스마트시스템 운영체제
(LD01600)

김준철

정보시스템공학과

greensday@sungshin.ac.kr

5주차 강의

휴강(9.30)
(추석)

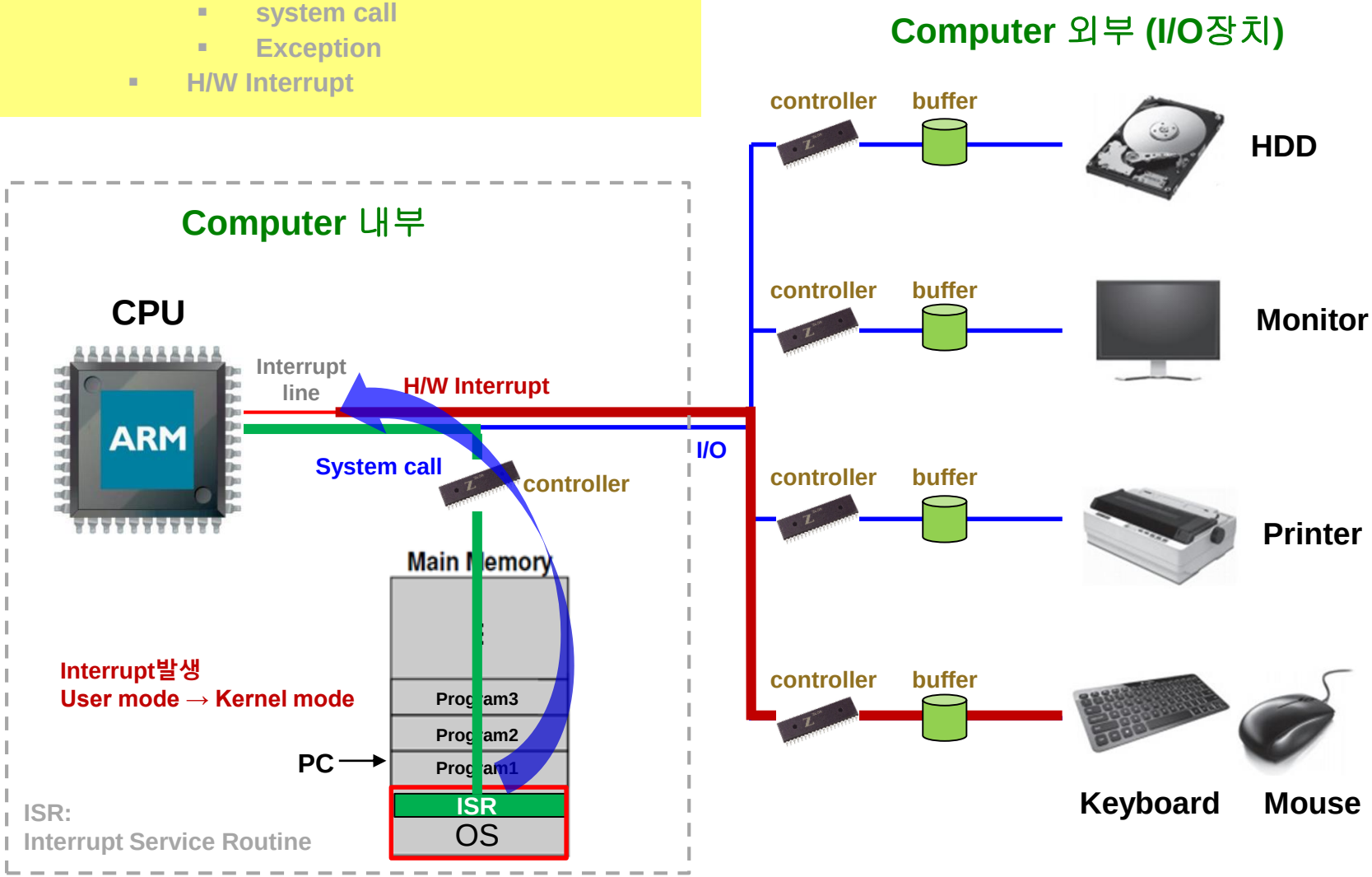
	주차	강의 목차
9.2	1	과목소개 / 운영체제 개요
9.9	2	컴퓨터 시스템 구조
9.16	3	process와 스레드1
9.22	4	process와 스레드2, CPU스케쥴링1
10.7	5	CPU스케쥴링2
10.14	6	process 동기화
10.21	7	교착 상태
10.28	8	중간고사
11.4	9	물리 메모리 관리
11.11	10	가상메모리 기초
11.18	11	가상메모리 관리
11.25	12	입출력시스템1
12.2	13	입출력시스템2, 파일시스템1
12.9	14	파일시스템2
12.16	15	기말고사

Operating Systems

ch.04 CPU scheduling 2

Interrupt

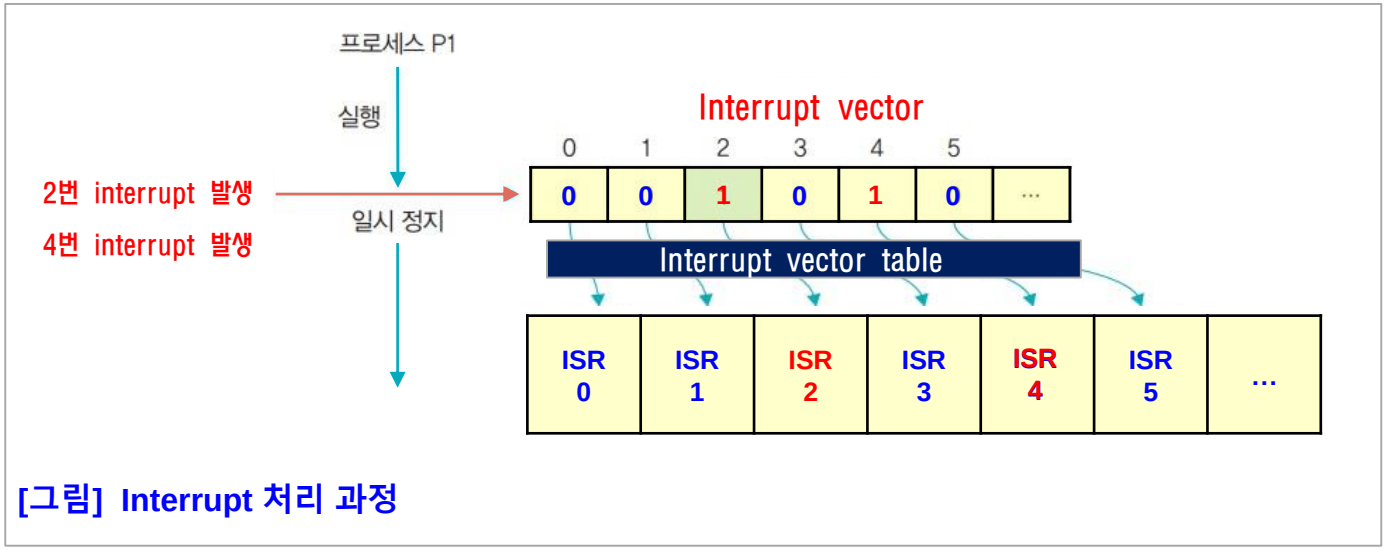
- **Interrupt : CPU 사용 권한이 OS로 넘어감**
  - S/W Interrupt (=Trap)
    - system call
    - Exception
  - H/W Interrupt



Interrupt 처리 과정

- I/O요청의 Interrupt(인터럽트)

(예) 입출력을 요청하고 입출력이 완료되어 인터럽트를 발생시킴



[그림] Interrupt 처리 과정

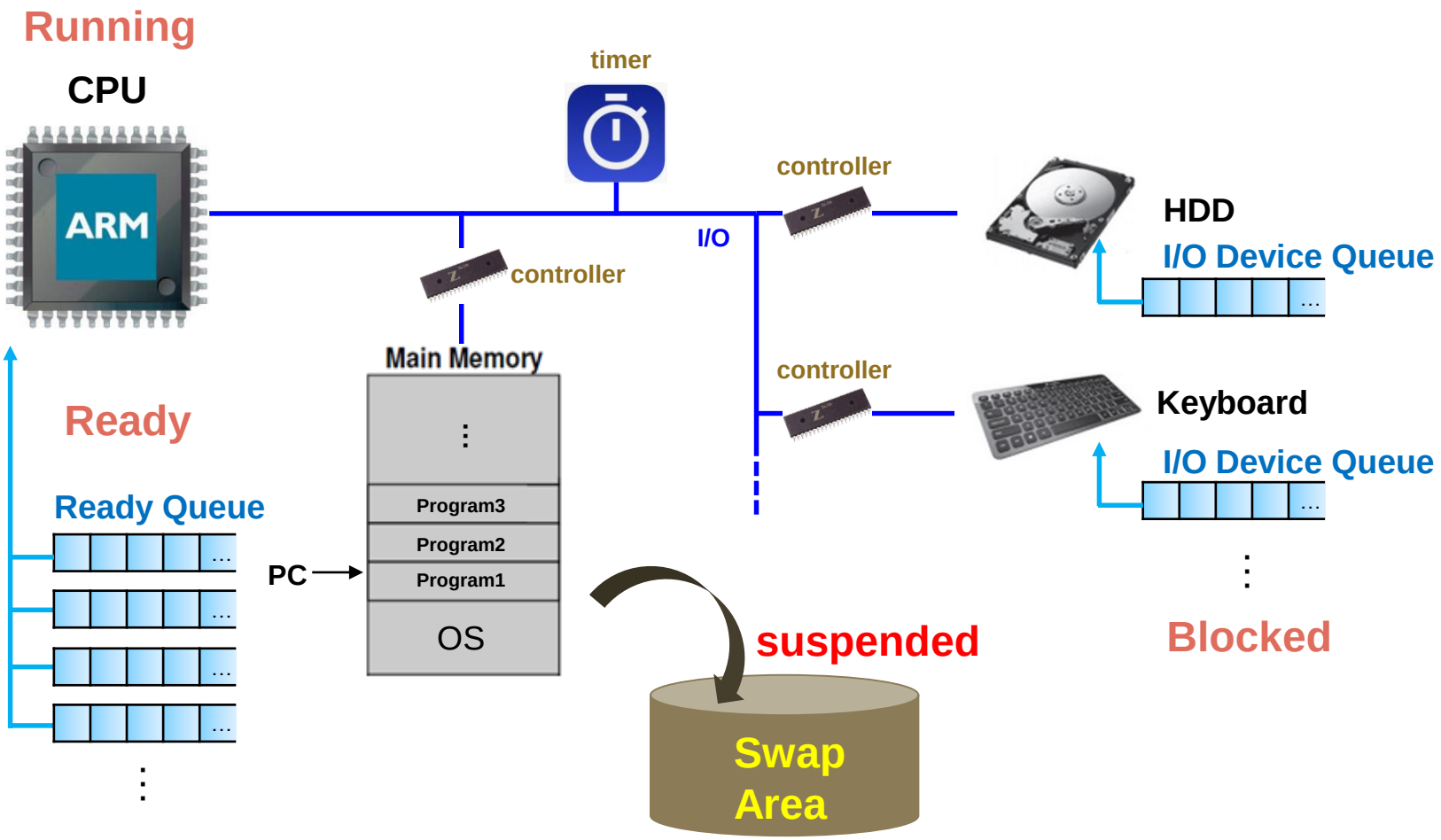
❶ Interrupt가 발생하면 현재 실행 중인 process는 일시 정지 상태가 됨

