

# 2장. 프로세스

What is a Process ?

“**실행 중인** 프로그램(program **in execution**)”

프로그램 : 수동적(passive)

프로세스 : 능동적(active)

## 2.1 개요

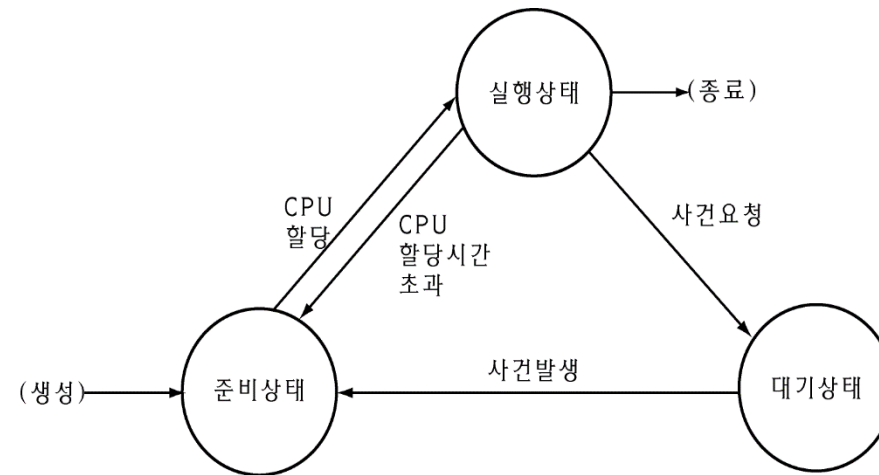
다중프로그래밍(Multiprogramming) 환경에서,

- 실행 중인 프로그램
- CPU를 할당하는 대상
- 시스템 내부에서의 작업 단위
- PCB에 존재하는 개체
- 하나의 스래드로 구성된 태스크
- ...

## 2.1.1 프로세스 상태

기본적으로,

- ① 준비(ready) : CPU 할당을 기다리고 있는 상태
- ② 실행(running) : CPU가 할당된 상태
- ③ 대기(waiting) : 요청된 사건(예:입출력) 발생이 처리되기를 기다리고 있는 상태



[그림2.1] 프로세스 상태 변환도

## 2.1.2 프로세스 영역

기본적으로,

- ① 코드(code) : 프로그램의 코드(명령어)가 저장된 영역
- ② 데이터(data) : 프로그램의 데이터가 저장된 영역
- ③ 스택(stack) : 프로그램이 실행되는 과정에서 일시적인 데이터(예: 함수호출 반환주소)를 저장하기 위해 LIFO(Last-In First-Out) 방식으로 관리되는 영역

## 2.1.3 프로세스 제어블록(PCB)

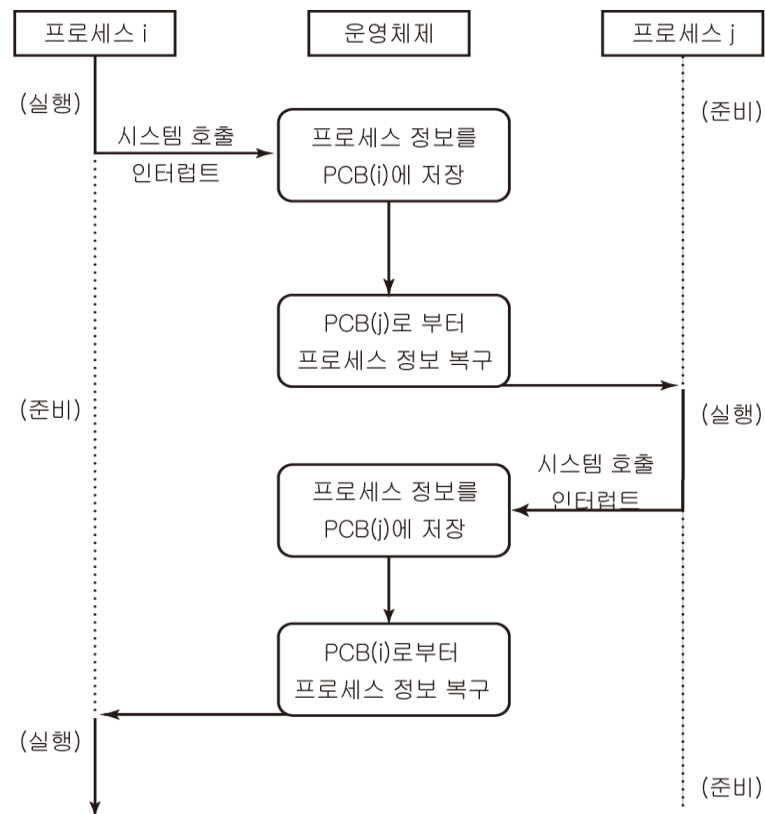
프로세스 관리에 필요한 정보들을 보관하기 위한 운영체제 내부의 자료구조

- 프로세스 식별자(PID)
- 프로세스 상태
- 우선순위
- 메모리 주소공간
- CPU 레지스터 값
- ...
- ...
- ....

※ 정보량이 매우 많음을 유의하자 !

