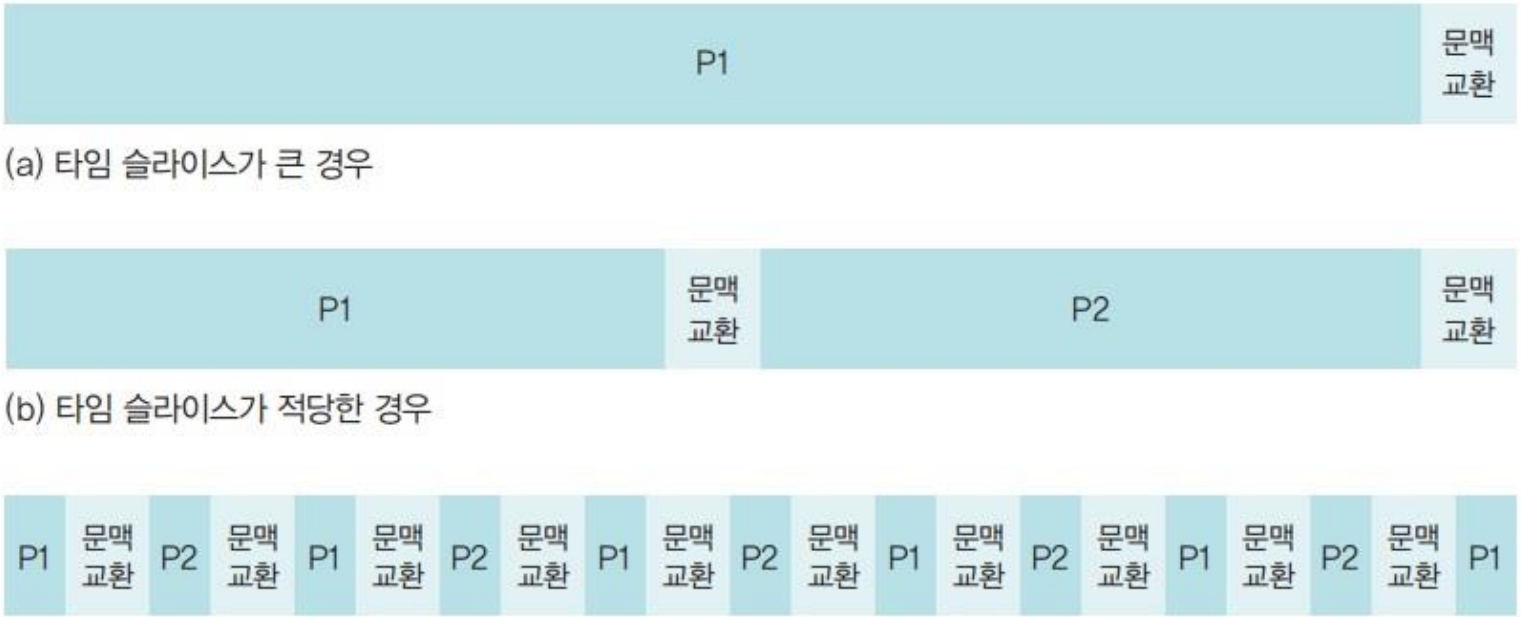


그림 4-22 **타임 슬라이스 크기와 문맥 교환의 관계**

4-6 SRT 우선 스케줄링

SRT 스케줄링의 동작 방식

- 기본적으로 라운드 로빈 스케줄링을 사용하지만, CPU를 할당받을 프로세스를 선택할 때 남아 있는 작업 시간이 가장 적은 프로세스를 선택

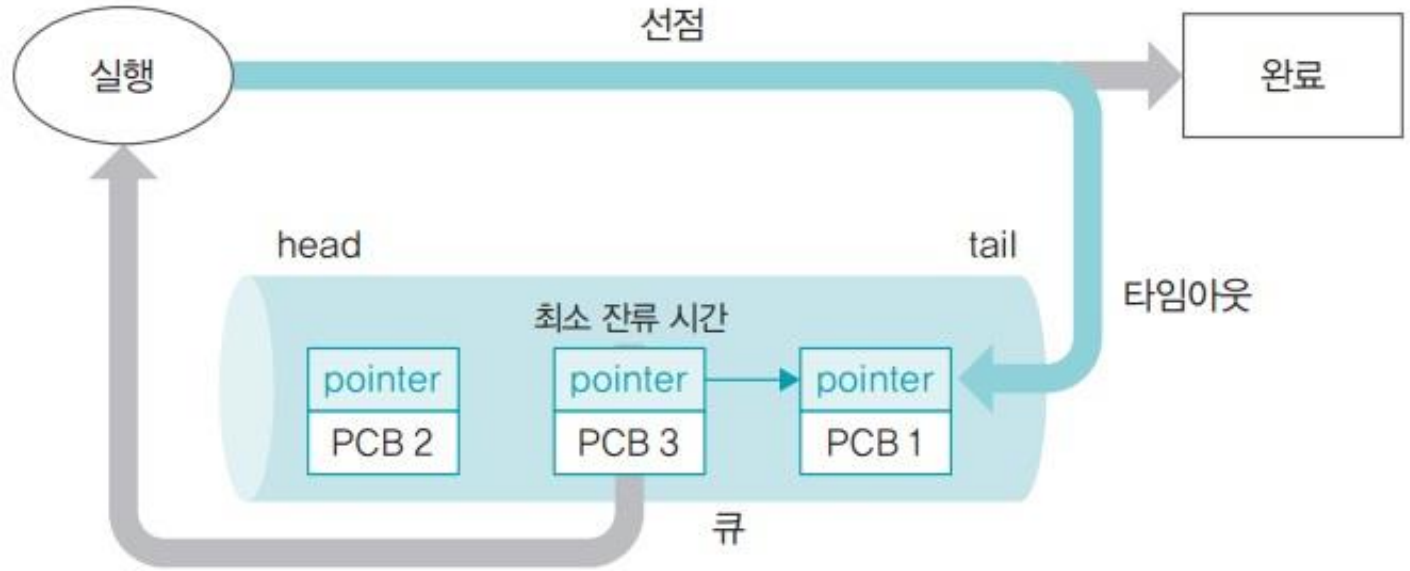