❷ Interrupt 처리 후 재 시작하기 위해 현재 process 관련 정보를 임시로 저장

❸ 처리할 Interrupt가 결정되면 Interrupt vector에 등록된 Interrupt Service Routine (ISR)가 실행
(인터럽트 벡터, 인터럽트 서비스 루틴)

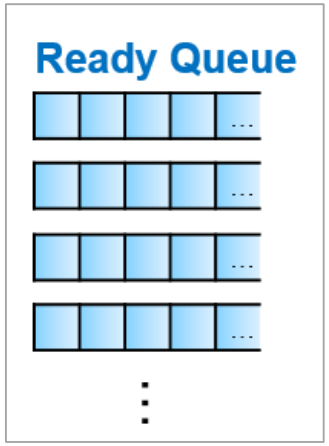
❹ Interrupt vector에 연결된 ISR에 따라 Interrupt 처리를 마치면 일시 정지된 process가 다시 실행됨

Queues


Running
CPU
ARM
timer
controller
I/O
controller
HDD
I/O Device Queue
Main Memory
controller
Keyboard
I/O Device Queue
Ready
Ready Queue
PC
Program3
Program2
Program1
OS
suspended
Swap Area
Blocked

Multilevel Queue(다중 Queue)

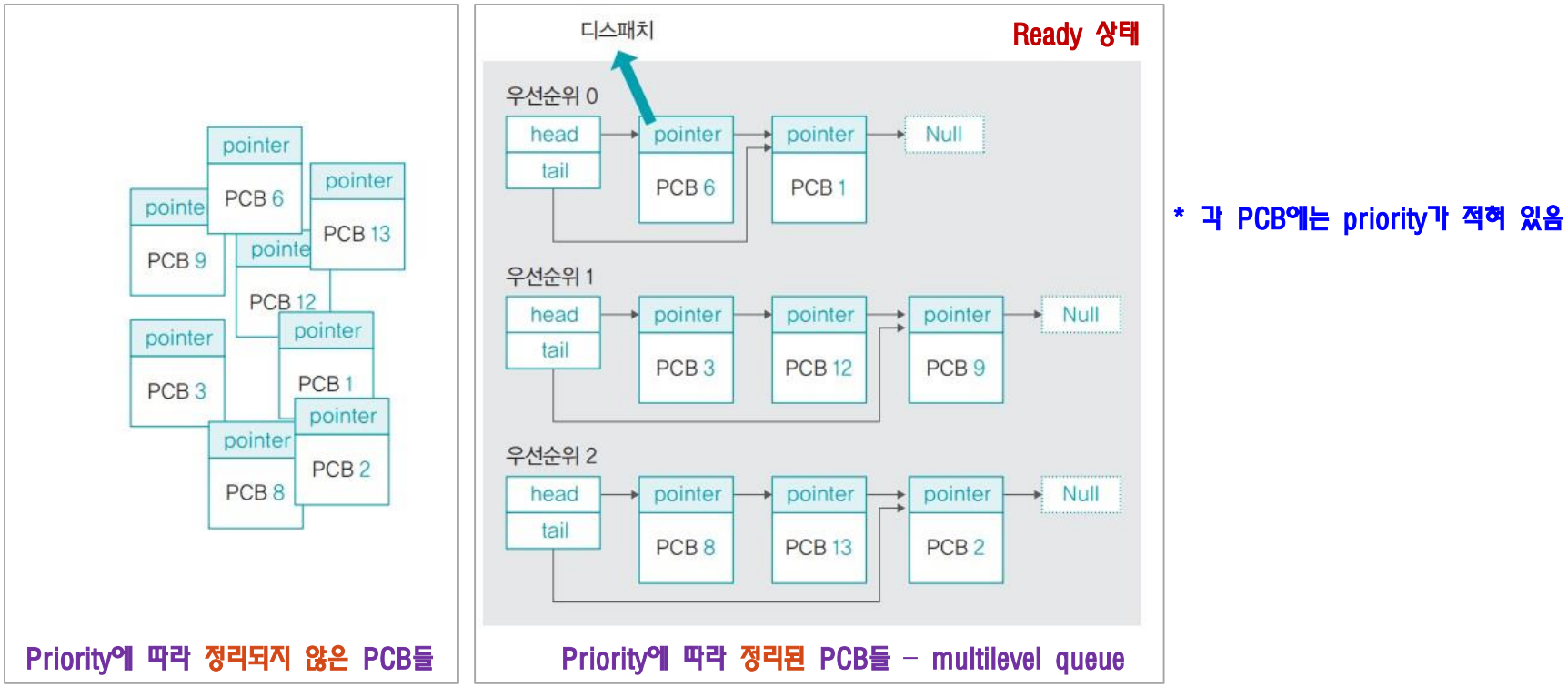
- process의 priority(우선순위)를 배정하는 방식
  - Static priority (고정 우선순위) 방식
    - 운영체제가 process에 priority(우선순위)를 부여하면 **process가 끝날 때까지 바뀌지 않는 방식**
    - process가 작업하는 동안 priority가 변하지 않기 때문에 구현하기 쉽지만, system의 상황이 시시각각 변하는데 priority를 고정하면 **system의 변화에 대응하기 어려워 작업 효율이 떨어짐**
  - Dynamic priority (변동 우선순위) 방식
    - process 생성 시 부여 받은 priority가 **process 작업 중간에 변하는 방식**
    - 구현하기 어렵지만 **system의 효율성을 높일 수 있음**



Multilevel Queue(다중 Queue)

1. ready 상태의 Multilevel Queue

- process는 ready 상태에 들어올 때마다 **자신의 priority에 해당하는 Queue의 마지막**에 삽입
- CPU scheduler는 **priority가 가장 높은 Queue**(0번 Queue)의 맨 앞에 있는 process 6에 CPU 할당

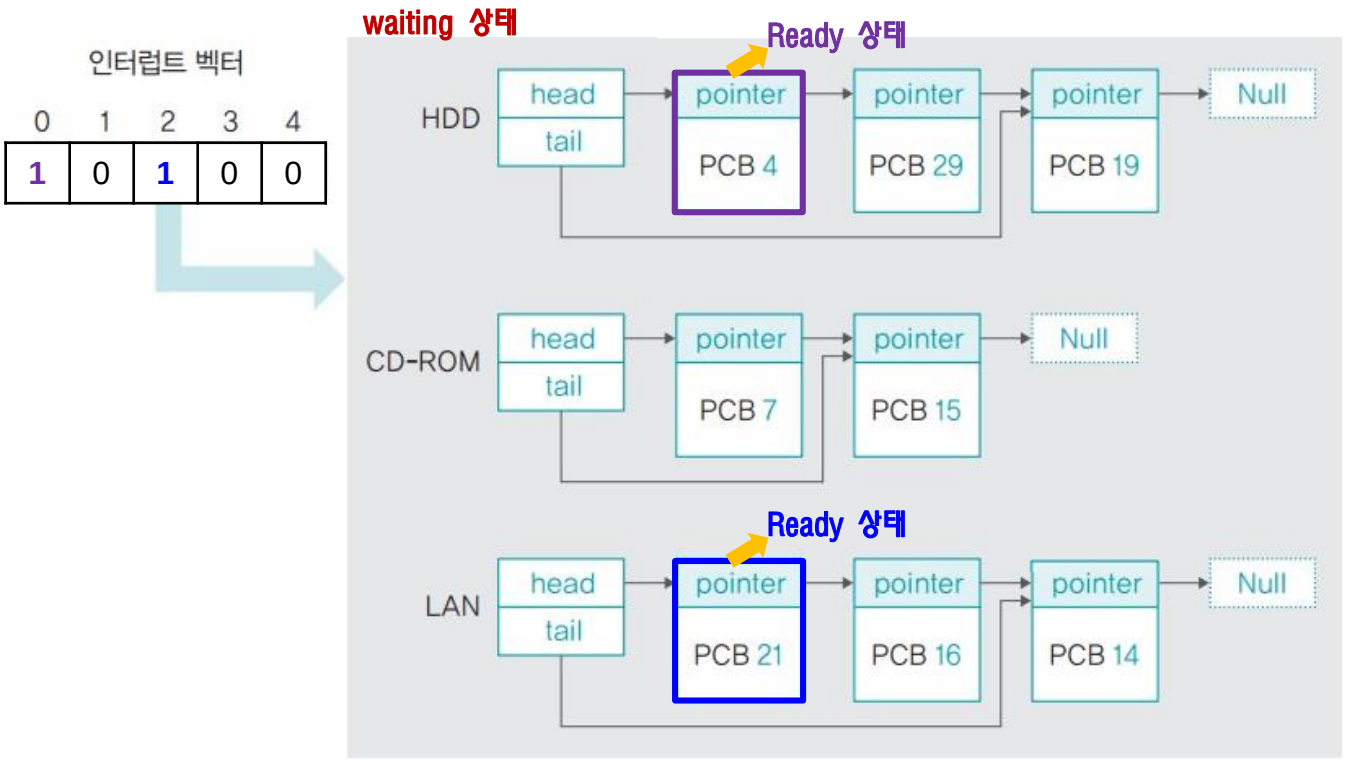


[그림] ready 상태의 multilevel queue

Multilevel Queue(다중 Queue)

