운영체제

2019.6.26

컴퓨터공학과 이병문
가천대학교 Gachon University

강의일정

- 6/24 과목, 강의소개
- 25 운영체제 개요
- 26 Process, Thread
- 27 Concurrent Process
- 28 Concurrent Process
- 7/1 중간고사1

내용

프로세스(Process) 개념

프로세스(Process) 관리

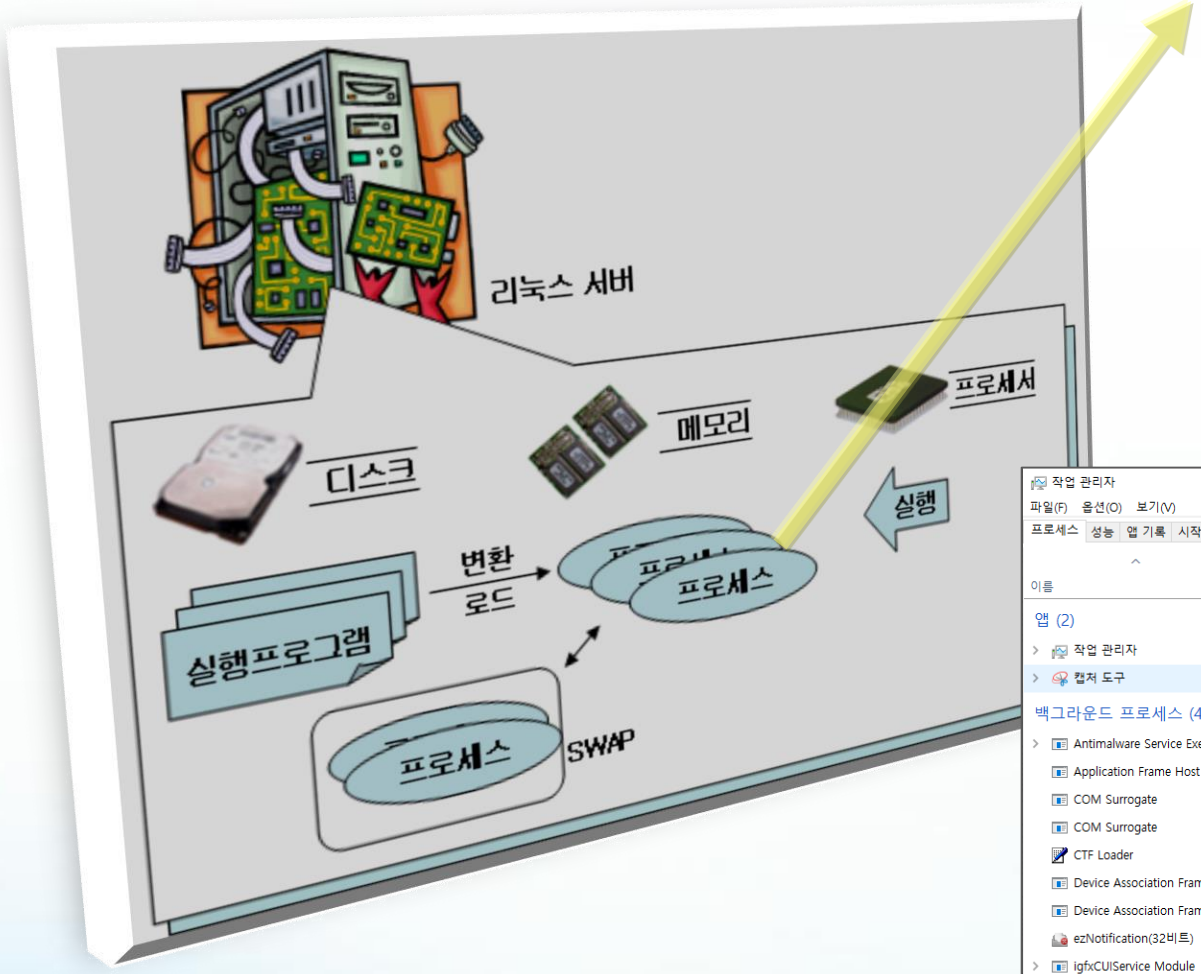
쓰레드(Thread)

병행프로세스

- Precedence Graph
- Concurrent Program method (fork/join, parbegin/parend)

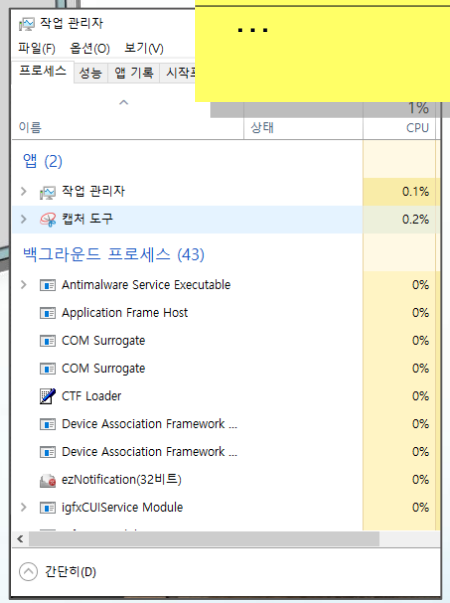
프로세스 개념

■ Process ; 실행중인 프로그램
(메모리에 로딩된 형태)

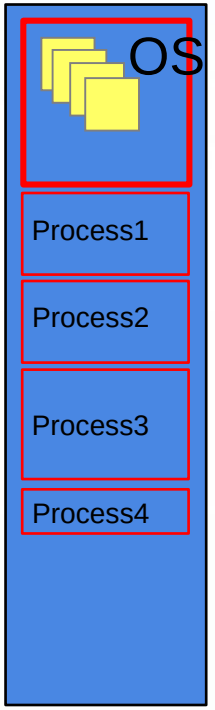


PCB

- process ID
- process state
- program counter
- registers
  …
- memory limit
- list of open files
- …

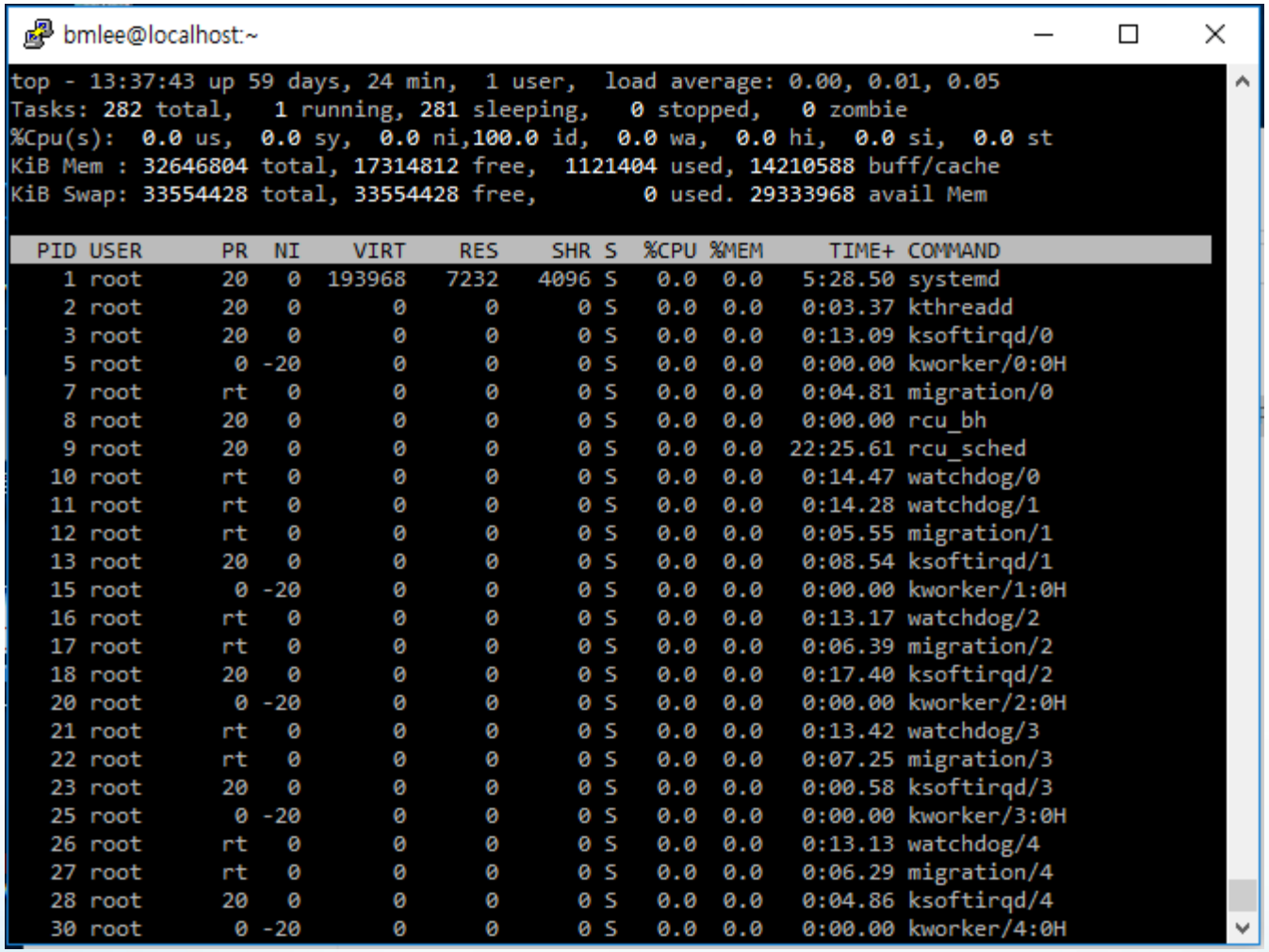


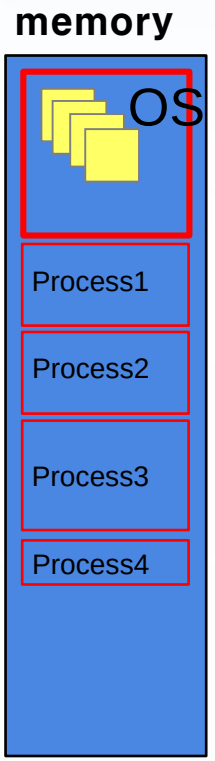
memory



프로세스 개념

- Process ; 실행중인 프로그램
  (메모리에 로딩된 형태)