## 2.1.4 프로세스 문맥 교환 (Context Switch)

- ① 현재 실행 중인 프로세스의 정보를 PCB에 저장하고,
- ② 다른 프로세스의 정보를 PCB로부터 복구시키는 작업



## 2.1.5 프로세스 생성 및 종료

시스템 호출(system call)에 의해 생성 및 종료

생성할 경우,

- ① PCB 엔트리를 할당하고 PID 번호를 부여한다.
- ② 프로세스를 위한 메모리 공간을 확보한다.
- ③ PCB 자료구조의 모든 정보를 초기화한다.

- 부모(parent) 프로세스 : 호출한 프로세스
- 자식(child) 프로세스 : 생성된 프로세스

✓ 최초의 프로세스는 어떻게 생성할까요?

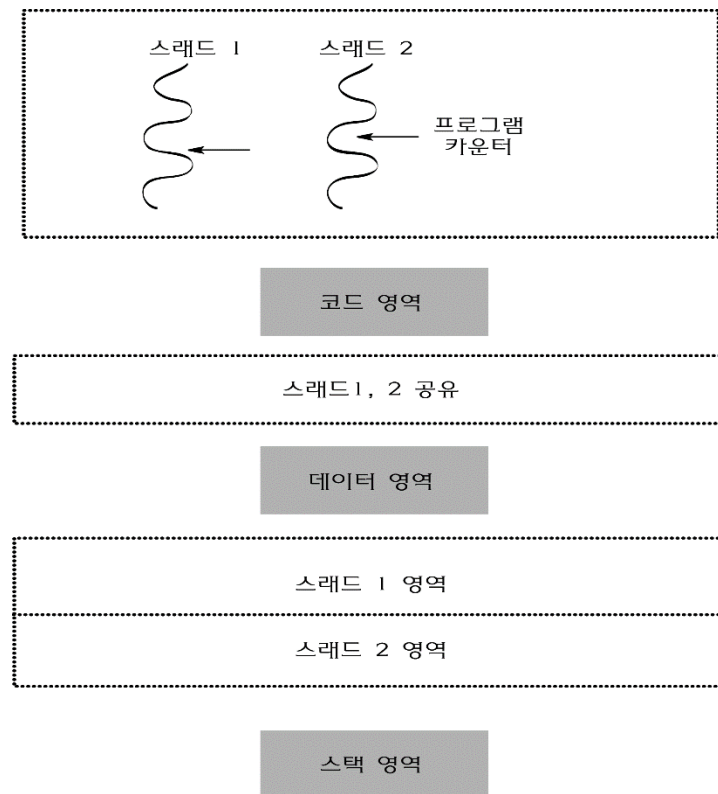
## 2.2 스래드(thread)

❖ 프로세스 = 태스크(task) + 스래드(thread)

- ① 태스크 : 정적인 부분(프로그램, 자원 등)
- ② 스래드 : 동적인 부분
  - CPU 제어의 흐름(flow of CPU control)
  - 실행 단위(unit of execution)
  - 독립적인 프로그램 카운터
  - 한 프로세스 내부에서 스케줄링이 가능한 개체
  - .....
  - .....



## 2.2.2 다중 스래딩(multi-threading)



※ 코드영역과 데이터영역은 공유하지만, 스택영역은 별도로 유지함을 유의하자!!!

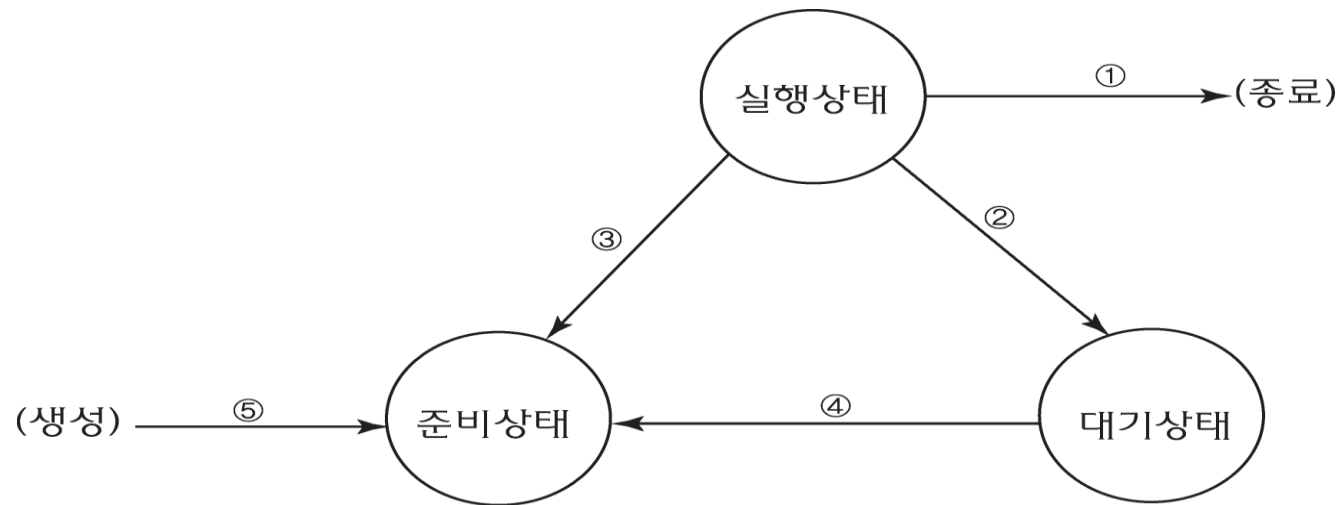
## 2.3 프로세스 스케줄링

**스케줄러(scheduler) = 스케줄링 정책(policy) + 디스패처(dispatcher)**

- 1) **스케줄링 정책**: **준비 큐**에 등록된 여러 개의 프로세스들 중에서 하나의 프로세스를 선정한다.
- 2) **디스패처** : 선정된 프로세스에게 CPU를 할당한다.