2. waiting 상태의 Multilevel Queue

- system의 효율을 높이기 위해 waiting 상태에서는 **같은 입출력을 요구한 process끼리 모아 놓음**

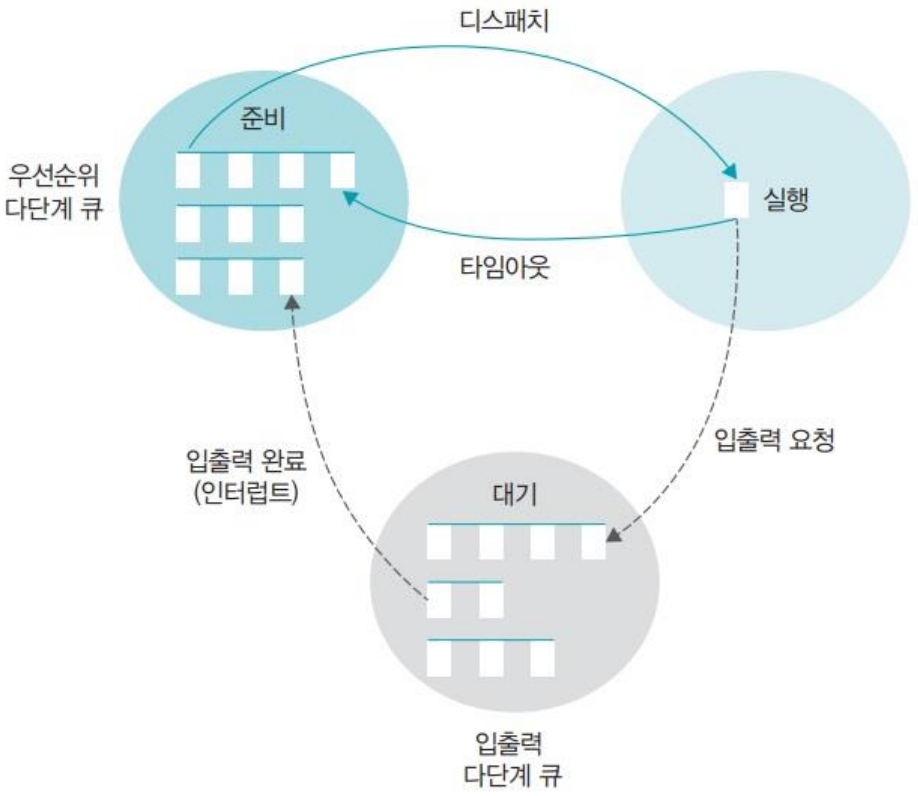


[그림] waiting 상태의 Multilevel Queue

Process 상태와 Multilevel Queue

- Multilevel Queue 비교

  - **ready Queue**
    - **한 번에 하나의 process**를 꺼내어 CPU를 할당
  - **waiting Queue**
    - **여러 개의 process를 동시**에 꺼내어 ready 상태로 옮김
    - waiting Queue에서 동시에 끝나는 interrupt를 처리하기 위해 interrupt vector를 사용



[그림] process 상태와 Multilevel Queue

Scheduling algorithm의 종류

[표] scheduling algorithm의 종류

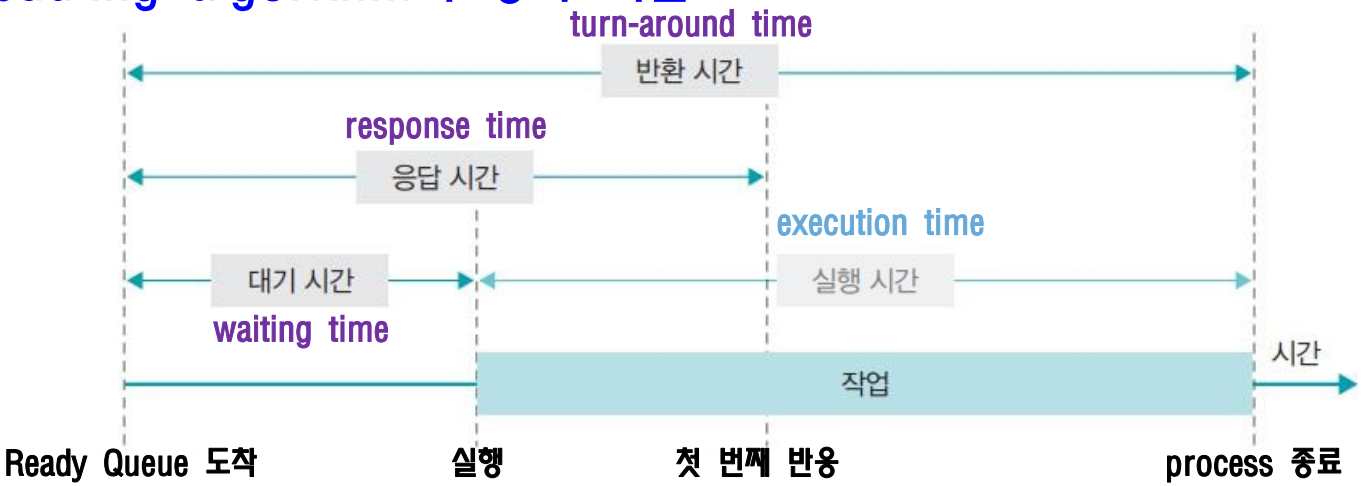
구분	종류
nonpreemptive	FCFS 스케줄링, SJF 스케줄링, HRN 스케줄링
preemptive	라운드 로빈 스케줄링, SRT 스케줄링, 다단계 큐 스케줄링, 다단계 피드백 큐 스케줄링

* Priority(우선순위) scheduling

- **preemptive(선점) scheduling**
  - OS가 **실행 상태에 있는 process의 작업을 중단**시키고 새로운 작업을 시작할 수 있는 방식
- **nonpreemptive(비선점) scheduling**
  - process가 종료되거나 자발적으로 대기 상태에 들어가기 전까지는 계속 실행 되는 방식
    (OS가 실행 상태에 있는 **process의 작업을 강제로 중단시키지 못함**)

Scheduling algorithm의 평가 기준

- scheduling algorithm의 평가 기준



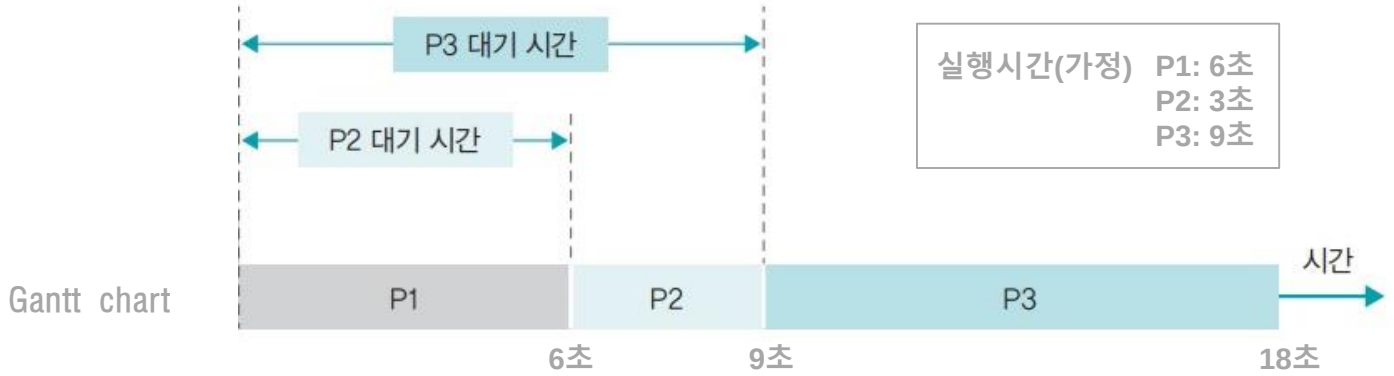
[그림] 대기시간, 응답시간, 실행시간, 반환시간의 관계

- **CPU 사용률 :** 전체 system의 동작 시간 중 CPU가 사용된 시간을 측정하는 방법
  (이상적 수치는 100%, 실제로는 90%에도 못 미침)
- **throughput(처리량) :** 단위 시간당 작업을 마친 process의 수

- **waiting time** : CPU에서 **실행되기 전에 대기**하는 시간의 합
- **response time** : **작업**을 시작한 후 **첫 번째 출력(반응)**이 나오기까지의 시간
- **turn-around time** : waiting time(대기 시간)을 포함하여 **실행이 종료될 때까지**의 시간

Scheduling algorithm의 성능 비교

- **평균 대기 시간(AWT, Average Waiting Time)**
  - scheduling algorithm의 **일반적 성능비교의 척도**
  - **모든 process의 대기 시간을 합한 뒤 process의 수로 나눈 값**



[그림] 평균 대기 시간

[EX] P1, P2, P3가 P1,P2,P3 순서로 동시에(시간차가 없이) 도착했다고 가정했을 때 Average Waiting Time은? (nonpreemptive 방식 사용)

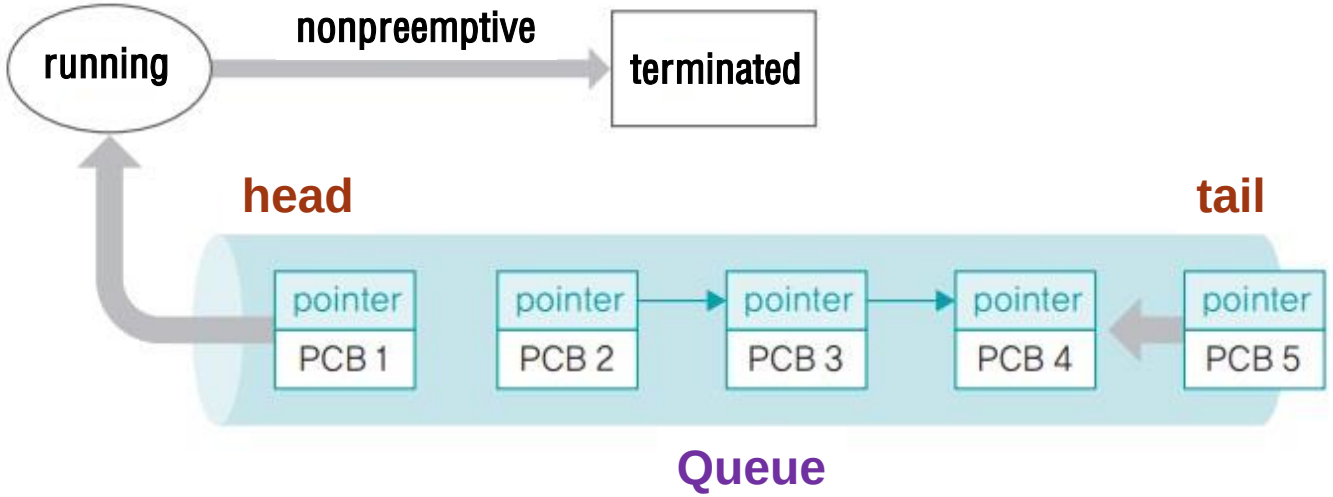
P1의 대기시작 : 0초
P2의 대기시작 : 6초
P3의 대기시작 : 9초