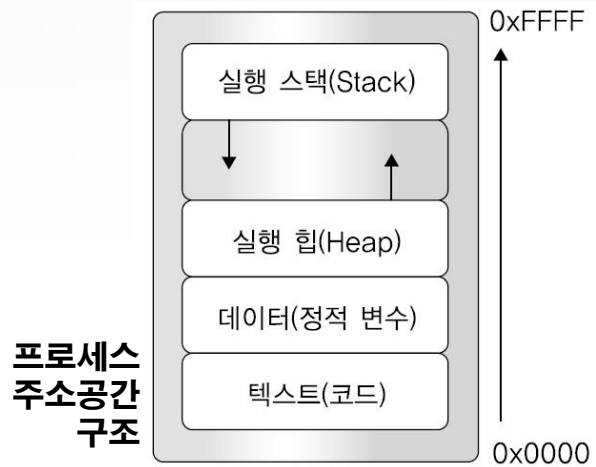




프로세스 개념

■ Process Image

☑ 프로세스



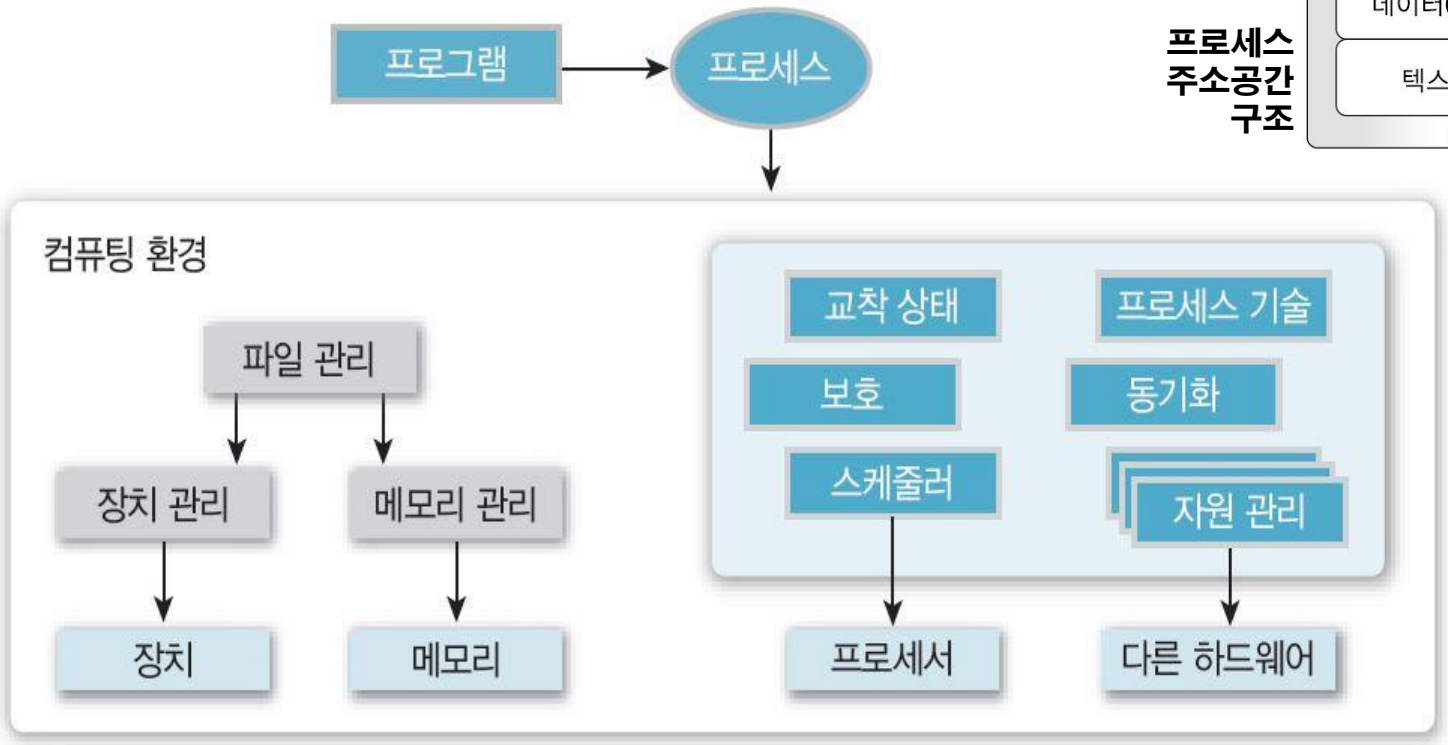


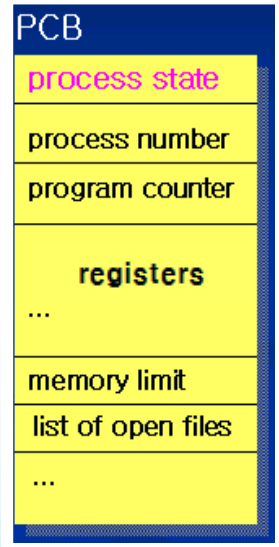
그림 3-3 시스템 관점에서 바라본 프로세스

프로세스 개념

■ Process Classification

- ☑ 시스템(커널) 프로세스
- ☑ 사용자 프로세스
- ☑ 병행 프로세스
  - 독립 프로세스
  - 협동 프로세스

■ Process State



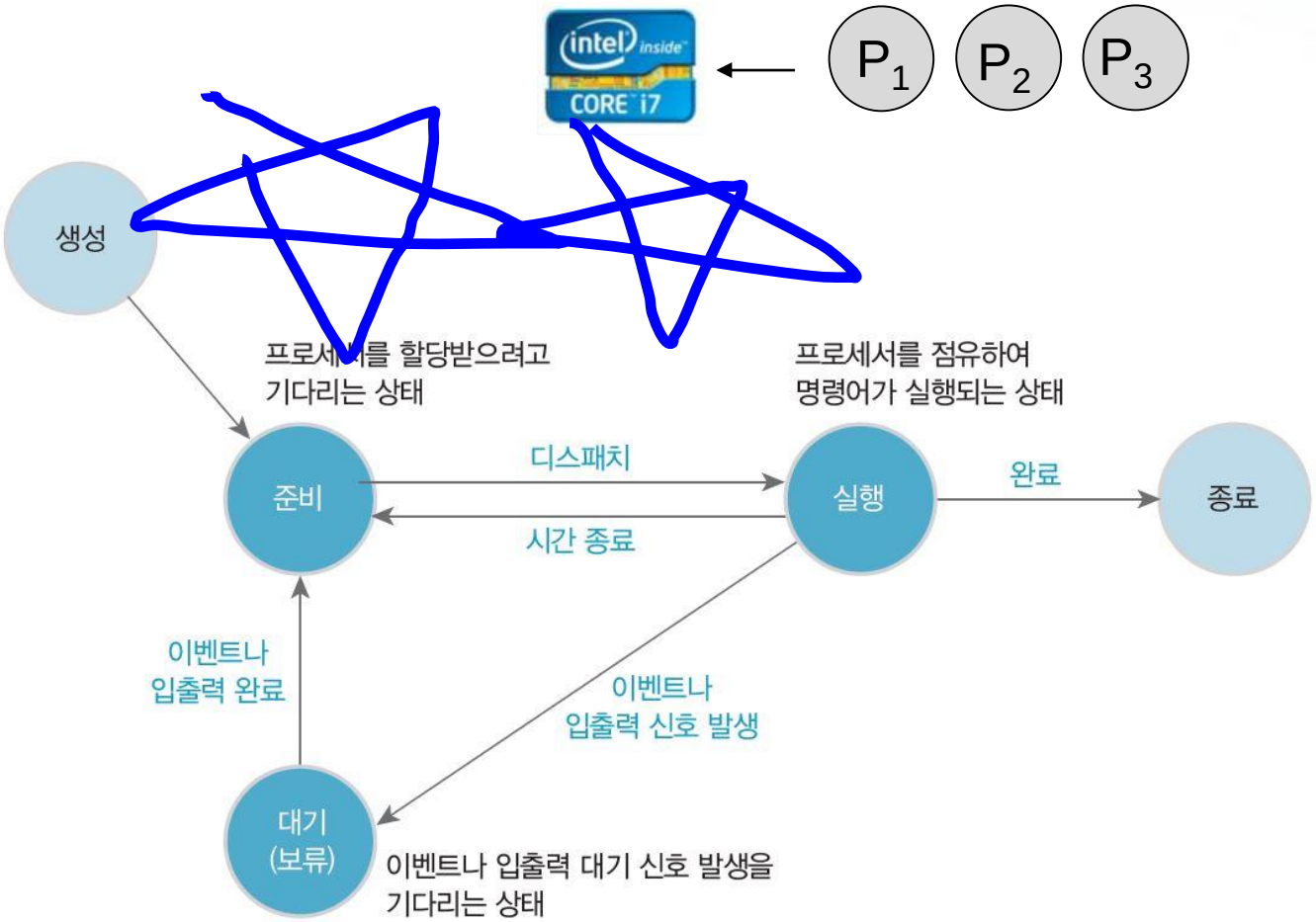


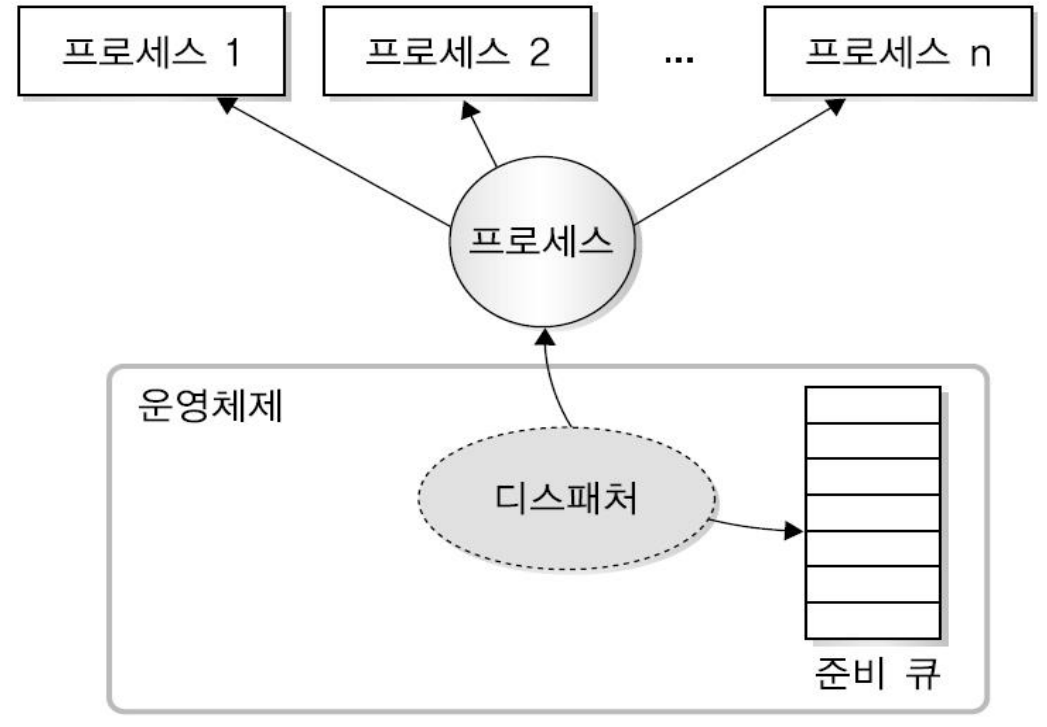
그림 3-5 프로세스의 상태 변화

프로세스 개념

■ Process Scheduling

☑ Dispatcher ; Process Scheduler

준비 큐(Ready Queue) **맨 앞에 있던 프로세스가 프로세서를 선택(배당되어 실행)하는 것.**



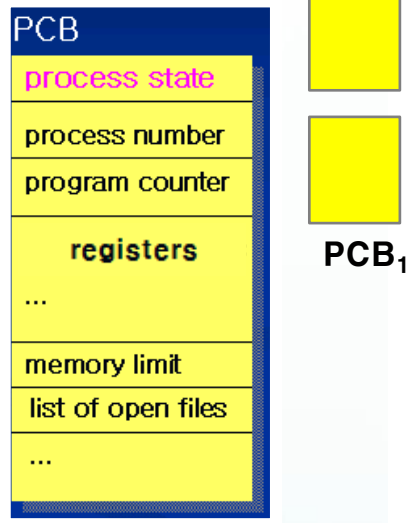