## 2.3.2 선점/비선점

- ❖ 선점(preemptive) 방식: **강제적으로** CPU를 빼앗김(①②③④⑤ 경우)
- ❖ 비선점(non\_preemptive) : **자발적으로** CPU를 반납(①② 경우)



- ① 실행 중인 프로세스가 종료될 경우
- ② 실행 중인 프로세스가 대기 상태로 전환될 경우
- ③ 실행 중인 프로세스가 준비 상태로 전환될 경우
- ④ 대기 상태인 프로세스가 준비 상태로 전환될 경우
- ⑤ 새로운 프로세스가 생성될 경우

## 2.3.3 스케줄링 정책

### ❖ 정책결정에 고려사항

- CPU 이용률 (CPU utilization) : 주어진 시간에 대한 CPU 사용시간
- 처리율 (throughput) : 단위 시간당 처리된 프로세스의 개수
- 반환 시간 (turnaround time): 프로세스가 생성된 후 종료될 때까지 소요된 시간
- 대기 시간 (waiting time): 프로세스가 준비상태에서 소요된 시간
- 응답 시간 (response time): 어떤 사건(event)가 발생한 후 첫 번째 응답이 나오는데 소용된 시간

### 2.3.3 스케줄링 정책

1. FCFS: 선입 선처리
  - ✓ 호송효과(convoy effect)
2. SJF: 최단 작업 선처리
  - SRTF(Shortest Remaining-Time First)
  - HRN(Highest-response Ratio Next)
3. Priority: 우선순위
4. RR(Round Robin): 순환처리
  - ✓ 시간 할당량(time quantum)
5. 다단계 큐(Multi-level Queue)
6. 다단계 피드백 큐(Multi-level Feedback Queue)

## (1)선입 선처리(FCFS)

- 가장 먼저 준비 큐에 등록된 프로세스를 선정
- 전형적인 비선점 방식
- 호송효과(convoy effect)

예) 준비 큐에 등록된 순서: p1,p2,p3

<u>프로세스</u>	<u>CPU 사용시간(초)</u>
P1	13
P2	4
P3	2

P1의 대기시간 = 0초

P2의 대기시간 = 13초

P3의 대기시간 = 17초

평균 대기시간 =  $(0+13+17)/3=10$ (초)

## (2)최단 작업 선처리(SJF)

- CPU 사용시간이 가장 짧은 프로세스를 선정
- 기본적으로 비선점 방식, SRTF(선점방식 SJF)
- 정확한 CPU 사용시간의 예측이 불가능

예) 준비 큐에 등록된 순서: p1,p2,p3

<u>프로세스</u>	<u>CPU 사용시간(초)</u>
P1	13
P2	4
P3	2

P3의 대기시간 = 0초

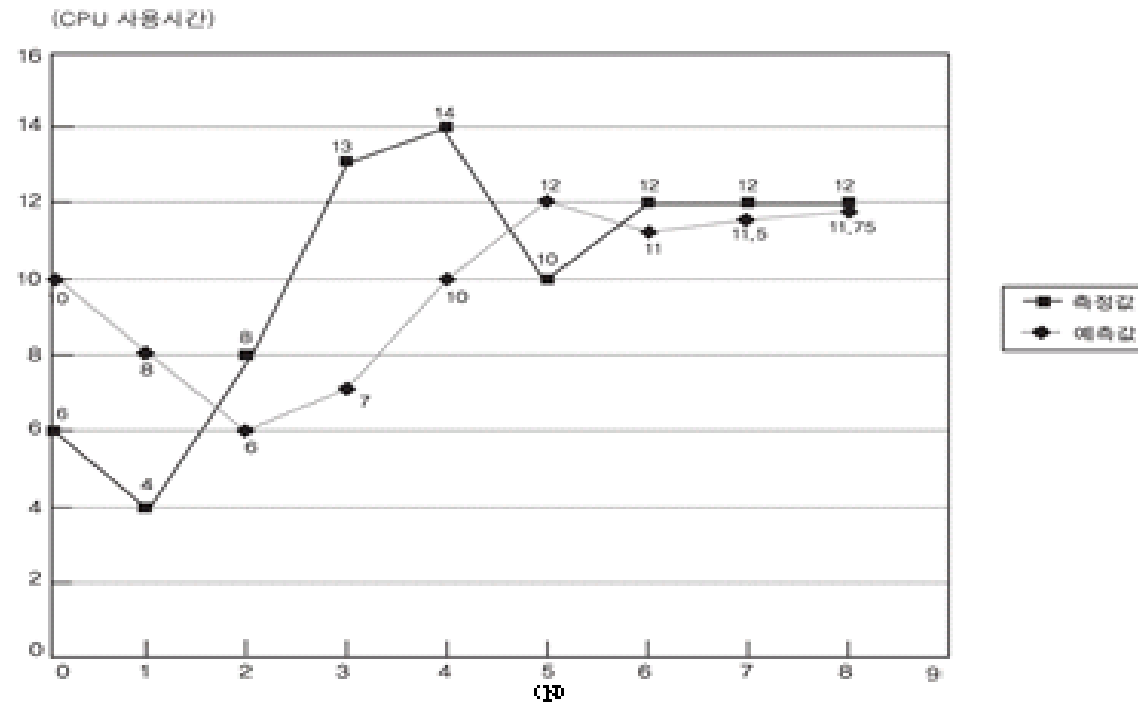
P2의 대기시간 = 2초

P1의 대기시간 = 6초

평균 대기시간 =  $(0+2+6)/3=8/3$ (초)