} **평균대기시간 = (0+6+9) / 3 = 5초**

1. FCFS(First Come First Served) Scheduling (1)

FIFO (First In First Out)

- **FCFS scheduling의 동작 방식**
  - **Ready Queue에 도착한 순서대로 CPU를 할당**하는 방식
  - nonpreemptive 방식 algorithm
  - 한 번 실행되면 그 process가 끝나야만 다음 process를 실행할 수 있음
  - 1개의 Queue – **모든 process는 priority가 동일**



[그림] FCFS scheduling의 동작

1. FCFS(First Come First Served) Scheduling (2)

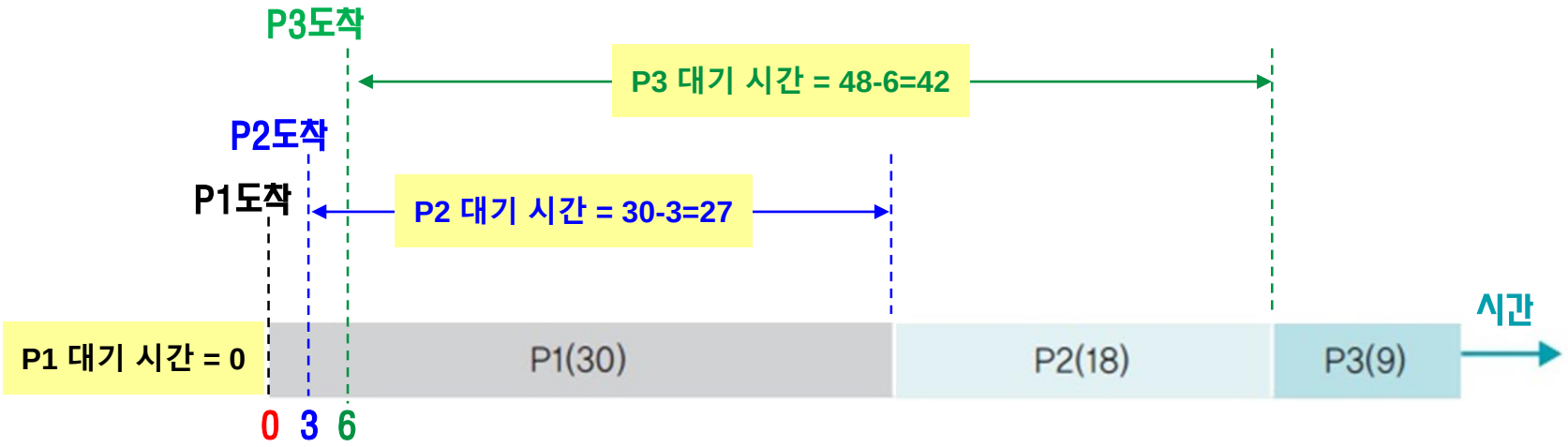
- FCFS scheduling의 성능

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

※ Average Waiting Time(AWT)은?

(0+27+42)÷3=23 ms

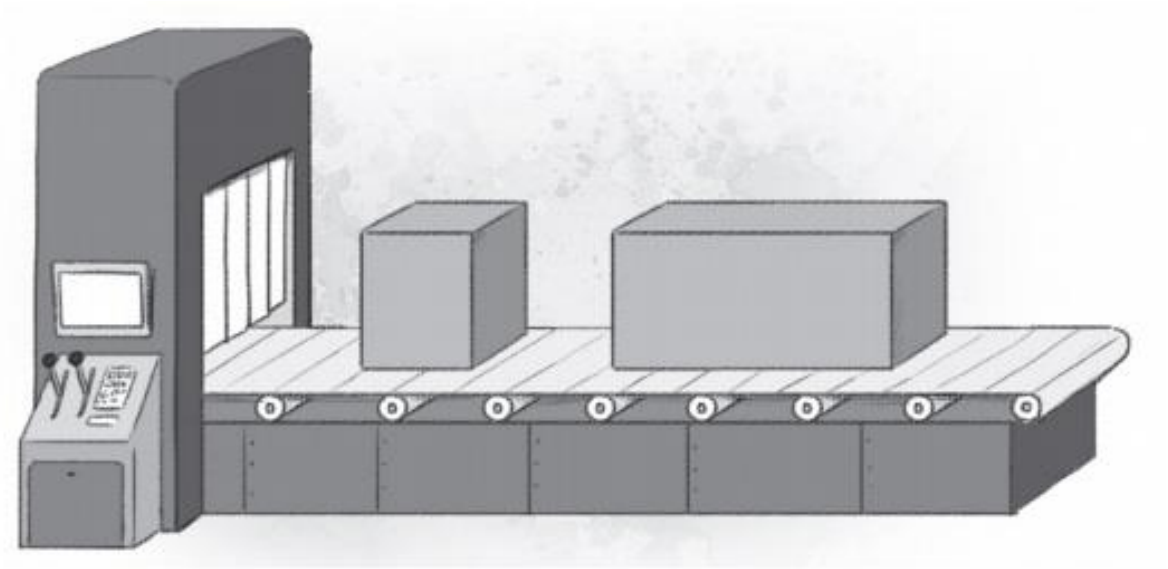
milli-second



[그림] FCFS scheduling의 평균 대기 시간

1. FCFS(First Come First Served) Scheduling (3)

- FCFS scheduling의 평가
  - 처리 시간이 긴 process가 CPU를 차지하면 **다른 process들은 계속 기다려서 system의 효율성이 떨어짐** → convoy effect (호위 효과)
  - 특히 현재 작업 중인 process가 입출력 작업을 요청하는 경우 CPU가 작업하지 않고 쉬는 시간이 많아져 작업 효율이 떨어짐



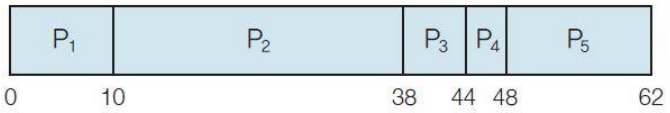
[그림] 콘보이 효과

1. FCFS(First Come First Served) Scheduling (4)

- FCFS scheduling 예제 – average waiting time과 average turn around time 구하기

프로세스	도착 시간	실행 시간
P_1	0	10
P_2	1	28
P_3	2	6
P_4	3	4
P_5	4	14

(a) 준비 큐



(b) 간트 차트

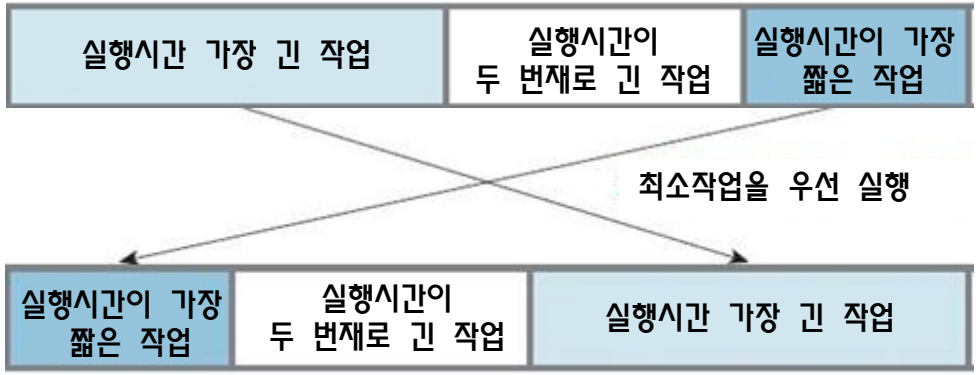
[그림] FCFS scheduling 예

*** 예제의 반환시간과 대기시간**

프로세스	반환시간	대기시간
P_1	10	0
P_2	(38 – 1) = 37	(10 – 1) = 9
P_3	(44 – 2) = 42	(38 – 2) = 36
P_4	(48 – 3) = 45	(44 – 3) = 41
P_5	(62 – 4) = 58	(48 – 4) = 44
평균 반환시간 : 38.4[= (10 + 37 + 42 + 45 + 58)/5]		평균 대기시간 : 26[= (0 + 9 + 36 + 41 + 44)/5]