그림 4-23 SRT 스케줄링의 동작

4-6 SRT 우선 스케줄링

SRT 스케줄링의 성능

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

※ 총 대기 시간
0(P1)+4(P3)+16(P2)+27(P1)=47밀리초

※ 평균 대기 시간
47÷3=15.66밀리초

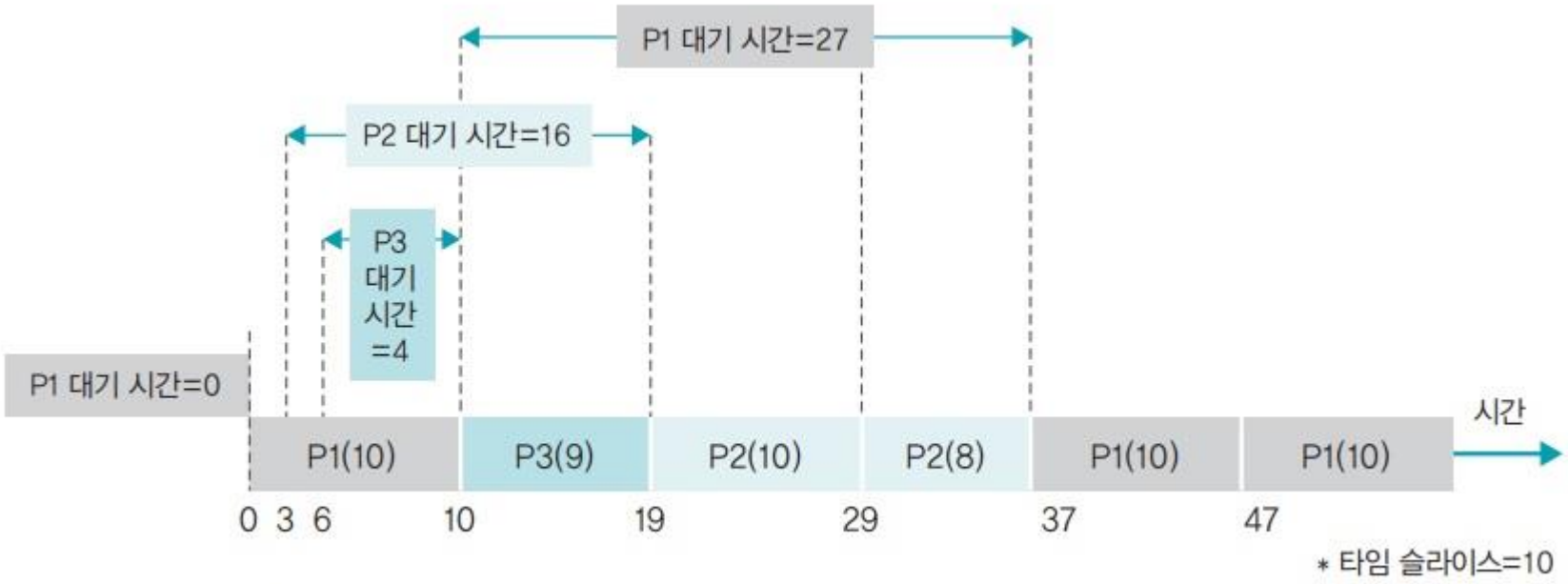


그림 4-24 SRT 스케줄링의 평균 대기 시간

4-6 SRT 우선 스케줄링

SRT 스케줄링의 평가

- 현재 실행 중인 프로세스와 큐에 있는 프로세스의 남은 시간을 주기적으로 계산하고, 남은 시간이 더 적은 프로세스와 문맥 교환을 해야 하므로 SJF 스케줄링에는 없는 작업이 추가됨
- 운영체제가 프로세스의 종료 시간을 예측하기 어렵고 아사 현상이 일어나기 때문에 잘 사용하지 않음