## 지수 평균(exponential average) 공식(CPU 사용시간 예측 방법)

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, (0 \leq \alpha \leq 1)$$



[그림 2.5] 지수 평균을 이용한 CPU 사용 시간 예측 ( $\alpha=1/2$ )



## (2')선점SJF(SRTF)

- CPU 잔여시간(remaining time)이 가장 짧은 프로세스를 선정
- 준비 큐에 새로운 프로세스가 등록될 때마다 스케줄링
- 기아 현상(starvation)

예) p2는 p1 실행 4초 후, p3는 6초 후 준비 큐에 등록

<u>프로세스</u>	<u>등록시간</u>	<u>CPU 사용시간(초)</u>
P1	0	13
P2	4	5
P3	6	2

P1의 대기시간 =  $(2+2+3) = 7$ 초

P2의 대기시간 = 2초

P1의 대기시간 = 0초

평균 대기시간 =  $(7+2+0)/3=3$ (초)

문맥교환 횟수 = 4회

### (3)우선순위(Priority)

- 프로세스가 생성될 때 우선순위를 부여한다.
- 우선순위가 가장 높은 프로세스를 선정한다.
- 기본적으로 비선점 방식이지만, 선점 방식의 우선순위 스케줄링도 가능함
- 기아현상(starvation) 발생

### (3')HRN(Highest-response Ration Next)

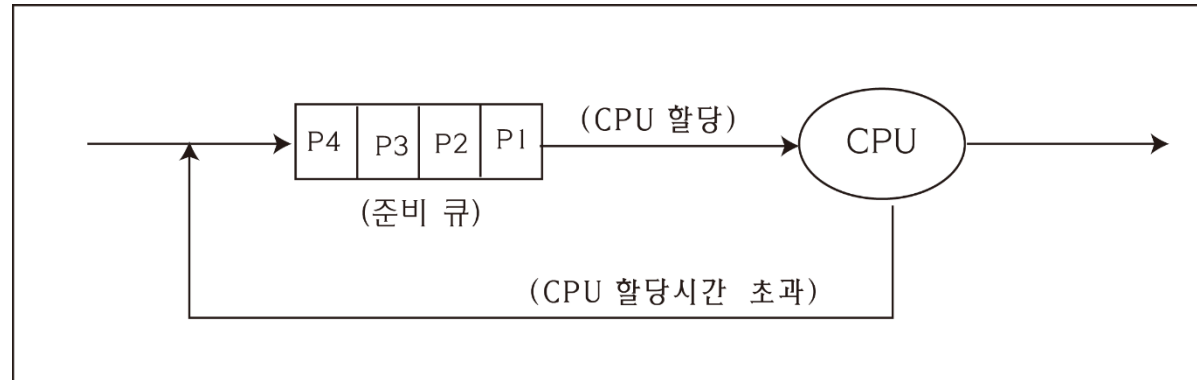
- 기아 현상(starvation)을 방지하기 위해 Brinch Hansen 제안
- CPU 대기시간을 고려한 우선순위 조정

$$\text{우선순위} = (\text{CPU 대기시간} + \text{CPU 사용시간}) / \text{CPU 사용시간}$$

- 에이징(aging) : CPU 대기시간이 긴 프로세스의 우선순위를 상향시킴으로 기아 현상(starvation)을 방지하는 효과를 기대함.

#### (4)순환처리(RR:Round-Robin)

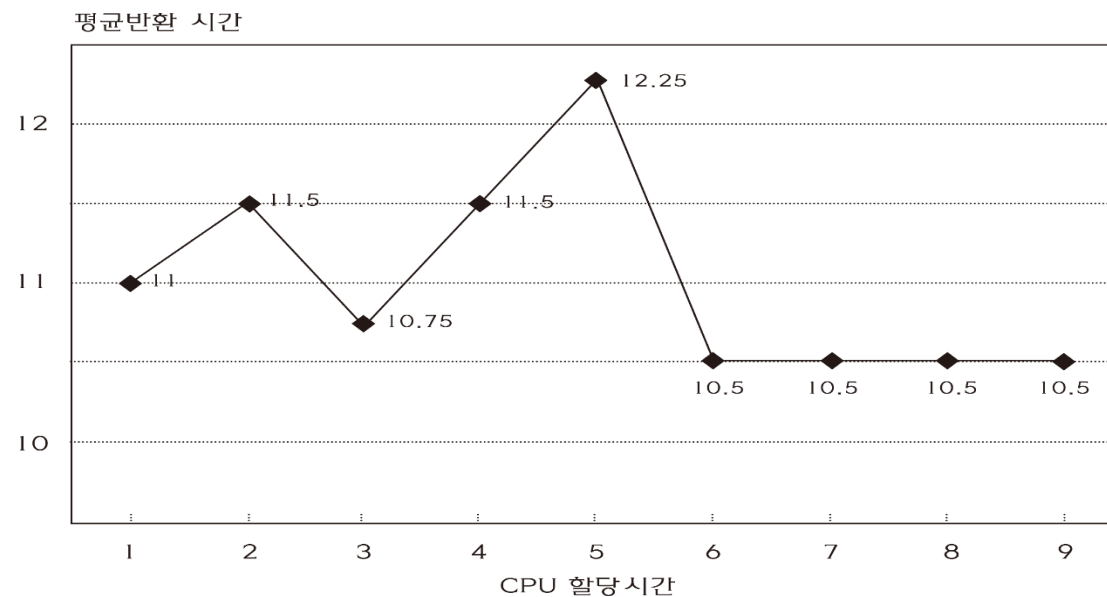
- 시분할(time-sharing) 시스템을 지원하기 위함
- 전형적인 선점 방식(=선점 방식의 FCFS)
- CPU 할당시간 결정이 중요함