평균 대기시간이 최소인 최적 algorithm

- 평균 대기시간을 줄이는 방법



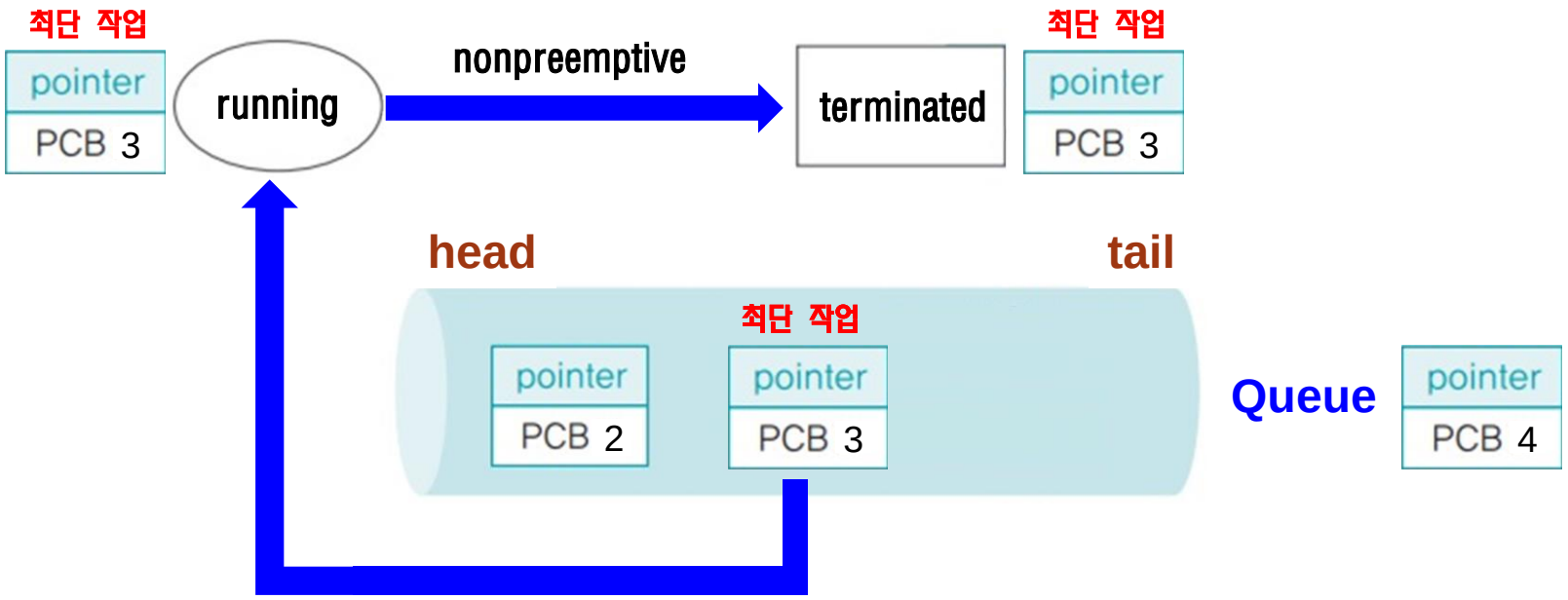
- 최적의 algorithm?

→ **실행 시간이 짧은 작업을 먼저 실행 시키는 것**

(SJF, Shortest Job First)

2. SJF(Shortest Job First) scheduling (1)

- SJF scheduling의 동작 방식
  - Ready Queue에 있는 process 중에서 **실행 시간이 가장 짧은 작업부터 CPU를 할당**
  - nonpreemptive 방식 algorithm
  - 최단 작업 우선 scheduling이라고도 함
  - **Convoy effect를 완화**하여 system의 효율성을 높임



[그림] SJF scheduling의 동작

2. SJF(Shortest Job First) scheduling (2)

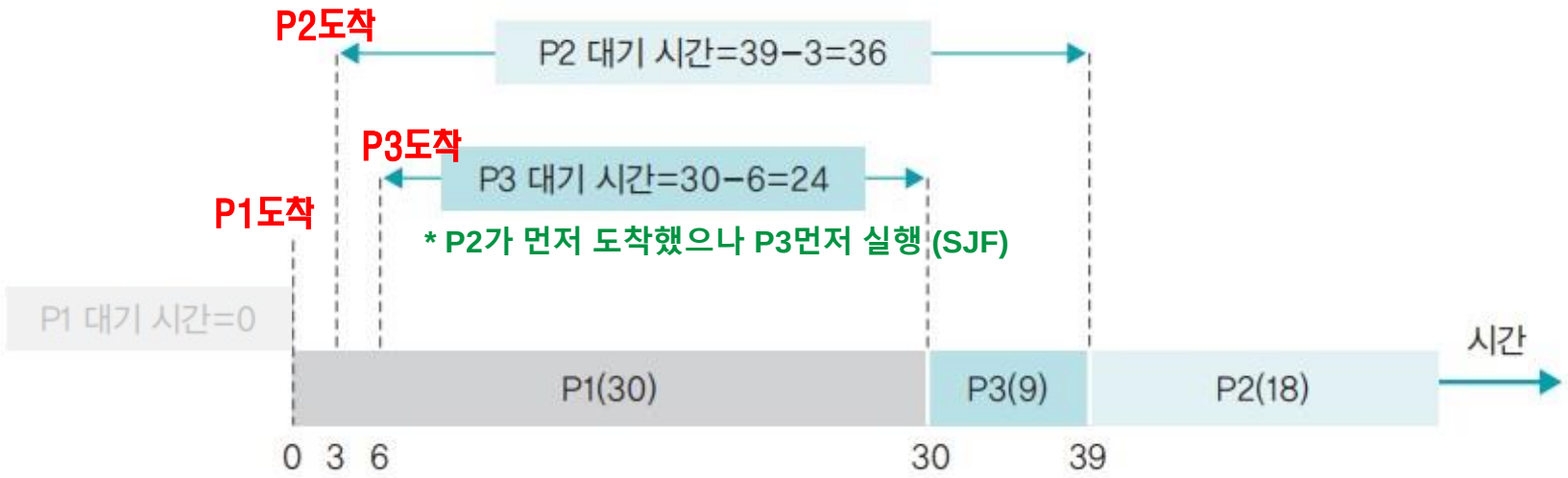
- SJF scheduling의 성능

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

※ Average Waiting Time(AWT)은?

(0+24+36)÷3=20ms

cf. FCFS의 평균대시기간=23ms



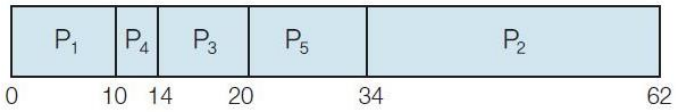
[그림] SJF scheduling의 평균 대기 시간

2. SJF(Shortest Job First) scheduling (3)

- SJF scheduling 예제 – average waiting time과 average turn around time 구하기

프로세스	도착 시간	실행 시간
P_1	0	10
P_2	1	28
P_3	2	6
P_4	3	4
P_5	4	14

(a) 준비 큐



(b) 간트 차트

[그림] SJF scheduling 예

*** 예제의 반환시간과 대기시간**

프로세스	반환시간	대기시간
P_1	10	0
P_2	(62 – 1) = 61	(34 – 1) = 33
P_3	(20 – 2) = 18	(14 – 2) = 12
P_4	(14 – 3) = 11	(10 – 3) = 7
P_5	(34 – 4) = 30	(20 – 4) = 16
평균 반환시간 : 26[= (10 + 61 + 18 + 11 + 30)/5]		평균 대기시간 : 13.6[= (0 + 33 + 12 + 7 + 16)/5]

FCFS – AWT: 26
ATT: 38.4

2. SJF(Shortest Job First) scheduling (4)

- **SJF scheduling의 평가**
  - 대기시간 측면에서 가장 최적의 scheduling 방법
  - 문제점 – 비현실적

    1. 운영체제가 process의 **종료 시간을 예측하기 어려움**

    → **process가 OS에게 자신의 작업 시간을 알려주는 방법으로 해결**

    (힘든 방법)

    ➡ **정확한 예측이 필요함**

    2. **starvation 현상이 일어날 수 있음**

    → **aging(에이징, 나이먹기)로 해결**

    - process가 양보할 수 있는 상한선을 정하는 방식

    - process가 자신의 순서를 양보할 때마다 나이를 한 살씩 먹어 최대 몇 살까지 양보하도록 규정하는 것

3. HRN(Highest Response ratio Next) scheduling (1)

- HRN scheduling의 동작 방식 -최고 응답률 우선 scheduling
  - SJF scheduling의 단점: starvation 현상
    - starvation 완화를 위해서 HRN scheduling 사용 가능
  - HRN scheduling
    - Service를 받기 위해 기다린 시간 / CPU 사용 시간 → 두 가지를 고려하여 scheduling
    - nonpreemptive 방식
  - process의 priority를 결정하는 기준

$$\text{우선순위(점수)} = \frac{\text{대기시간} + \text{CPU 사용 시간}}{\text{CPU 사용 시간}}$$

→ 점수가 높아야 priority 높음

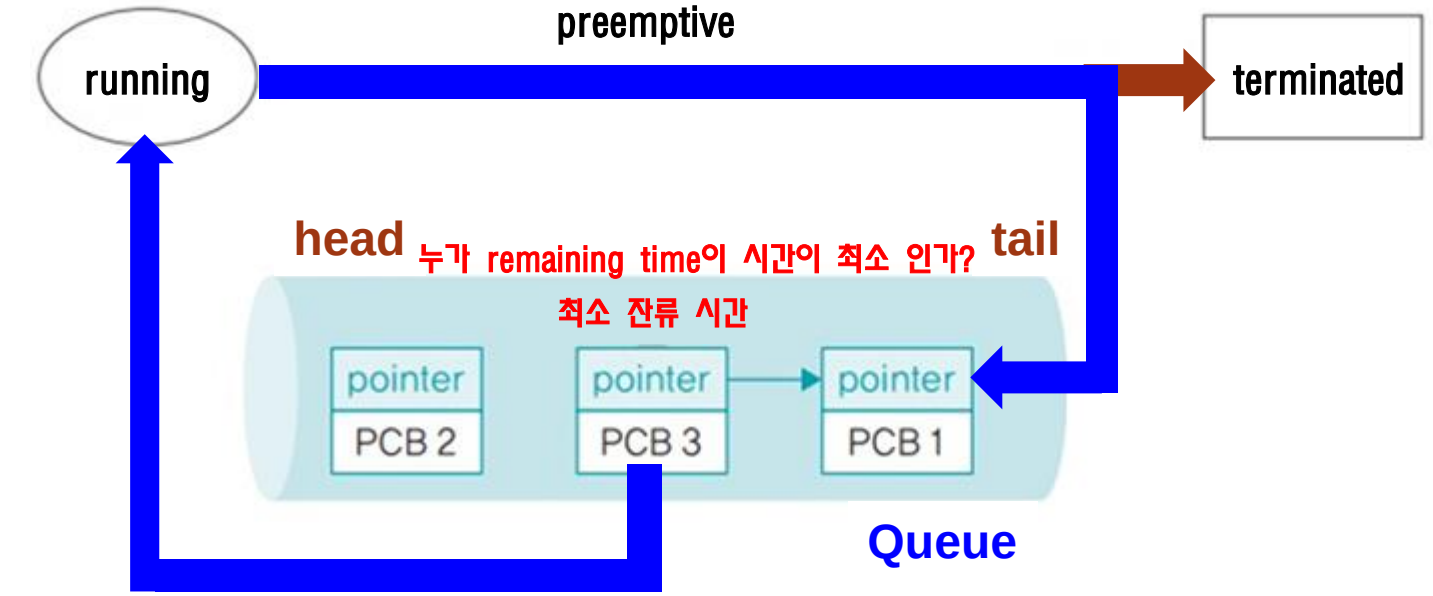
* 대기시간만 고려해서 priority를 계산하면 어떤 방식 인가? FCFS