4-7 우선순위 스케줄링

우선순위 스케줄링의 동작 방식

- 프로세스의 중요도에 따른 우선순위를 반영한 스케줄링 알고리즘

표 4-5 프로세스의 우선순위

도착 순서	도착 시간	작업 시간	우선순위
P1	0	30	3
P2	3	18	2
P3	6	9	1

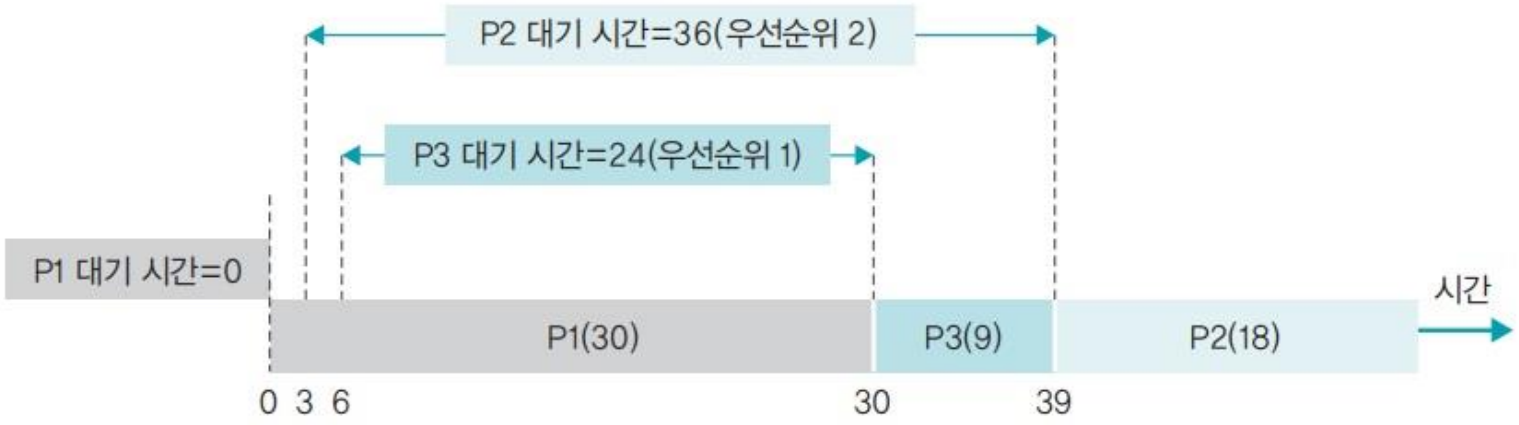


그림 4-25 FCFS 스케줄링에 우선순위를 적용한 결과

4-7 우선순위 스케줄링

우선순위 적용

- 우선순위는 비선점형 방식과 선점형 방식에 모두 적용할 수 있음
  - **(비선점형 방식) SJF 스케줄링** : 작업 시간이 짧은 프로세스에 높은 우선순위를 부여
  - **(비선점형 방식) HRN 스케줄링** : 작업 시간이 짧거나 대기 시간이 긴 프로세스에 높은 우선순위를 부여
  - **(선점형 방식) SRT 스케줄링** : 남은 시간이 짧은 프로세스에 높은 우선순위를 부여

4-7 우선순위 스케줄링

고정 우선순위 알고리즘

- 한 번 우선순위를 부여받으면 종료될 때까지 우선순위가 고정
- 단순하게 구현할 수 있지만 시시각각 변하는 시스템의 상황을 반영하지 못해 효율성이 떨어짐

변동 우선순위 알고리즘

- 일정 시간마다 우선순위가 변하여 일정 시간마다 우선순위를 새로 계산하고 이를 반영
- 복잡하지만 시스템의 상황을 반영하여 효율적인 운영 가능

4-7 우선순위 스케줄링

우선순위 스케줄링의 평가

- 준비 큐에 있는 프로세스의 순서를 무시하고 우선순위가 높은 프로세스에 먼저 CPU를 할당하므로 공평성을 위배하고 아사 현상을 일으킴
- 준비 큐에 있는 프로세스의 순서를 무시하고 프로세스의 우선순위를 매번 바꿔야 하기 때문에 오버헤드가 발생하여 시스템의 효율성을 떨어뜨림

4-8 다단계 큐 스케줄링

다단계 큐 스케줄링의 동작 방식

- 우선순위에 따라 준비 큐를 여러 개 사용하는 방식
- 프로세스는 운영체제로부터 부여받은 우선순위에 따라 해당 우선순위의 큐에 삽입
- 우선순위는 고정형 우선순위를 사용
- 상단의 큐에 있는 모든 프로세스의 작업이 끝나야 다음 우선순위 큐의 작업이 시작됨