프로세스와 디스패처

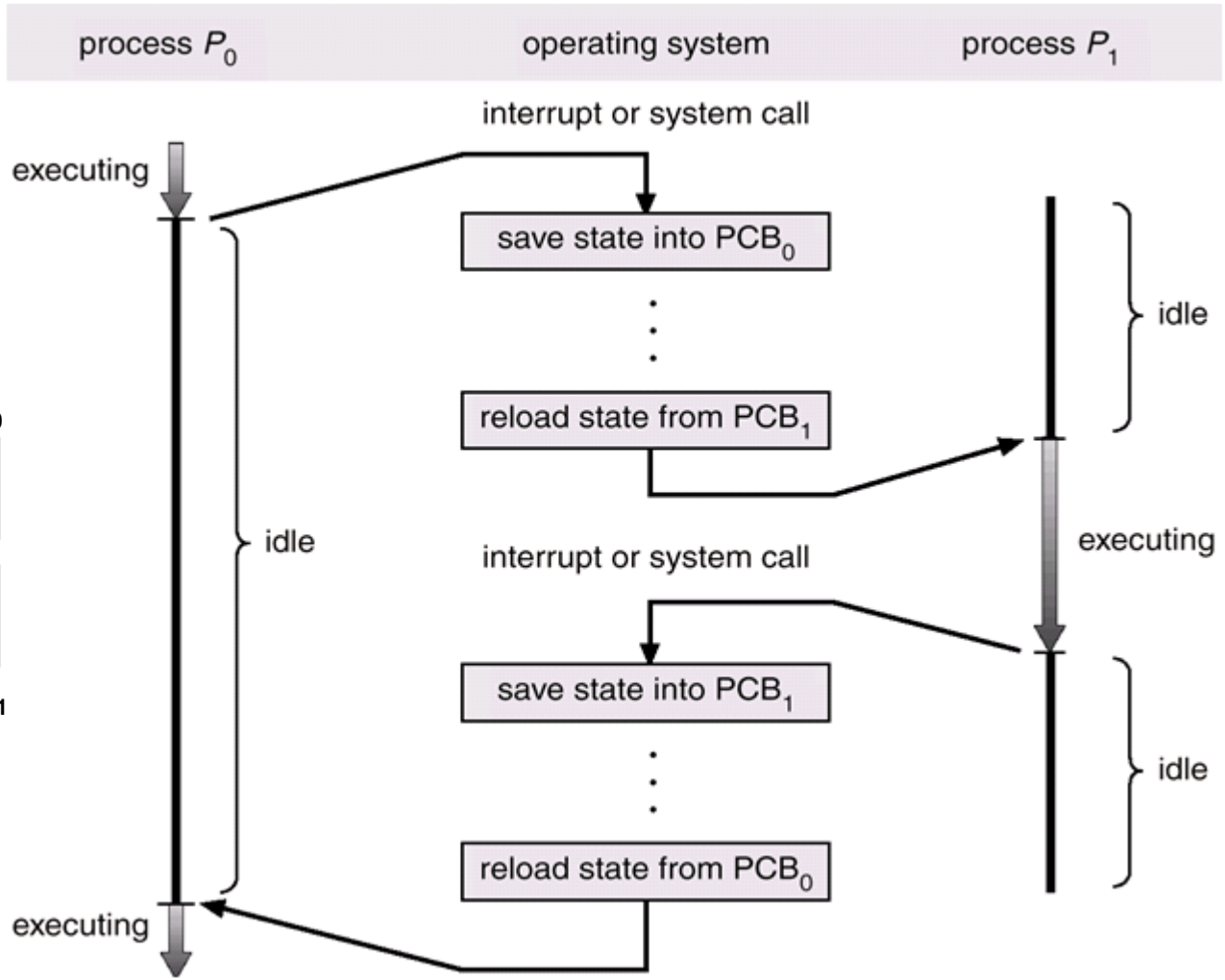
프로세스 개념

■ **CPU switch** (from Process P_0 to Process P_1) = **Context Switch**

☑ 프로세스의 교환

1) Interrupt
2) Trap
3) System Call



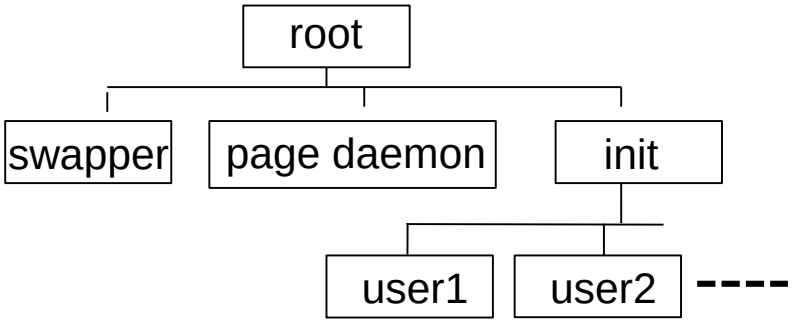


Context Switch ; **횟수▲ -> 성능▼**

프로세스 관리

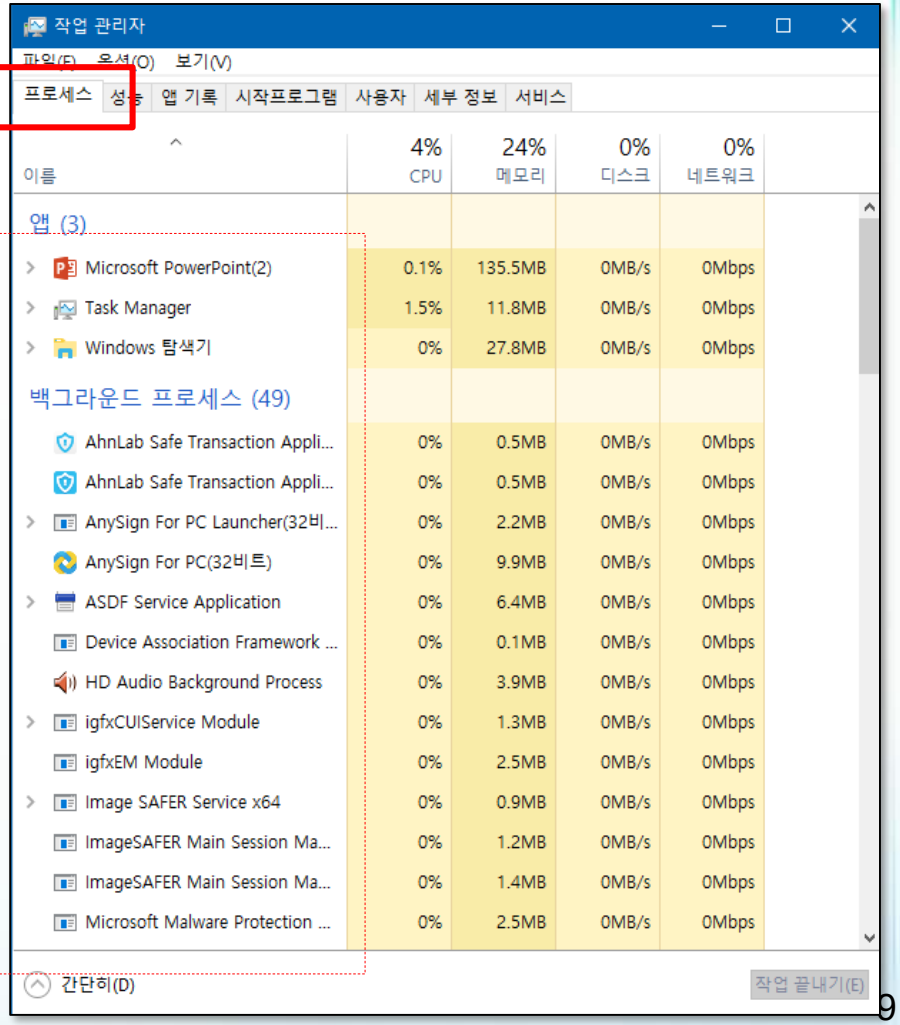
■ 프로세스 구조

☑ Hierarchical Tree Structure, Parent-Child Concept



☑ Process Management

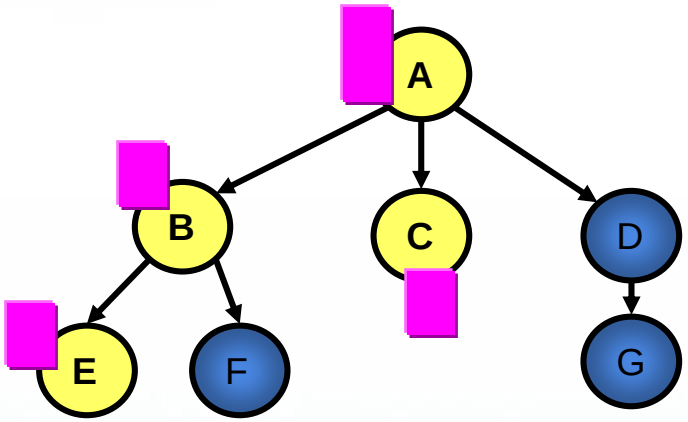
- **프로세스의 생성**(new)**과 종료**(exit)
- **프로세스의 제거**(kill)
- **프로세스의 서스펜드**(stopped)
- **프로세스의 재시작** (resume)