3. HRN(Highest Response ratio Next) scheduling (3)

- HRN scheduling의 평가
  - 실행 시간이 짧은 process의 priority를 높게 설정하면서도 **대기 시간을 고려하여 starvation을 완화**
  - 대기 시간이 긴 process의 priority를 높임으로써 CPU를 할당 받을 확률을 높임
  - 문제점
    - **형평성 문제**를 완전히 개선하지는 못함(많이 사용되지 않음)

4. SRT(Shortest Remaining Time) scheduling (1)

- **SRT scheduling의 동작 방식** → SJF에서 preemptive 방식을 적용
  - CPU를 할당받을 process를 선택할 때 **남아 있는 작업 시간이 가장 적은 process를 선택**
  - preemptive 방식 algorithm임 (cf. SJF : nonpreemptive 방식 algorithm임)



[그림] SRT scheduling의 동작

4. SRT(Shortest Remaining Time) scheduling (2)

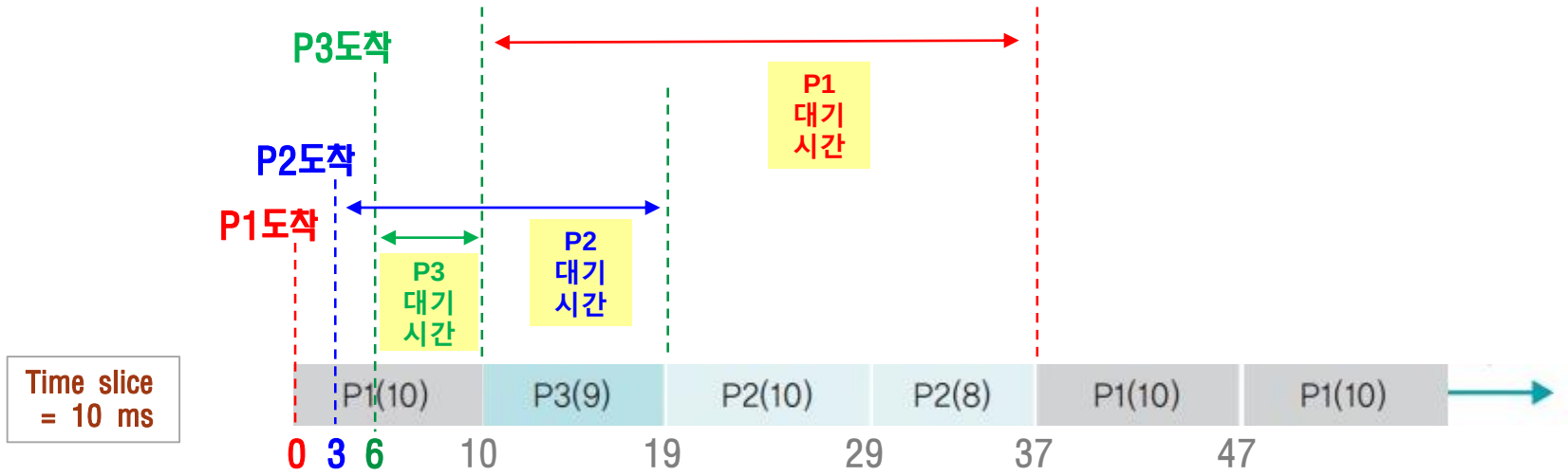
- SRT scheduling의 성능 * Time slice 지나고 다시 scheduling을 다시 한다고 가정

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

* time slice = 10 ms

※ Average Waiting Time은?

[(0+27)+ (4)+ (16)] / 3 = 15.66 ms
P1 P2 P3



[그림] SRT scheduling의 평균 대기 시간

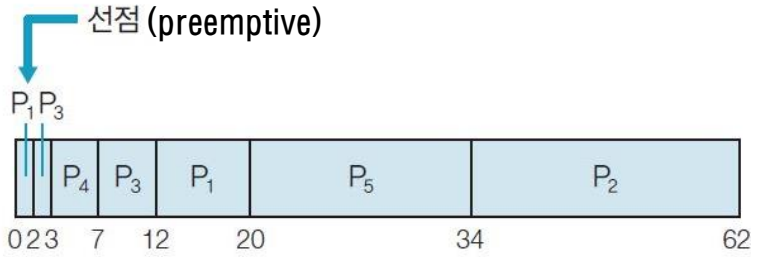
4. SRT(Shortest Remaining Time) scheduling (3)

- SRT scheduling 예제 – average waiting time과 average turn around time 구하기

* Time slice없이 process가 Queue에 도착하면 scheduling을 다시 한다고 가정

프로세스	도착 시간	실행 시간
P_1	0	10
P_2	1	28
P_3	2	6
P_4	3	4
P_5	4	14

(a) 준비 큐



(b) 간트 차트

[그림] SRT 우선 scheduling 예

*** 예제의 반환시간과 대기시간**

프로세스	반환시간	대기시간
P_1	20-0=20	(12 – 2) = 10
P_2	62-1=61	(34 – 1) = 33
P_3	12-2=10	(7 – 3) = 4
P_4	7-3=4	(3 – 3) = 0
P_5	34-4=30	(20 – 4) = 16
평균 반환시간 : 25[=(20+61+10+4+30)/5]		평균 대기시간 : 12.6[=(10 + 33 + 4 + 0 + 16)/5]

4. SRT(Shortest Remaining Time) scheduling (4)

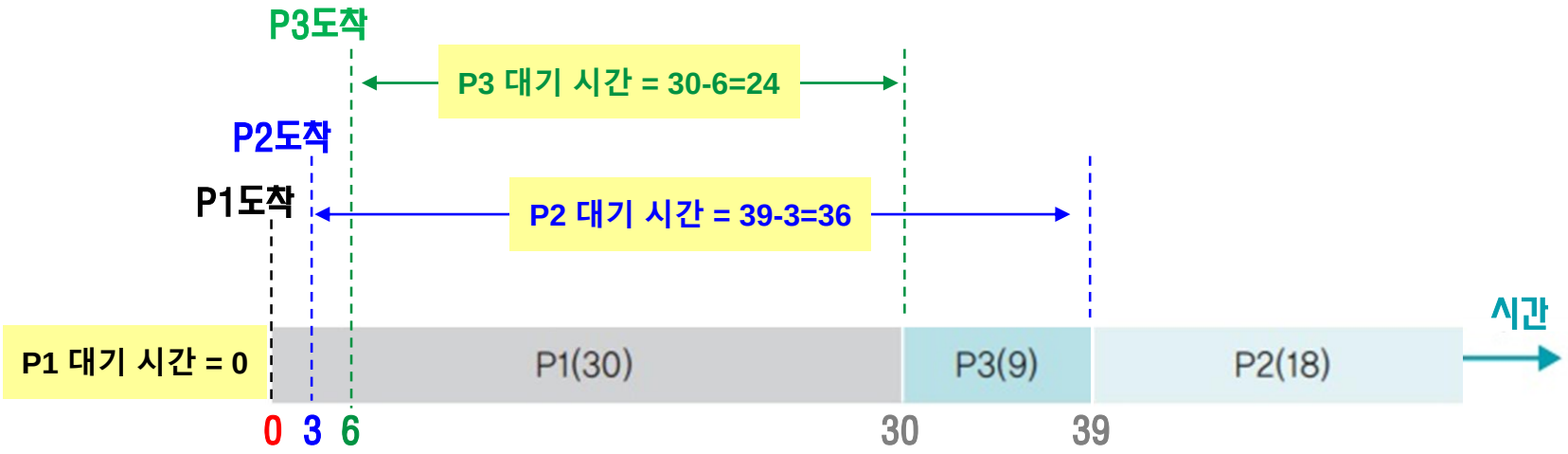
- SRT scheduling의 평가
  - 현재 실행 중인 process와 Queue에 있는 process의 **남은 시간을 주기적으로 계산**하고, 남은 시간이 더 적은 process와 **context switching**을 해야 하므로 **SJF scheduling에는 없는 할 일(작업)이 추가**됨
  - 문제점
    1. 운영체제가 process의 **종료 시간을 예측하기 어려움**
    2. **starvation 현상이 일어날 수 있음**

       → **문제점이 SJF와 같음**

Priority(우선순위) scheduling (1)

- priority scheduling의 동작 방식
  - process의 중요도에 따른 priority를 반영한 scheduling algorithm

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



[그림] FCFS scheduling에 priority를 적용한 결과

Priority(우선순위) scheduling (2)

- priority 적용
  - priority는 **nonpreemptive 과 preemptive에 모두 적용**할 수 있음
    - **(nonpreemptive) SJF scheduling** :

      작업 시간이 짧은 process에 **높은 priority**를 부여
    - **(nonpreemptive) HRN scheduling** :

      작업 시간이 짧거나 대기 시간이 긴 process에 **높은 priority**를 부여
    - **(preemptive) SRT scheduling** :

      남은 시간이 짧은 process에 **높은 priority**를 부여

Priority(우선순위) scheduling (4)

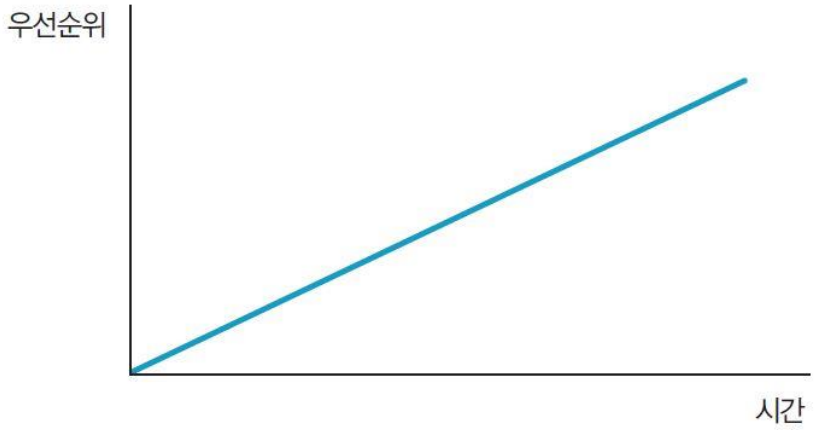
- **Static priority(고정 우선순위) algorithm**
  - 한 번 우선순위를 부여 받으면 **종료될 때까지 우선순위가 고정**
  - 단순하게 구현할 수 있지만 시시각각 변하는 system의 상황을 반영하지 못해 **효율성이 떨어짐**
- **Dynamic priority(변동 우선순위) algorithm**
  - 일정 시간마다 priority가 변하여 **일정 시간마다 priority를 새로 계산**하고 이를 반영
  - 복잡하지만 **system의 상황을 반영하여 효율적인 운영** 가능
- **Priority scheduling의 문제점**
  - **Ready Queue**에 있는 process의 순서를 무시하고 priority가 높은 process에 먼저 CPU를 할당