4-8 다단계 큐 스케줄링

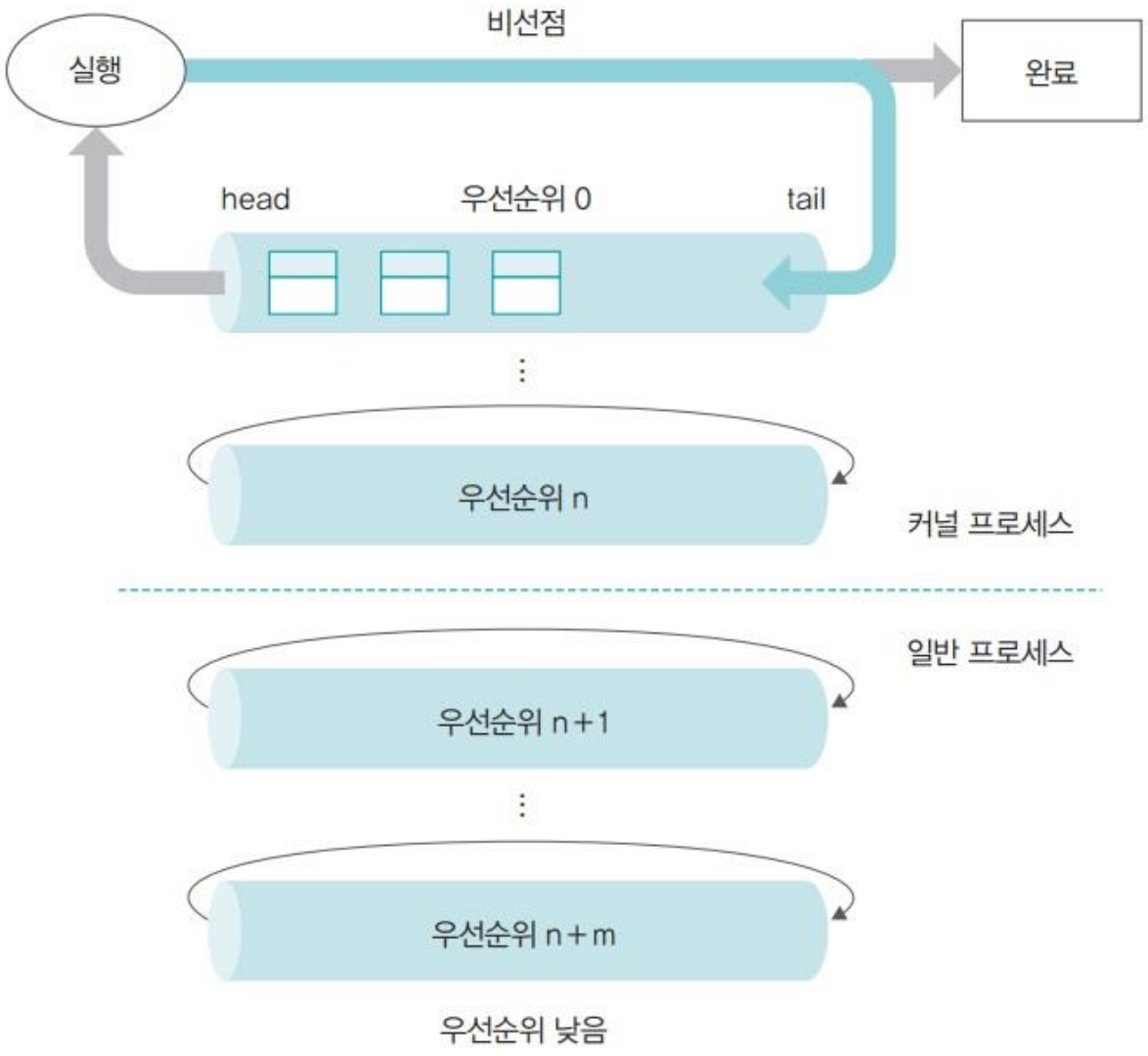


그림 4-26 다단계 큐 스케줄링의 동작

4-9 다단계 피드백 큐 스케줄링

다단계 피드백 큐 스케줄링의 동작 방식

- 프로세스가 CPU를 한 번씩 할당받아 실행될 때마다 프로세스의 우선순위를 낮춤으로써, 다단계 큐에서 우선순위가 낮은 프로세스의 실행이 연기되는 문제를 완화
- 우선순위가 낮아진다고 할지라도 커널 프로세스가 일반 프로세스의 큐에 삽입되지는 않음
- 우선순위에 따라 타임 슬라이스의 크기가 다름
- 우선순위가 낮아질수록 CPU를 얻을 확률이 적어짐. 따라서 한번 CPU를 잡을 때 많이 작업하라고 낮은 우선순위의 타임 슬라이스를 크게 함.
- 마지막 큐에 있는(우선순위가 가장 낮은) 프로세스는 무한대의 타임 슬라이스를 얻음
- 마지막 큐는 들어온 순서대로 작업을 마치는 FCFS 스케줄링 방식으로 동작