- **프로세스의 우선순위 변경**
- **프로세스의 보류**(blocked)
- **프로세스를 깨움**(wake)



프로세스 관리

■ 프로세스 생성(Creation)

☑ 프로세스 생성의 예



A Process

```
#include  <unistd.h>
main()
{
  int   B,E,C;

  B = fork();

  if (B == 0)
        E =  fork();
  else
        C=fork();
}
```

☑ 생성과정

1) fork() 시스템콜 함수는 PCB( )를 메모리에 생성해준다.

2) Child 프로세스가 생성되면 Parent 프로세스에게 PID값을 넘김

3) Parent 프로세스는 PID값으로 Child 프로세스를 구분함

프로세스 관리

■ 프로세스 생성(Creation)

☑ 예, Oracle 의 Solaris OS, **부팅과정에서 생성되는 프로세스들**…

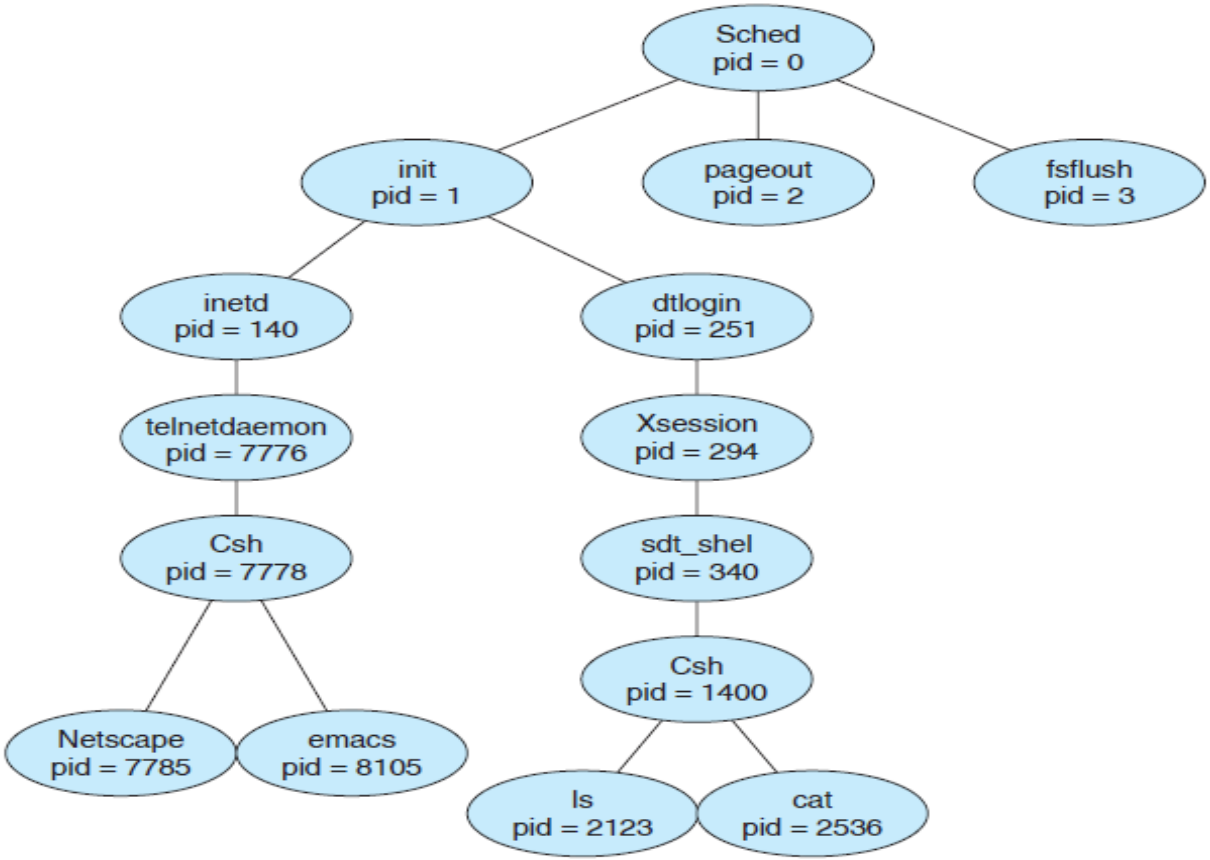


Figure 3.9 A tree of processes on a typical Solaris system.