    → **형평성을 위배**하고 **starvation 현상**을 일으킴
  - **Ready Queue** 에 있는 process **priority를 매번 바꿈** (순서를 무시)

    → **오버헤드**가 발생하여 system의 효율성을 떨어뜨림

Priority(우선순위) scheduling (3)

- starvation 문제
  - 해결 방법 : aging(에이징)



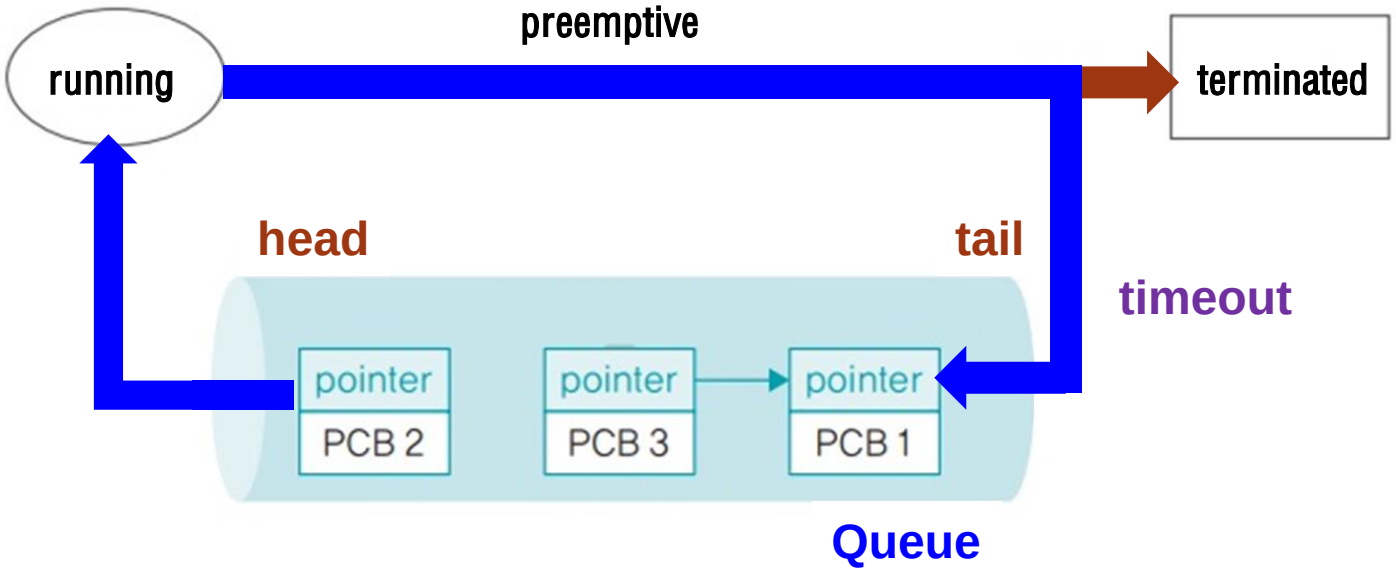
[그림] aging

(예) HRN에서 priority를 결정

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

5. Round Robin(라운드 로빈) scheduling (1)

- **Round Robin scheduling의 동작 방식**
  - process가 **time slice**(할당 받은 시간, **time quantum**) 동안 작업을 하다가 작업을 완료하지 못하면 ready queue에 가서 자기 차례를 기다리는 방식
  - preemptive algorithm 중 가장 단순하고 대표적인 방식
  - response time이 짧음 (average turn around time은 길어짐)
  - process들이 작업을 완료할 때까지 계속 순환하면서 실행



[그림] Round Robin scheduling의 동작

5. Round Robin(라운드 로빈) scheduling (2)

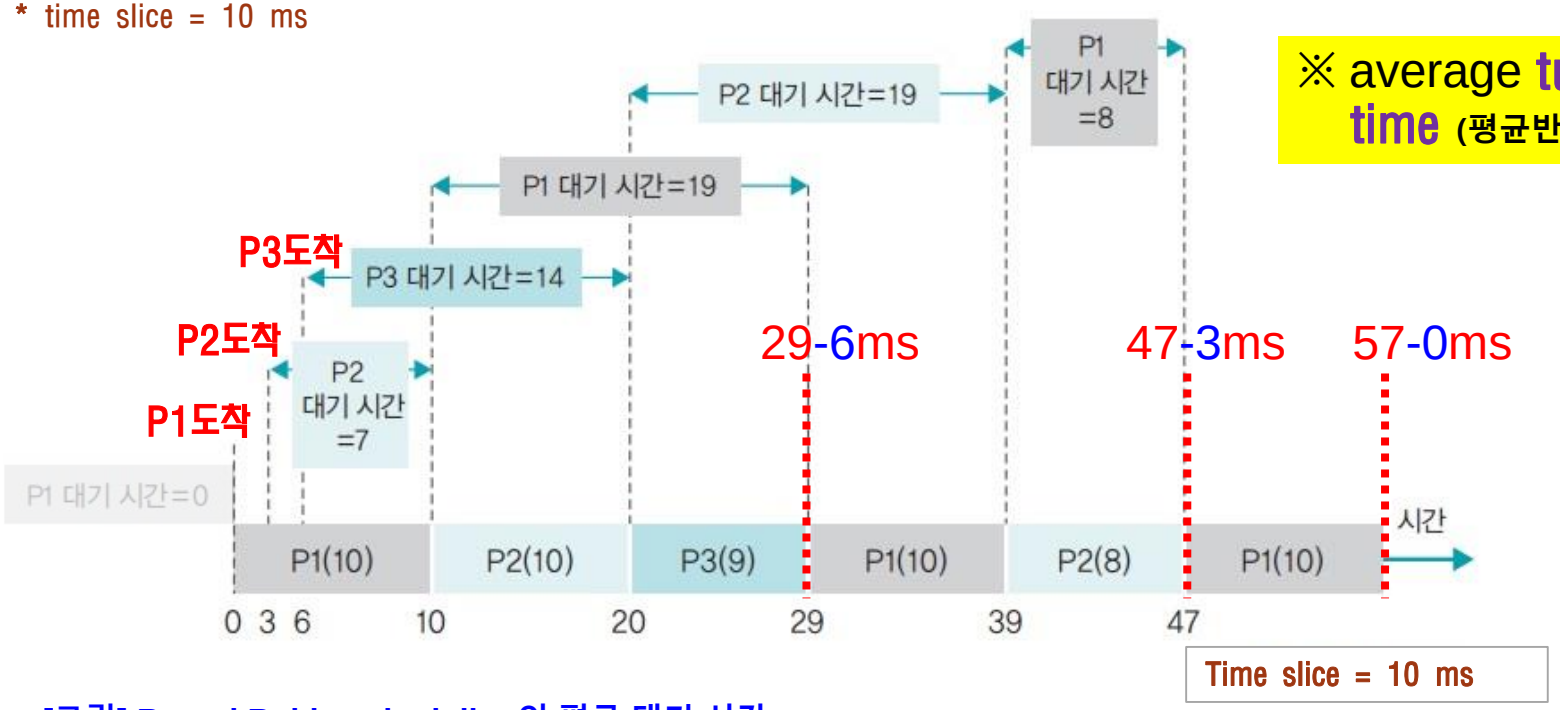
- Round Robin scheduling의 성능

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

* time slice = 10 ms

※ total Waiting Time은?
0(P1)+7(P2)+14(P3)+19(P1)+19(P2)+8(P1)=67 ms

※ Average Waiting Time은?
67÷3=22.33 ms



※ average turn-around time (평균반환시간) =?

[그림] Round Robin scheduling의 평균 대기 시간

5. Round Robin(라운드 로빈) scheduling (3)

- time slice의 크기와 context switching
  - Round Robin scheduling이 효과적으로 작동하려면 **context switching에 따른 추가 시간을 고려**하여 **time slice을 적절히 설정**해야 함

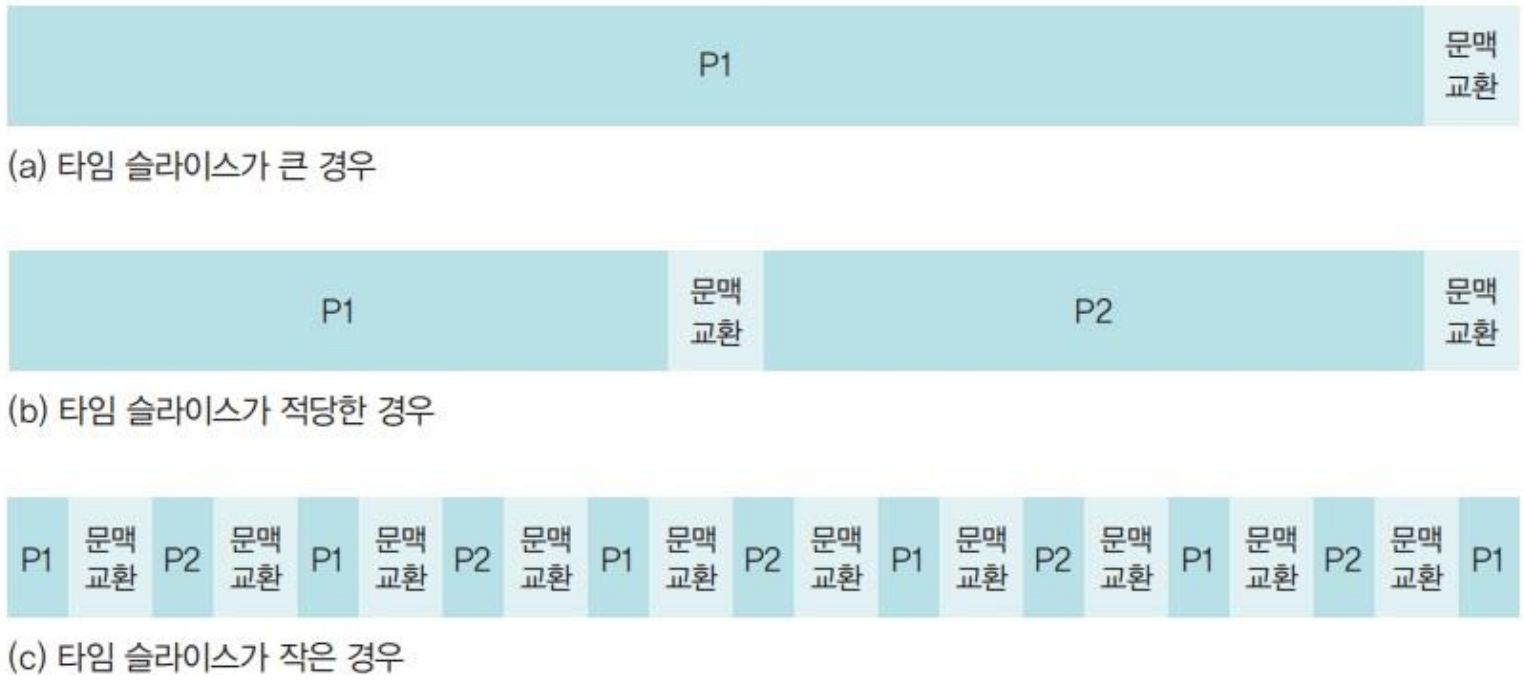
- time slice가 큰 경우? **FCFS scheduling과 같음**