4-9 다단계 피드백 큐 스케줄링

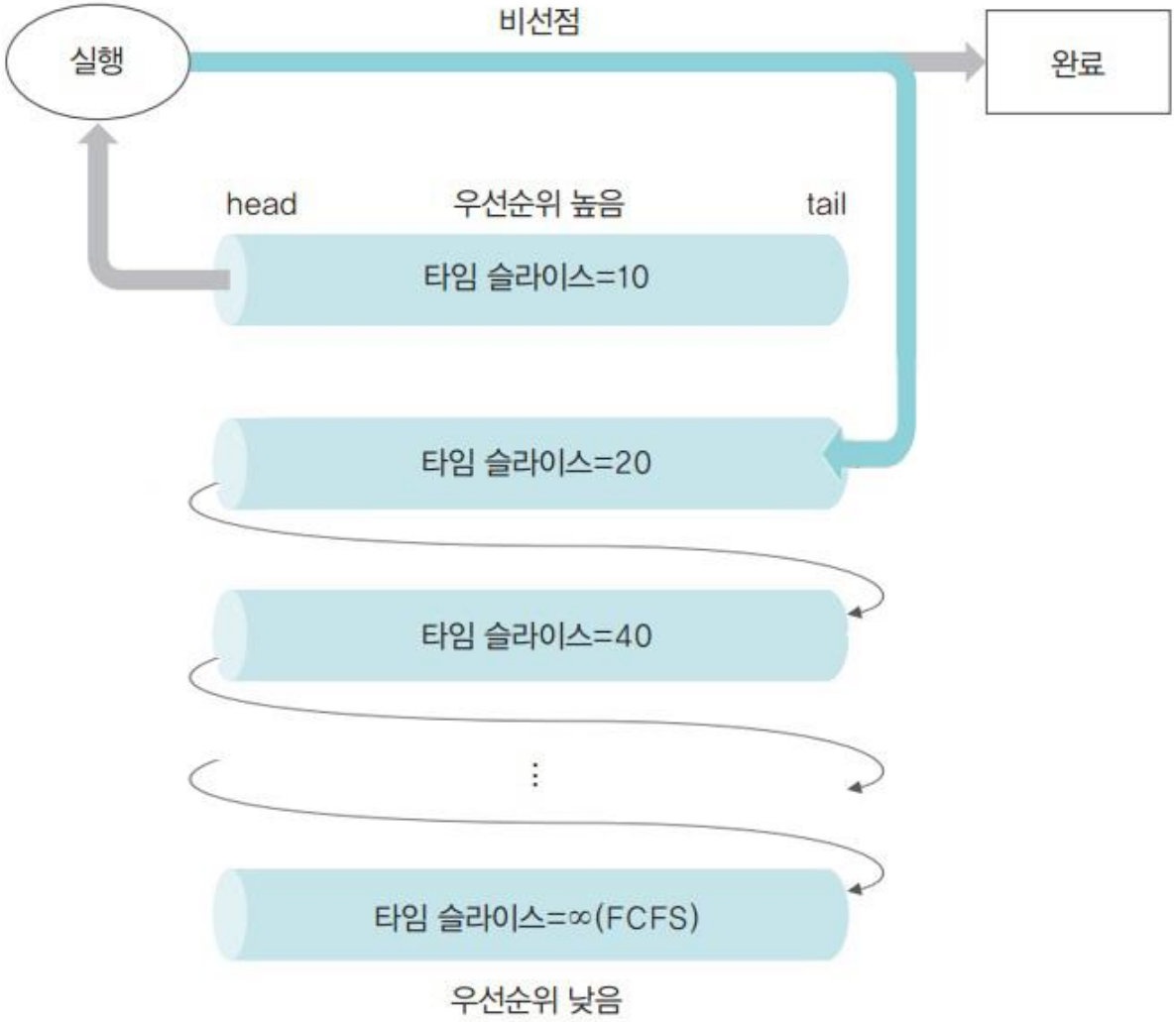


그림 4-27 다단계 피드백 큐 스케줄링의 동작

심화학습 5-1 인터럽트의 개념

폴링

- 입출력을 요청하면 운영체제가 주기적으로 입출력장치를 직접 확인해서 처리하는 방식

인터럽트

- 이벤트 드리븐 방식과 마찬가지로 입출력을 요청하고 입출력이 완료되면 이벤트를 발생시켜 알림

5-2 동기적 인터럽트와 비동기적 인터럽트

동기적 인터럽트

- 프로세스가 실행 중인 명령어로 인해 발생
- 동기적 인터럽트의 종류
  - 프로그램상의 문제 때문에 발생하는 인터럽트(예: 다른 사용자의 메모리 영역에 접근하는 경우, 오버플로나 언더플로에 의해 발생하는 경우 등)
  - 컴퓨터 작업자가 의도적으로 프로세스를 중단하기 위해 발생시킨 인터럽트(예: [Ctrl]+[C])
  - 입출력장치 같은 주변장치의 조작에 의한 인터럽트
  - 산술 연산 중 발생하는 인터럽트(예: 어떤 수를 0으로 나눔)