[그림 2.6] 순환 처리(RR:Round-Robin) 스케줄링

※ CPU 할당시간에 따라 평균 대기시간이 달라짐을 유의하자 !

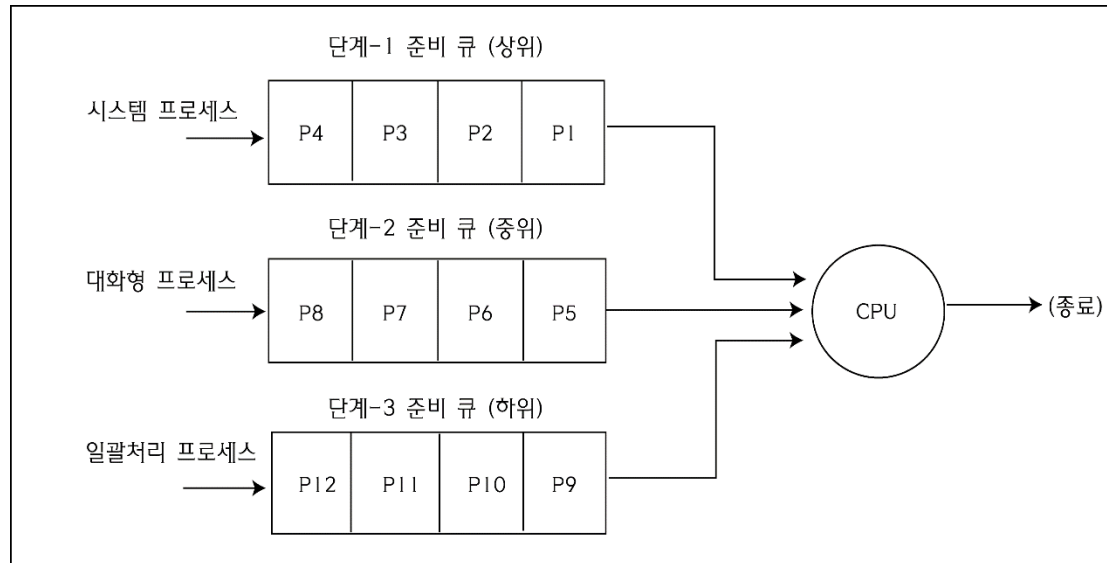
<u>프로세스</u>	<u>CPU 사용 시간(초)</u>
P1	6
P2	3
P3	1
P4	7



[그림 2.7] CPU 할당시간과 평균 대기 시간

## (5) 다단계 큐(Multi-level Queue)

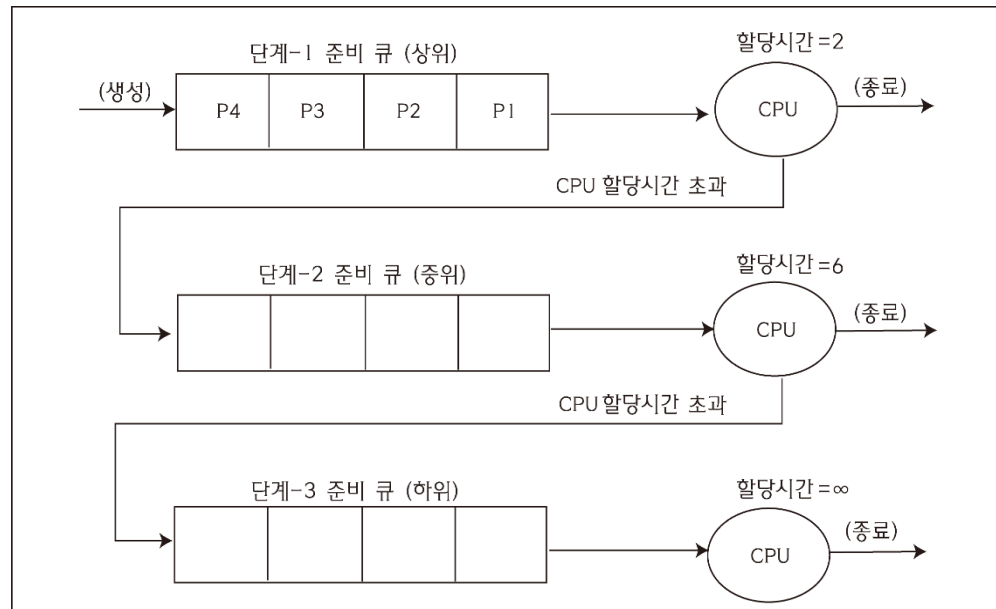
- 단계별 우선순위가 부여된 다수의 준비 큐를 사용함.
- 프로세스마다 단계별 우선순위 부여.
- 프로세스가 준비 큐들 사이를 이동할 수 없다