프로세스 관리

■ 프로세스 종료(Termination)

☑ **프로세스 종료의 예**

정상적인 종료 ; **프로그램의 정상종료**, exit()

비정상적인 종료 ; 1) Timeout

2) **파일입출력 실패(파일조회 최대횟수 초과)**

3) **오류발생(산술오류, 보호오류, 데이터오류)**

4) **메모리부족, 메모리 접근불가 지역의 접근시도**

프로세스가 종료->

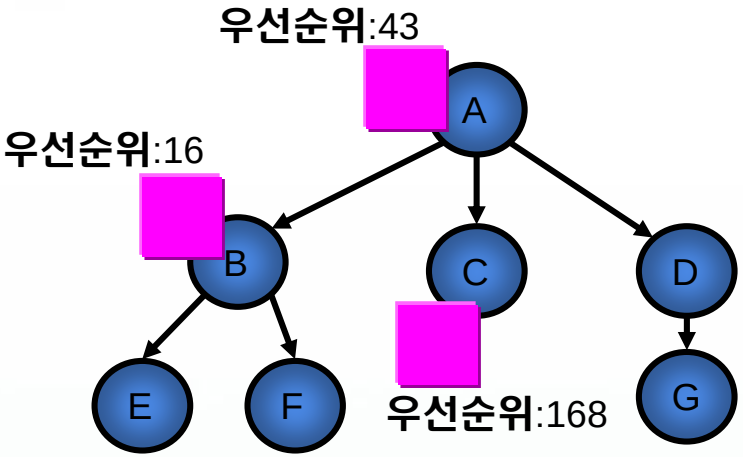
프로세스 제거

(PCB**제거 및 관련자원 회수**)

프로세스 관리

■ 프로세스 우선순위(Priority) 와 관리

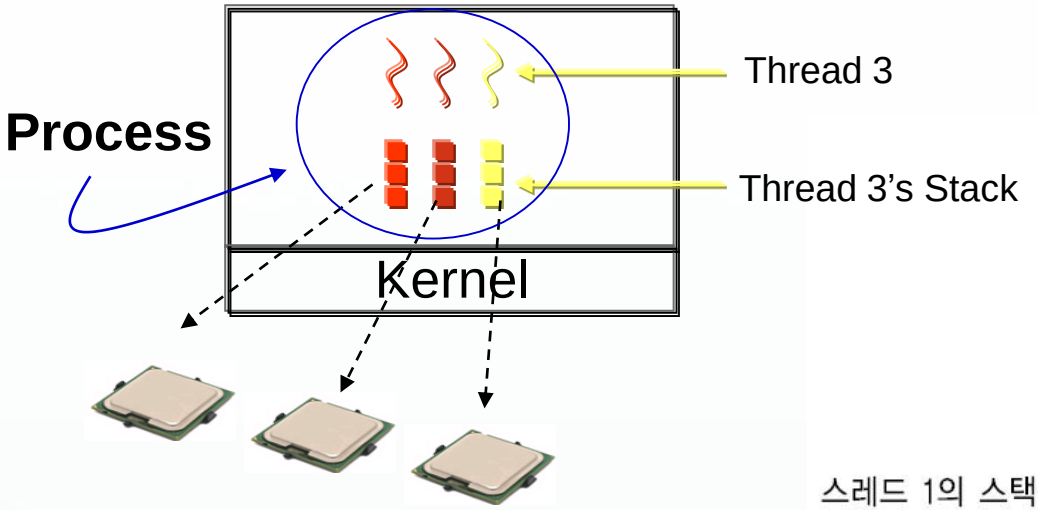
Dispatcher가 CPU를 할당할 프로세스를 선택할때 우선순위를 보고 결정함

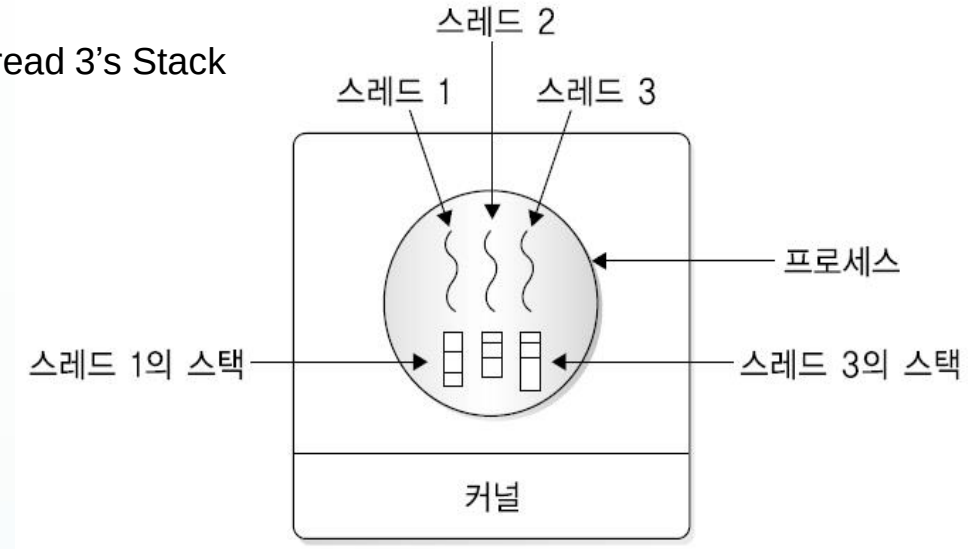


☑ 프로세서중심의 프로세스
☑ 입출력 중심의 프로세스

스레드(Thread)

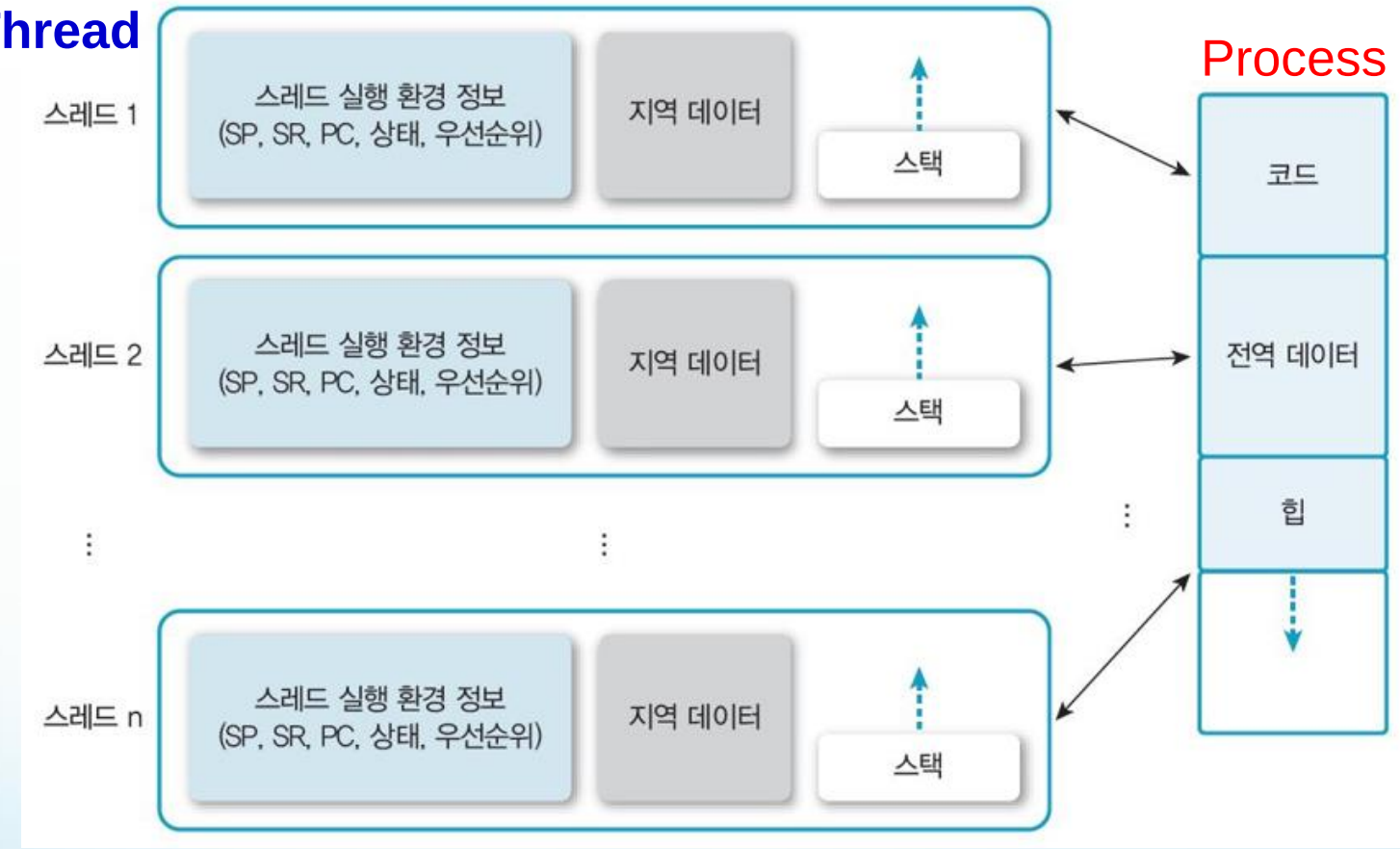
- **스레드**(Thread)
  - 프로세서(Processor)의 이용 **기본 단위**(Process > Threads)
  - 프로그램 명령을 실행하는 **프로세스 내의 개체**
  - 명령어를 실행할 수 있는 하나의 제어 흐름





스레드(Thread)

- **스레드(Thread)의 구조**
  - 1개 Process -> 1개 **이상의** Threads
  - 1개 Proces **내의** Threads **가 모두 종료되면** Process **도 제거됨**
  - **스레드 디스크립터**(threads descriptor)



스레드(Thread)

■ Thread 의 병렬수행

☑ (프로세스 하나에 포함된) 스레드들은 공동의 목적 달성을 위해 병렬수행
☑ 프로세스가 하나인 서로 다른 프로세서에서 프로그램의 다른 부분 동시 실행