비동기적 인터럽트

- 하드디스크 읽기 오류, 메모리 불량과 같은 하드웨어적인 오류로 발생
- 사용자가 직접 작동하는 키보드 인터럽트, 마우스 인터럽트 등이 있음

5-3 인터럽트 처리 과정

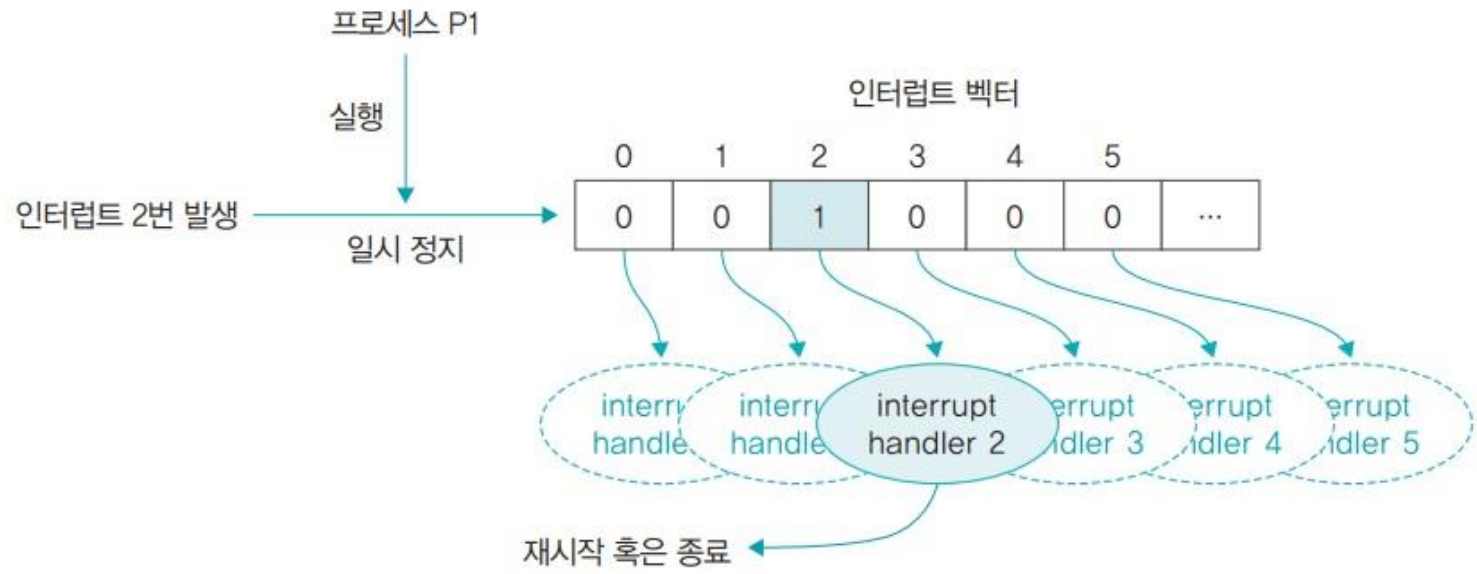


그림 4-30 **인터럽트 처리 과정**

❶ 인터럽트가 발생하면 현재 실행 중인 프로세스는 일시 정지 상태가 되며, 재시작하기 위해 현재 프로세스 관련 정보를 임시로 저장

❷ 인터럽트 컨트롤러가 실행되어 인터럽트의 처리 순서를 결정

❸ 먼저 처리할 인터럽트가 결정되면 인터럽트 벡터에 등록된 인터럽트 핸들러가 실행

❹ 인터럽트 벡터에 연결된 핸들러가 인터럽트 처리를 마치면 일시정지된 프로세스가 다시 실행되거나 종료