[그림 2.8] 다단계 큐(multi-level queue) 스케줄링

## (6)다단계 피드백 큐(Multi-level Feedback Queue)

- 단계별 우선순위가 부여된 다수의 준비 큐를 사용.
- 모든 프로세스는 최상위 우선순위 준비 큐에 등록.
- 프로세스가 준비 큐들 사이를 이동할 수 있다.



[그림 2.9] 다단계 피드백 큐(multi-level feedback queue) 스케줄링

## (6) 다단계 피드백 큐(Multi-level Feedback Queue)

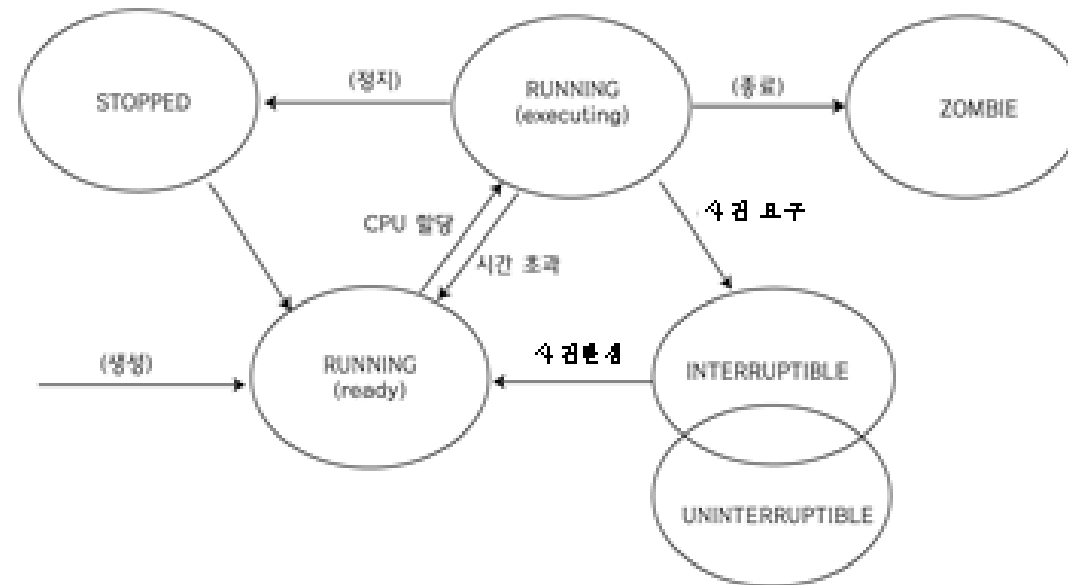
- 준비 큐마다 CPU 할당시간이 다르다. 우선순위가 낮은 준비 큐에게 더 많은 CPU 할당시간을 부여한다.
- 입출력 위주의 프로세스(I/O bounded process)가 CPU 위주의 프로세스(CPU bounded process) 보다 유리하다.
- 에이징(aging) 기법으로 우선순위가 낮은 준비 큐에 있는 프로세스를 우선순위가 높은 준비 큐로 이동시킴으로써 기아현상을 방지할 수 있다.