■ Thread병렬수행의 잇점

☑ 사용자 응답성 증가
☑ 프로세스의 자원과 메모리 공유 가능
☑ 경제성 좋음
☑ 다중 처리(멀티 프로세싱)로 성능과 효율 향상

■ 스레드/프로세스 모델

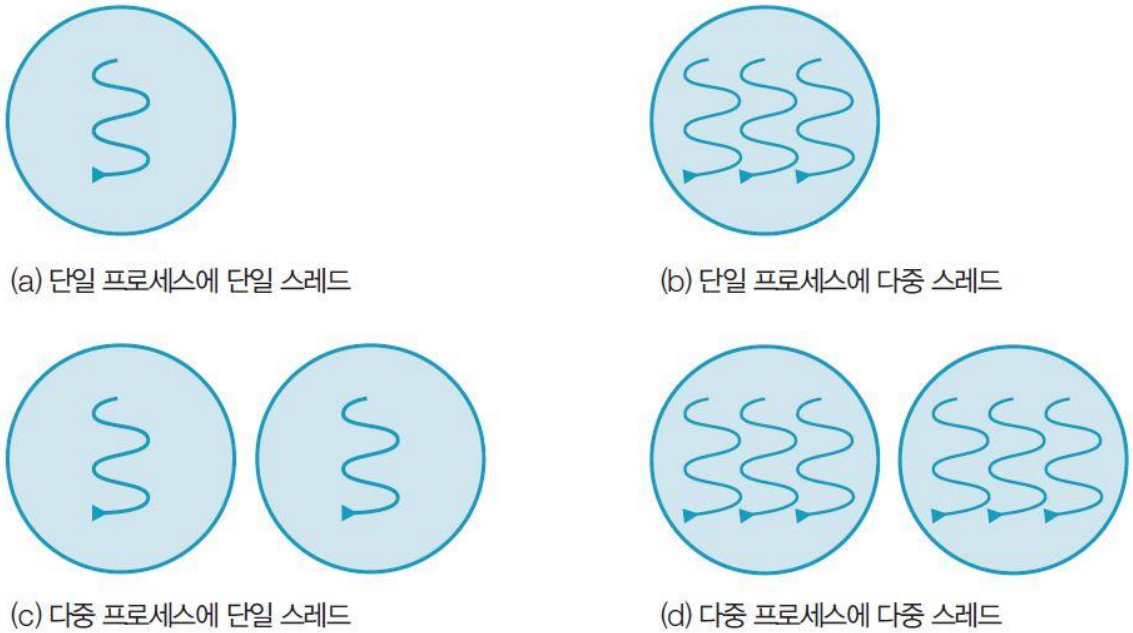


그림 3-14 단일 스레드 프로세스와 다중 스레드 프로세스

스레드(Thread)

■ **단일**(Single) thread vs **다중**(Multi) threads

☑ **단일 스레드 프로세스 모델**

프로세스 = 1 code
+ registers
+ stack

☑ **다중 스레드 프로세스모델**

프로세스 = 1 code
+ (registers
+ stack)
x n-threads

그림 3-15 프로세스 관리 측면에서 살펴본 단일 스레드 프로세스와 다중 스레드 프로세스

■ Thread usage

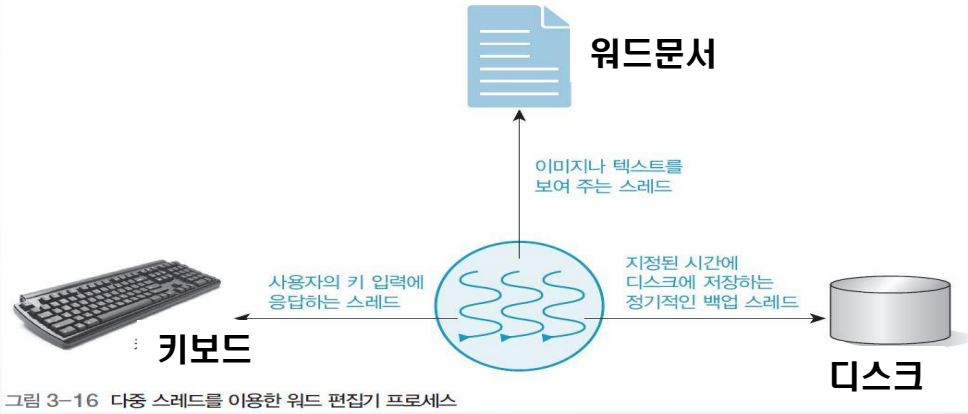


그림 3-16 다중 스레드를 이용한 워드 편집기 프로세스

스레드(Thread)

■ Thread usage

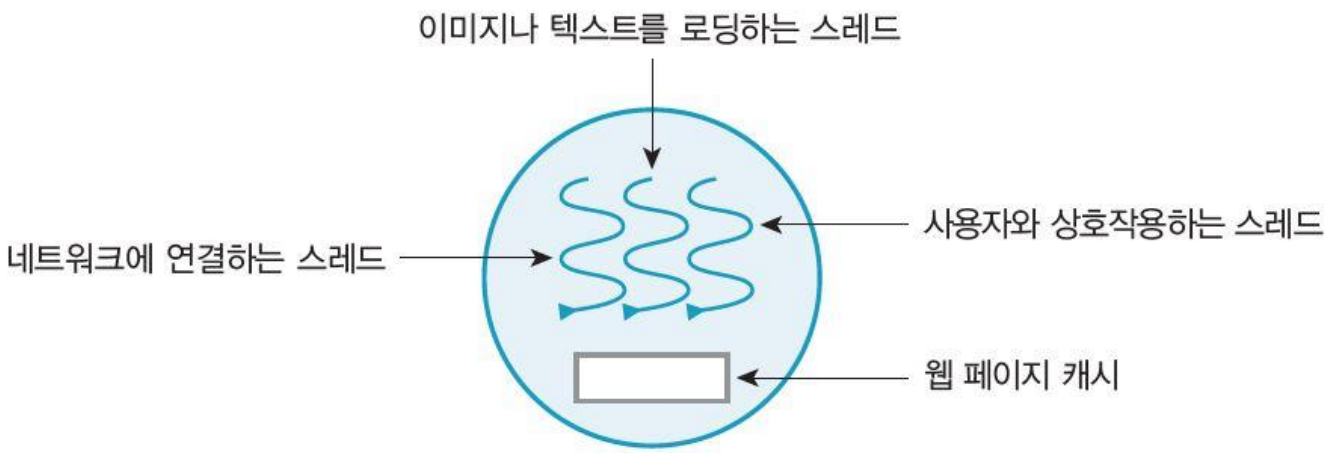


그림 3-17 다중 스레드를 이용한 웹 브라우저 프로세스

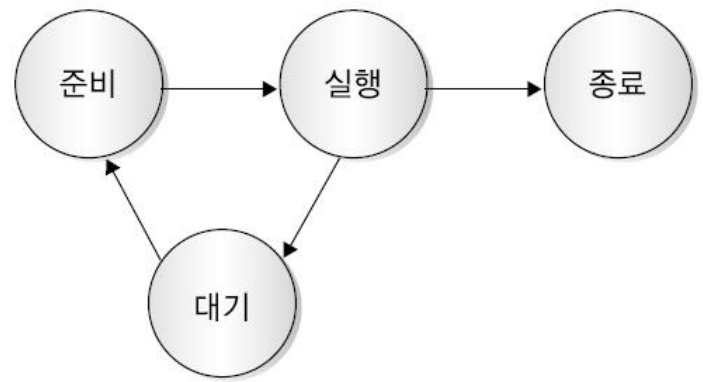
■ Thread state

준비 ; 스레드가 프로세서에 의해 실행될 수 있는 상태.
스레드는 준비 리스트에 삽입됨.

실행 ; 스레드가 프로세서를 점유하여 실행 중인 활성화 상태.

대기 ; 스레드가 이벤트를 기다릴 때,
(입출력 작업 등이 완료될 때까지 대기(보류) 상태)

종료 ; 스레드의 작업이 종료되면
자원해제, 레지스터 문맥과 스택 할당 제거.



스레드(Thread)

■ Thread Control Block (**스레드 제어 블록**)