5-4 인터럽트와 이중 모드

커널 모드

- 운영체제와 관련된 커널 프로세스가 실행되는 상태

사용자 모드

- 사용자 프로세스가 실행되는 상태

이중 모드

- 운영체제가 커널 모드와 사용자 모드를 전환하며 일 처리를 하는 것
- 궁극적인 목적은 자원 보호에 있음

5-4 인터럽트와 이중 모드

시스템 호출과 API

- 사용자 프로세스가 자원에 접근하려면 시스템 호출을 이용해야 함
- 사용자 프로세스는 API가 준비해놓은 다양한 함수를 이용하여 시스템 자원에 접근

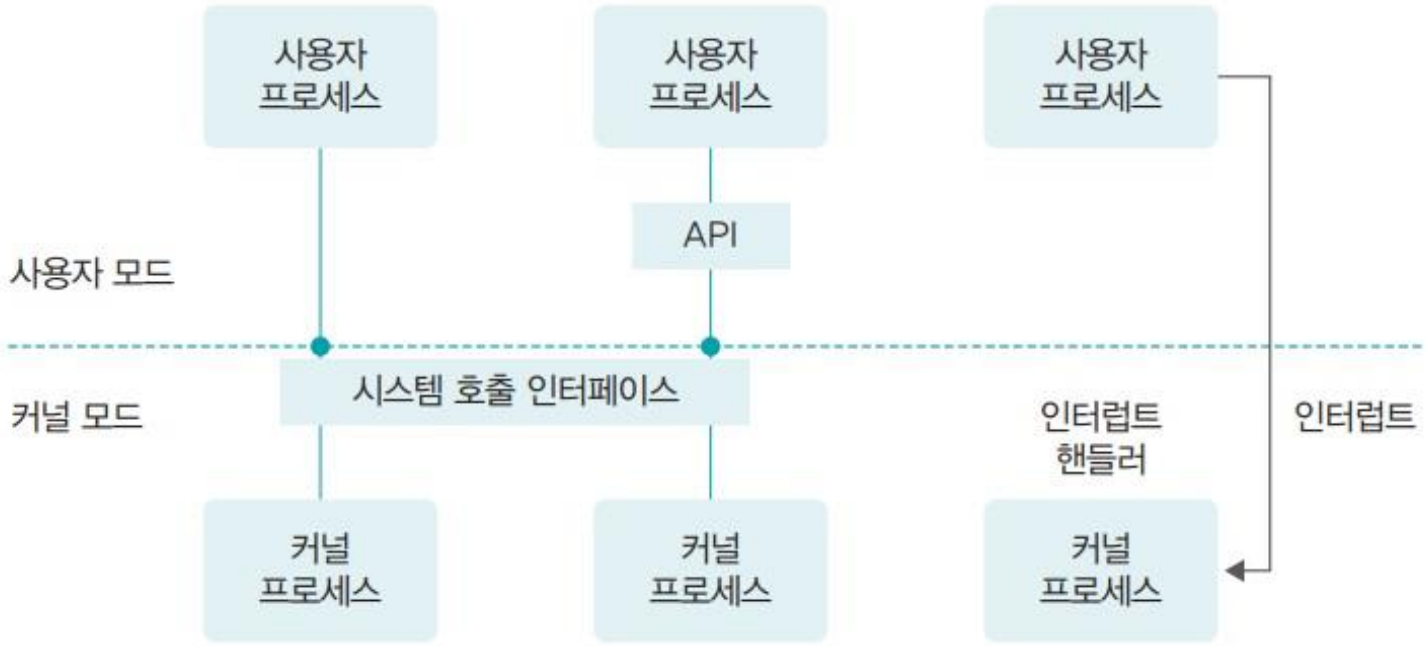


그림 4-31 시스템 호출과 API

Thank You