- time slice가 작은 경우? **context switching에 많은 시간을 낭비**

  - context switching이 너무 자주 일어나 context switching에 걸리는 시간이 실제 작업 시간보다 상대적으로 커짐
  - context switching에 많은 시간을 낭비하여 실제 작업을 못하는 문제가 발생

5. Round Robin(라운드 로빈) scheduling (4)

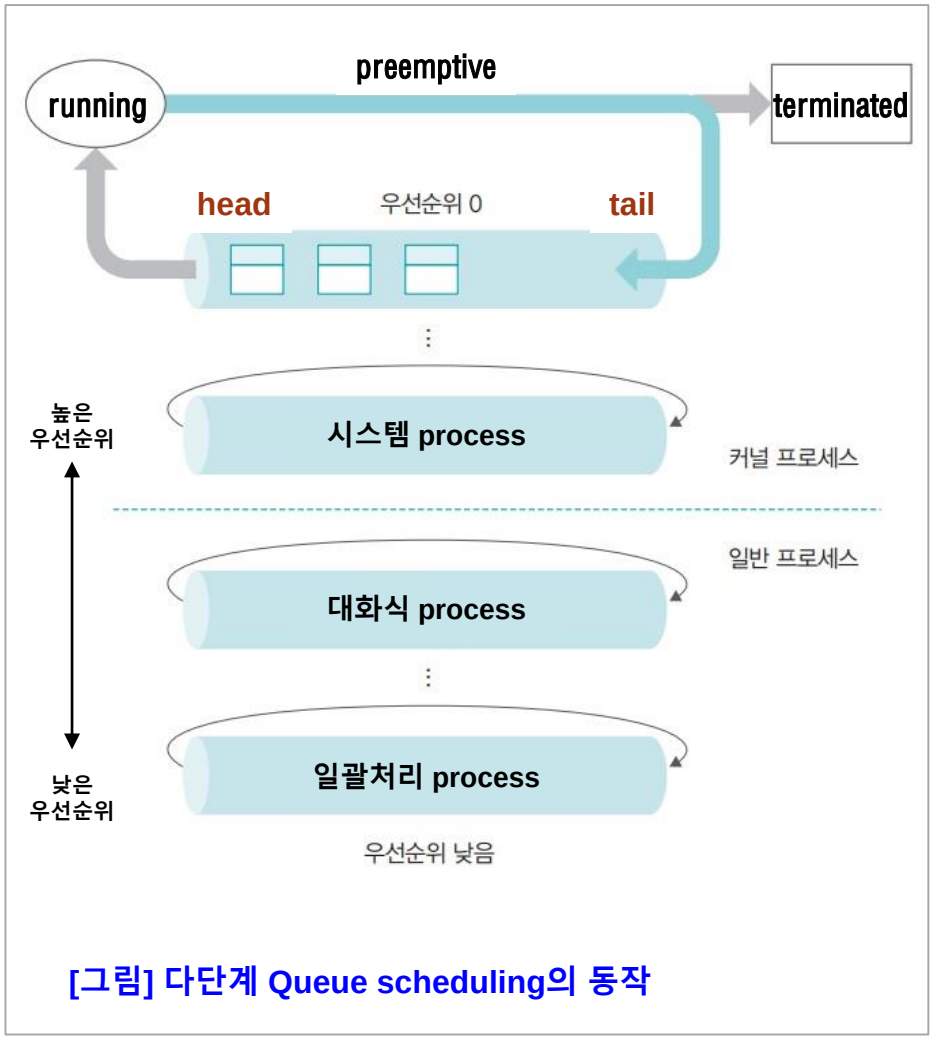
- **정리**
  - time slice는 되도록 작게 설정하되 context switching에 걸리는 시간을 고려하여 적당한 크기로 하는 것이 중요



[그림] time slice 크기와 context switching의 관계

6. Multilevel Queue(다단계 Queue) scheduling (1)

- Multilevel Queue scheduling의 동작 방식
  - priority에 따라 Ready Queue를 여러 개 사용하는 방식
  - process는 운영체제로부터 부여받은 priority에 따라 해당 priority의 Queue에 삽입
  - priority는 Static priority를 사용
  - 상단의 Queue에 있는 모든 process의 작업이 끝나야 다음 priority Queue의 작업이 시작됨

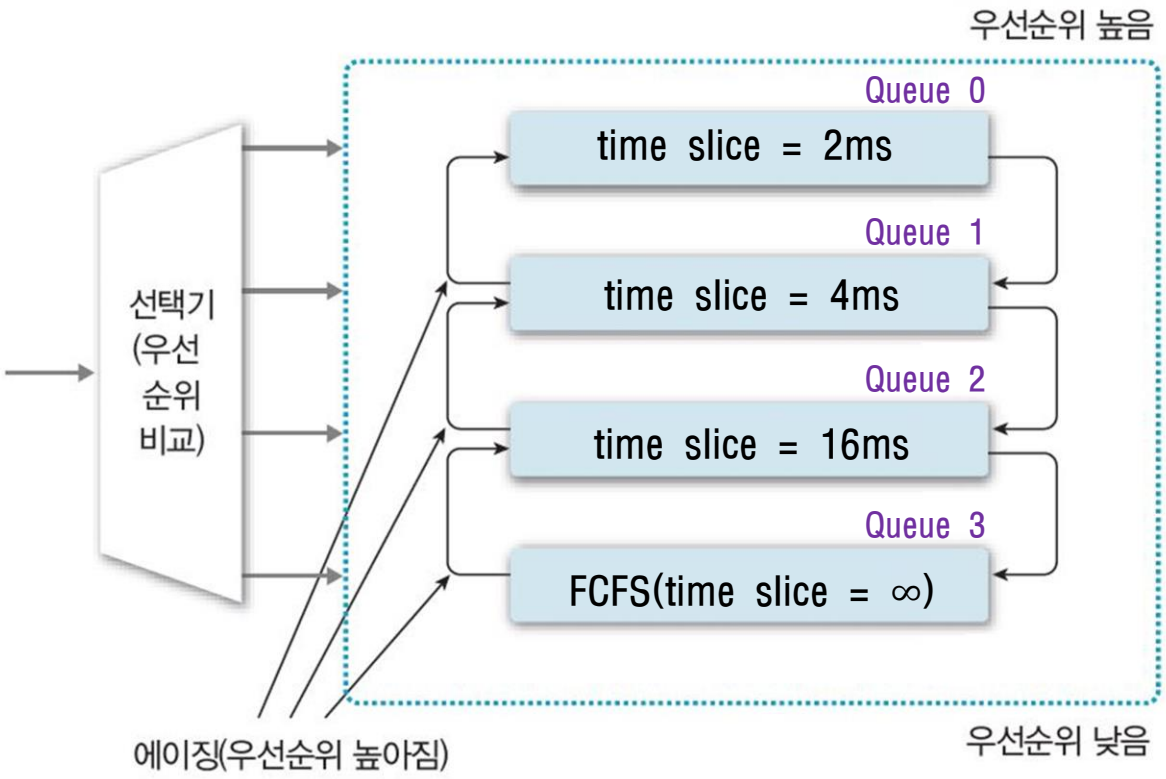


[그림] 다단계 Queue scheduling의 동작

7. Multilevel Feedback Queue scheduling (1)

(다단계 피드백 Queue)

- **원리**



[그림] 다단계 피드백 Queue scheduling

7. Multilevel Feedback Queue scheduling (2)

- **Multilevel Feedback Queue scheduling의 동작 방식**
  - process가 CPU를 한 번씩 할당 받아 **실행될 때마다 priority를 낮춤**
    Dynamic priority

  → **Multilevel Queue**에서 **priority가 낮은 process의 실행 연기 문제를 완화**

  - priority에 따라 time slice의 크기가 다름
    - 마지막 **Queue**에 있는(priority가 가장 낮은) process는 **무한대의 time slice**을 얻음
    - 마지막 **Queue**는 **FCFS scheduling 방식**으로 동작

  - priority가 낮아질수록 CPU를 얻을 확률이 적어짐.

  → 낮은 priority의 time slice을 크게 함 (CPU 할당 시 많이 작업 가능)

7. Multilevel Feedback Queue scheduling (4)

- 다단계 피드백 Queue scheduling의 장/단점

[표] 다단계 피드백 Queue scheduling의 장/단점

장점	• 매우 유연하여 스케줄러를 특정 시스템에 맞게 구성할 수 있다. • 자동으로 입출력 중심과 프로세서 중심 프로세스를 분류한다. • 적응성이 좋아 프로세스의 사전 정보가 없어도 최소작업 우선 스케줄링의 효과를 보여 준다.
단점	설계와 구현이 매우 복잡하다.

감사합니다.