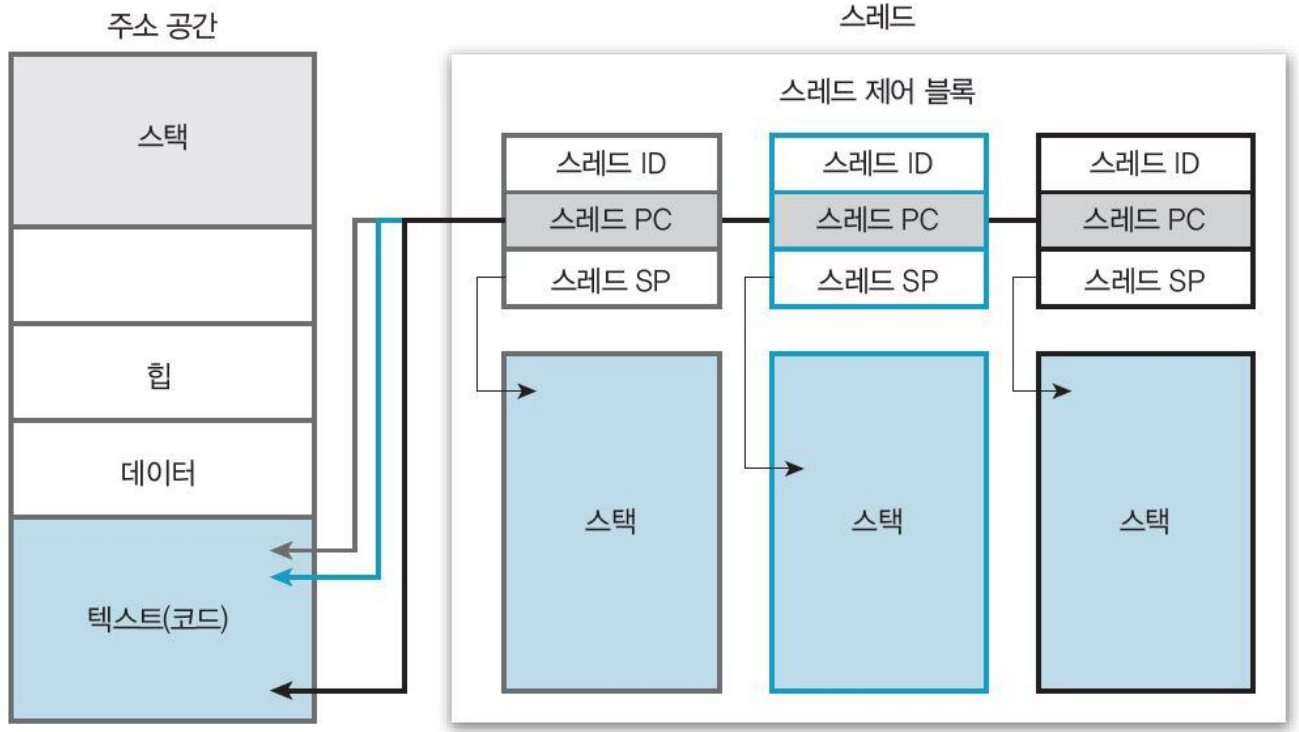


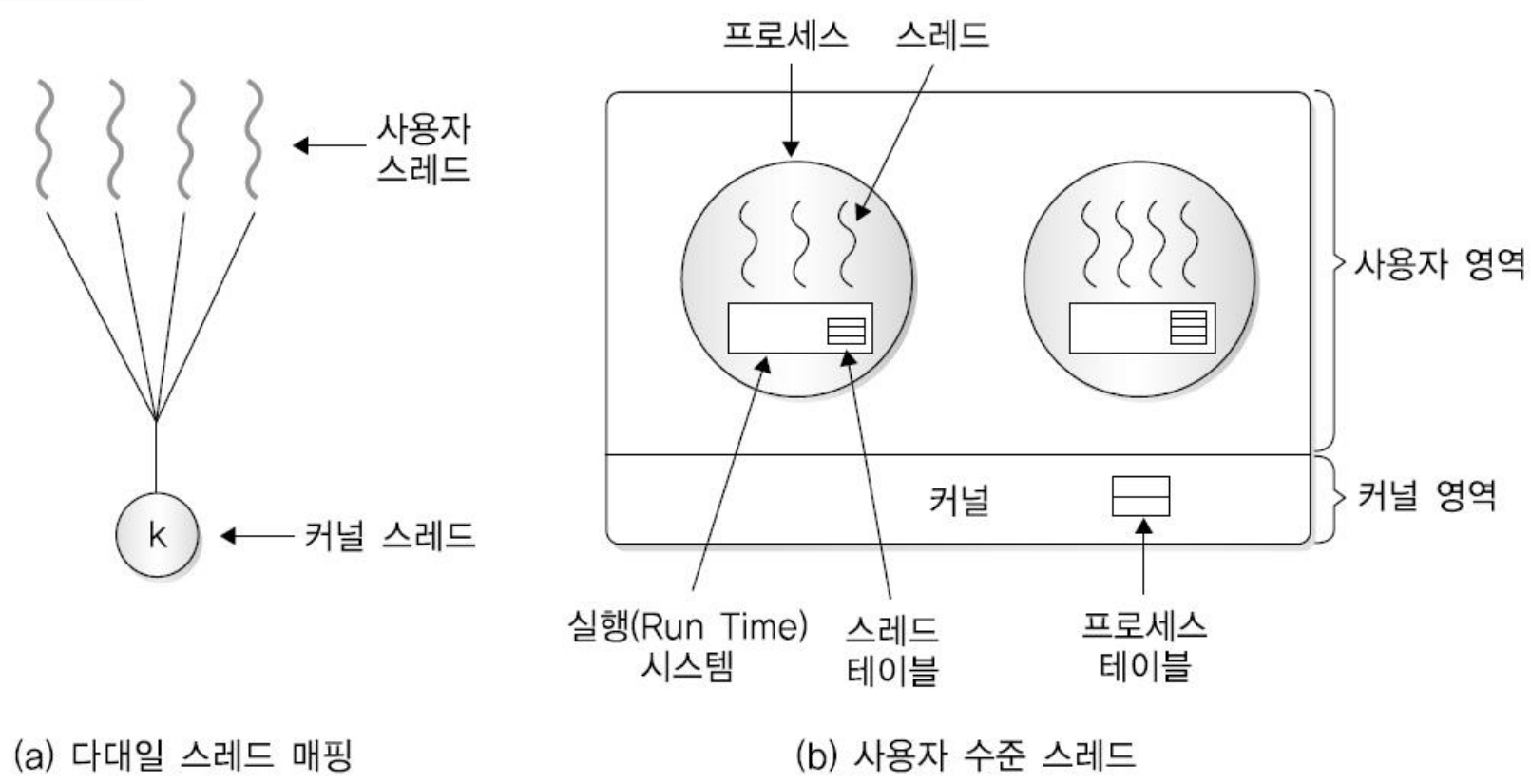
그림 3-19 스레드 제어 블록

☑ **TCB**

- 실행상태,
- 프로세서 레지스터
- 프로그램 카운터
- 스택 포인터
- 스케줄링 정보
- 우선순위
- 프로세서 시간
- 계정 정보
- PCB 포인터

스레드(Thread)

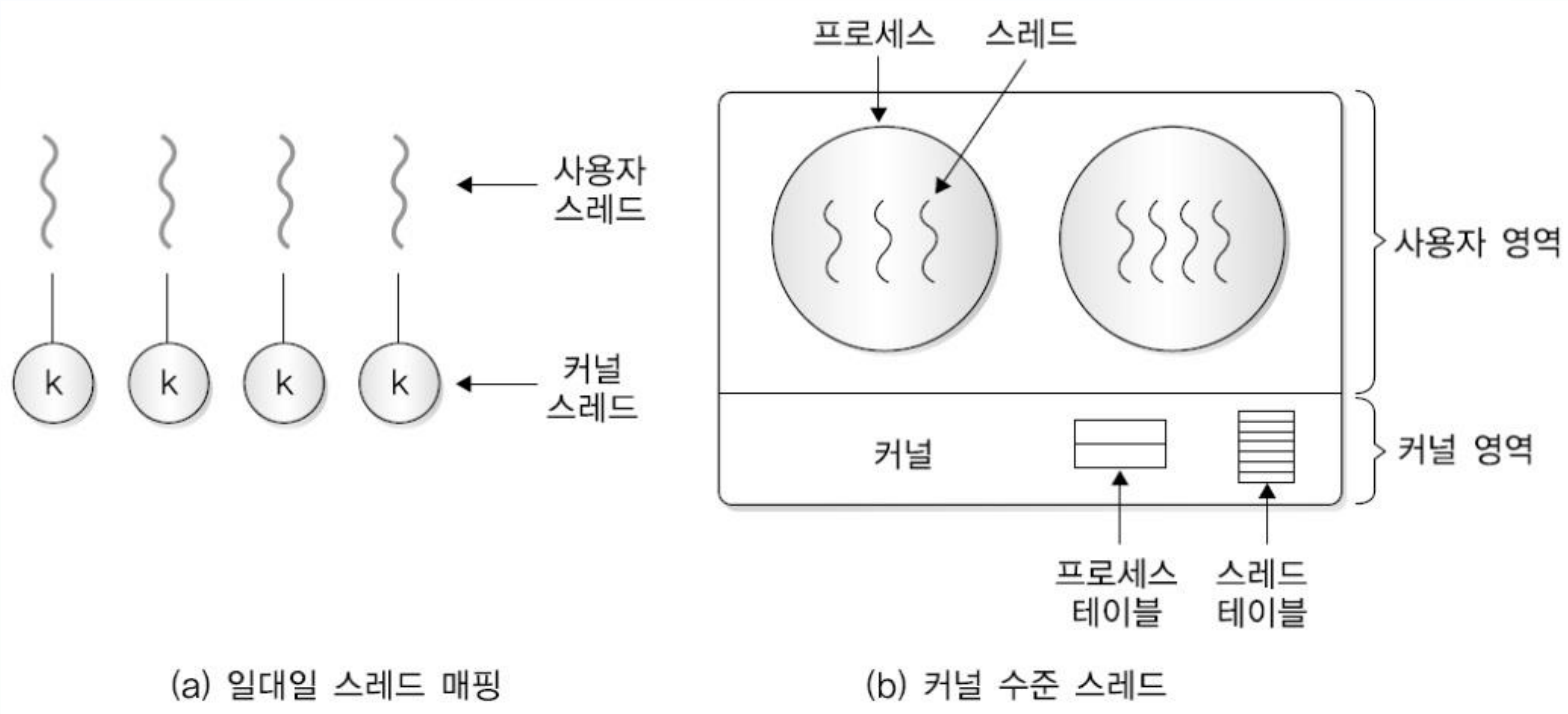
- User level thread



- thread**가 커널영역의 개입없이 사용자 영역에서 실행됨**
- kernel level thread **보다 성능이 좋음**

쓰레드(Thread)