## 2.4 리눅스 프로세스

### 2.4.1 프로세스 상태(※5가지의 상태를 정의하고 있음을 유의하자 !!!)

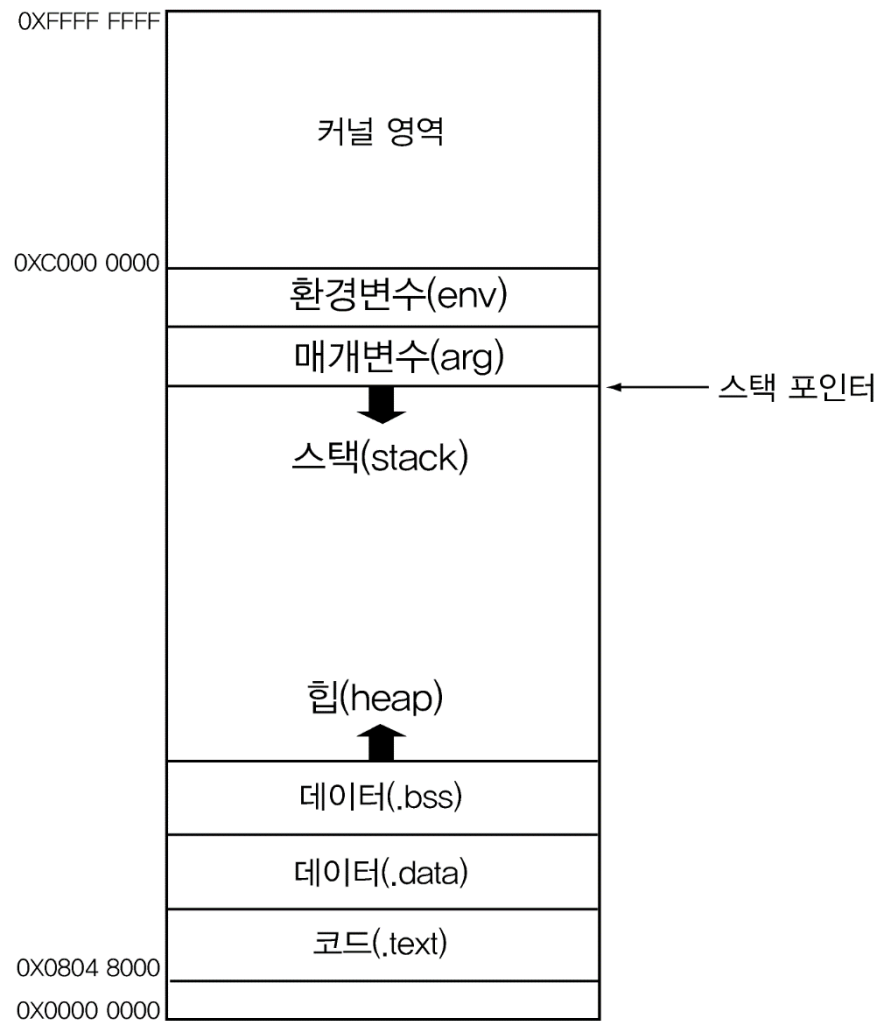
```
#define TASK_RUNNING      0
#define TASK_INTERRUPTIBLE 1
#define TASK_UNINTERRUPTIBLE 2
#define TASK_ZOMBIE      4
#define TASK_STOPPED      8
```



[그림 2.10] 리눅스의 프로세스 상태 변환

## 2.4.2 프로세스 영역

(※가상 메모리 주소임을 유의하자 !!!)



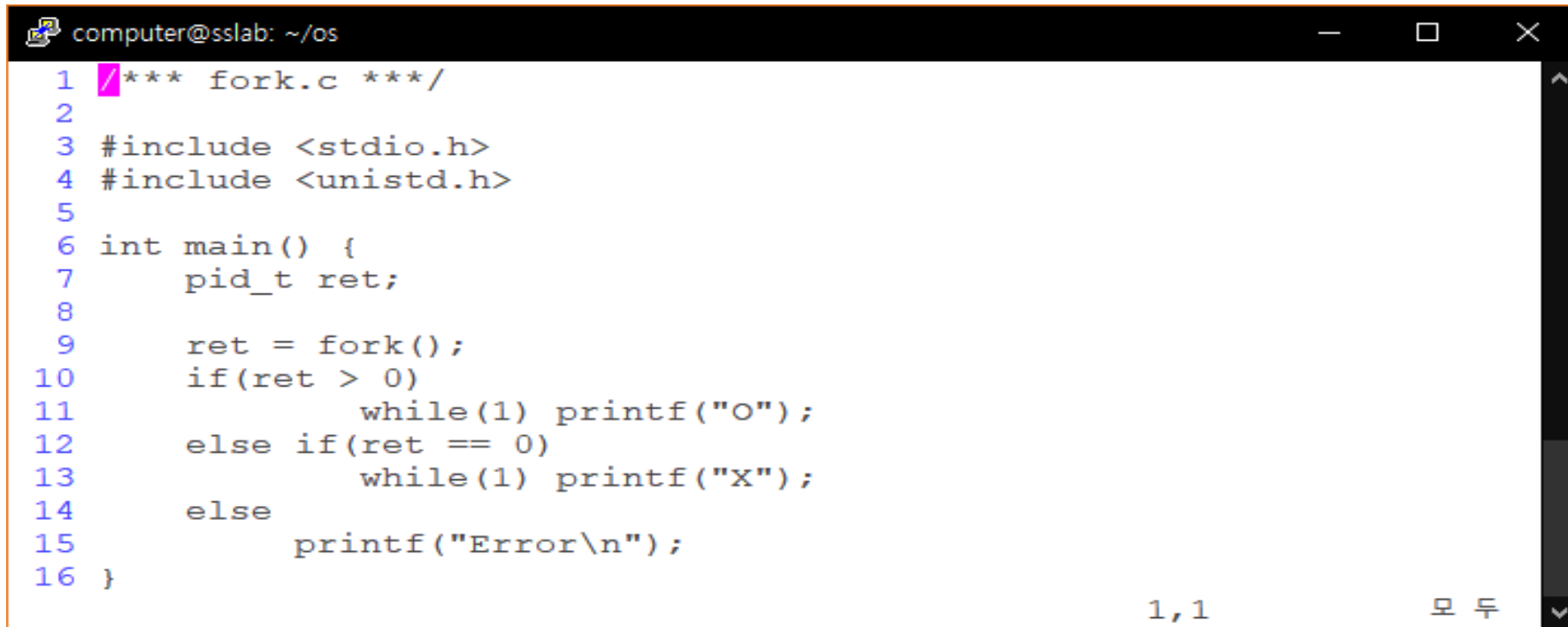
### 2.4.3 프로세스 제어 블록

(※ 아주 많은 종류의 정보를 가지고 있음을 유의하자 !!!)

```
struct task_struct {
    volatile long state; /*프로세스 상태 */
    unsigned long flags; /* 현재 시스템 상태 */
    int sigpending; /* 도착된 신호 */
    mm_segment_t addr_limit; /* 사용 가능한 주소공간의 한계 */
    struct exec_domain *exec_domain;
    volatile long need_resched; /* 스케줄링 여부 */
    unsigned long ptrace; /* 추적 상태 */
    int lock_depth; /* 잠금 상태 */
    long counter; /* CPU 사용 시간 */
    long nice; /* 사용자가 부여한 우선순위 값 */
    unsigned long policy; /* 스케줄링 정책 */
    :
    :
    :
    :
}
```

## 2.4.4 프로세스 생성(fork())

### <예제 프로그램 2-1>



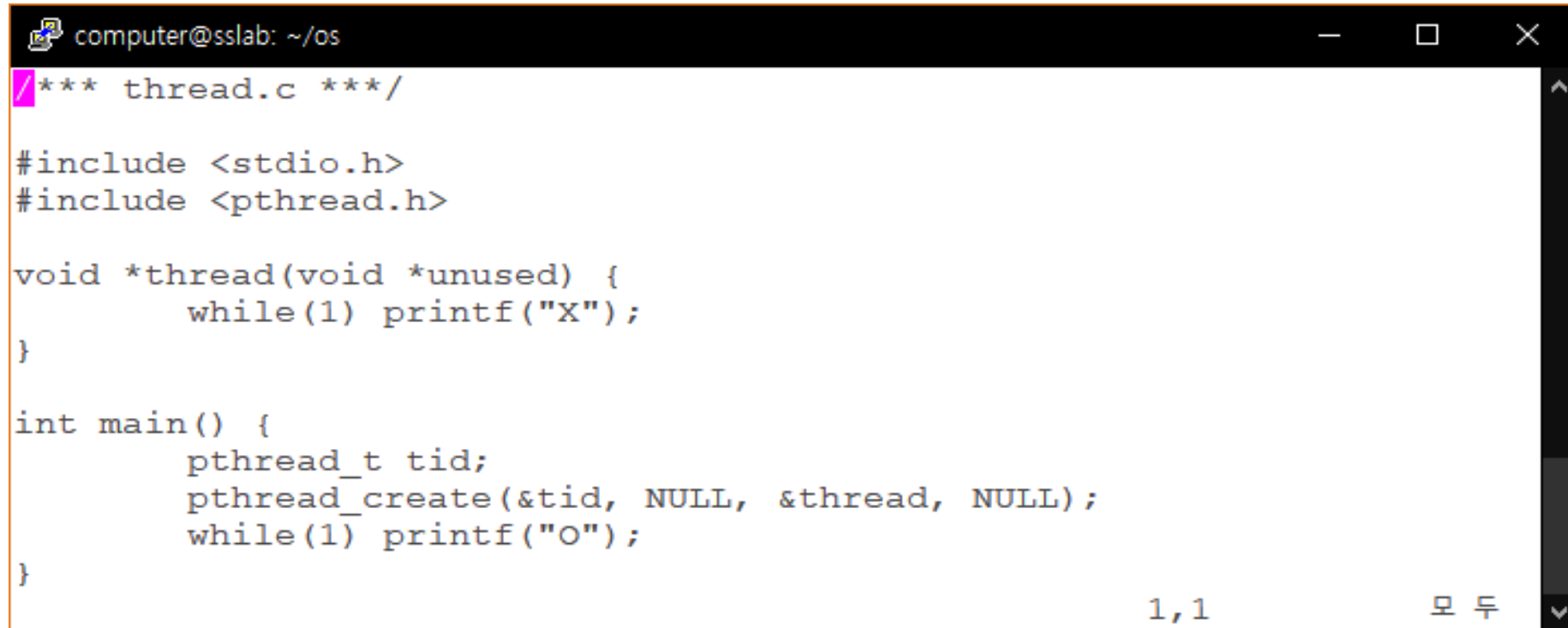
```
computer@sslab: ~/os
1  /** fork.c **/
2
3  #include <stdio.h>
4  #include <unistd.h>
5
6  int main() {
7      pid_t ret;
8
9      ret = fork();
10     if(ret > 0)
11         while(1) printf("O");
12     else if(ret == 0)
13         while(1) printf("X");
14     else
15         printf("Error\n");
16 }
```

```
% gcc fork.c -o fork
```

```
% ./fork
```

## 2.4.4 스래드 생성(pthread\_create())

<예제 프로그램 2-2>



```
computer@sslab: ~/os
/** thread.c **/

#include <stdio.h>
#include <pthread.h>

void *thread(void *unused) {
    while(1) printf("X");
}

int main() {
    pthread_t tid;
    pthread_create(&tid, NULL, &thread, NULL);
    while(1) printf("O");
}
```

```
% gcc thread.c -o thread -lpthread
```

```
% ./thread
```

## 2.4.5 스케줄링 정책

(※ 3가지의 정책을 혼용하여 사용하고 있음을 유의하자 !!!)

```
#define SCHED_OTHER 0
#define SCHED_FIFO  1
#define SCHED_RR    2
```

- ① SCHED\_OTHER : 일반 프로세스를 위한 다단계 피드백 방식
- ② SCHED\_FIFO : 실시간 프로세스를 위한 비선점 방식
- ③ SCHED\_RR : 실시간 프로세스를 위한 선점 방식