■ Kernel level thread

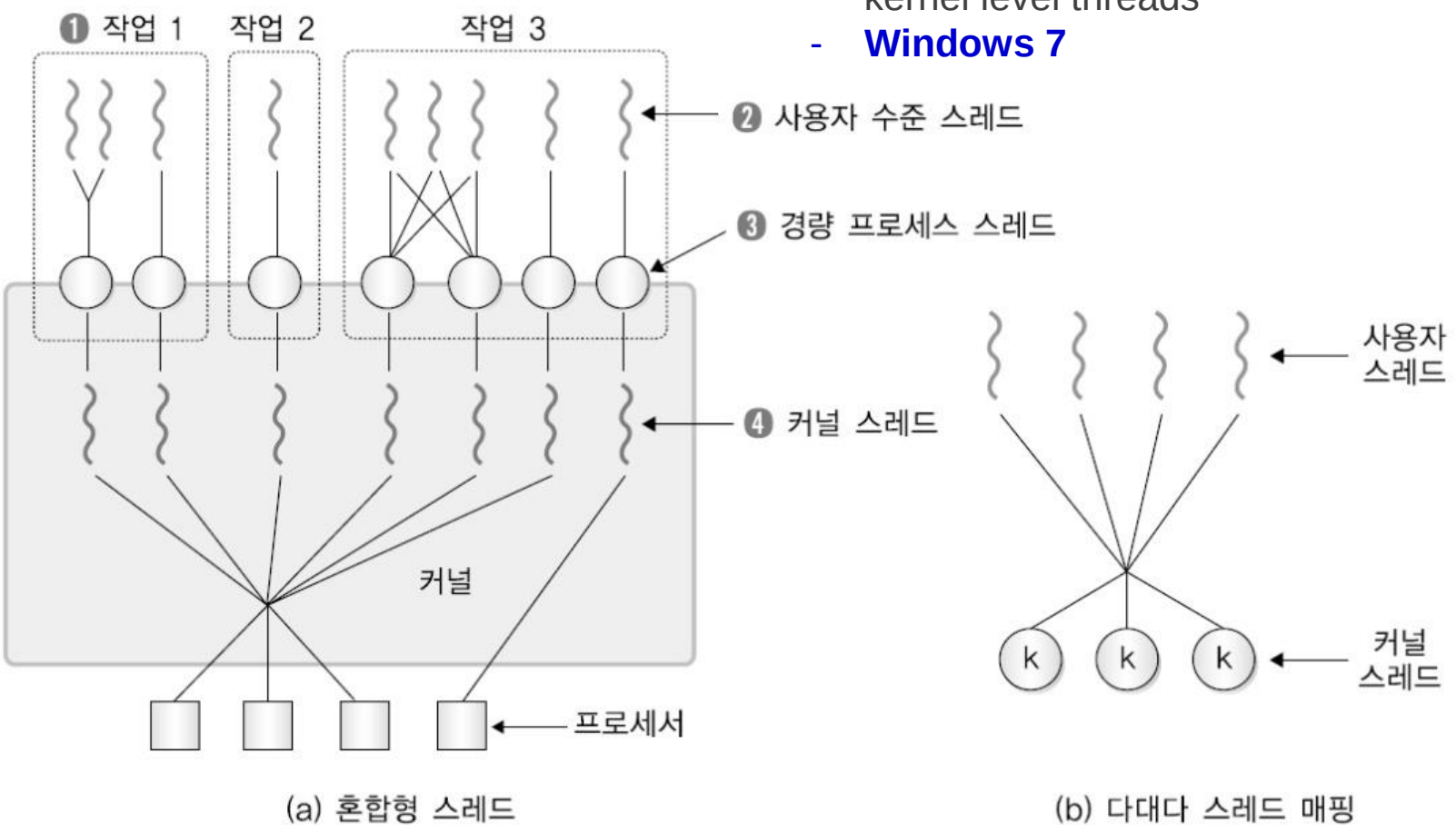


■ user thread**가 요구하는 트랩과 정지 상태의 시스템 호출 등 비동기적인 실행 ⇒ 다른 실행 가능한 스레드의 실행 방해(속도 향상 어려움)**

쓰레드(Thread)

■ **혼합형 방식**(Hybrid level thread)

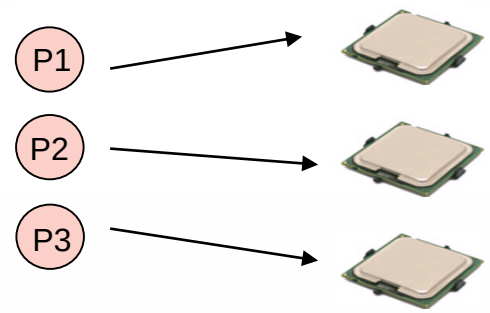
- thread **생성은 사용자 영역에서 시작**
- user level threads –**매핑**→ kernel level threads
- **Windows 7**



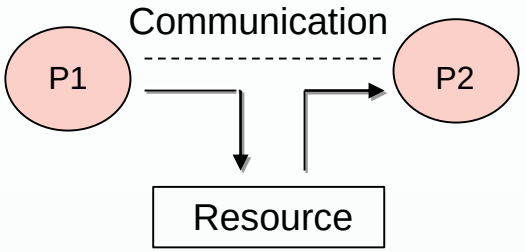
병행프로세스
(Concurrent Process)

병행프로세스(Concurrent Process)

- **Concurrency (병행성) -> 성능향상 !**
  - Multi processing = Parallel Processing
  - Multi processes
  - Multi threads
  - High performance efficiency



- **Concurrent Process**
  - 동시에 실행중인 2개 이상의 프로세스
  - Co-Operation between two or more processes
  - Asynchronous Concurrent Processes



- **Concurrent 가 가능 하려면 …**
  - 공유자원의 배타적 사용을 허용
  - 상호 프로세스 협력/동기화 관리
  - 프로세스간의 상호 통신방식이 필요
  - 상호 실행속도와 관계없이 일정한 실행결과가 나와야
  - 교착상태 해결방법 제시

Process Management

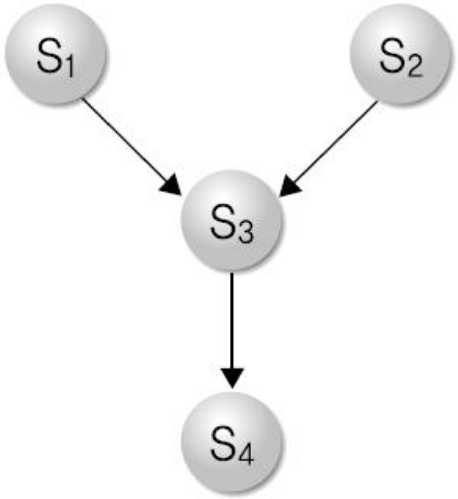
병행프로세스(Concurrent Process)

- **Precedence Graph (선행그래프)**

  - 병행프로세스는 실제로 처리내용이 병행적이어야 …
  - 선행그래프로 병행성을 판별 …
  - 병행성은 선행제약 조건을 따져봐야 …
  - 예) a = x + y , b = z – 1 ➔ 독립적, 동시수행 가능

간단한 산술 연산을 수행하는 알고리즘

```
01   a := x + y   → S1
02   b := z + 1   → S2
03   c := a – b   → S3
04   w := c + 1   → S4
```



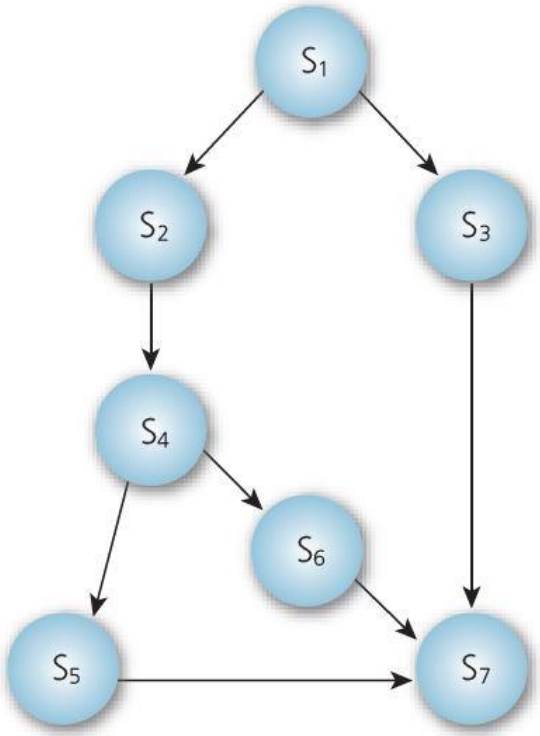
① c := a – b; – a와 b가 값을 할당받기 전에 수행하면 안 된다.

② w := c + 1; – c 값을 계산하기 전에 수행할 수 없다.

③ a := x + y, b := z + 1; – 서로 독립적이므로 동시에 수행할 수 있다.

병행프로세스(Concurrent Process)

- **선행조건(Precedence Constraint)**



(a) 선행 그래프

- S_2와 S_3은 S_1이 끝난 후에 수행한다.
- S_4는 S_2가 끝난 후에 수행한다.
- S_5와 S_6은 S_4가 끝난 후에 수행한다.
- S_7은 S_5, S_6, S_3이 끝난 후에 수행하고, S_3은 S_2, S_4, S_5, S_6과 병행하여 수행할 수 있다.

(b) 선행 관계

그림 4-3 비순환 선행 그래프와 선행 관계 예

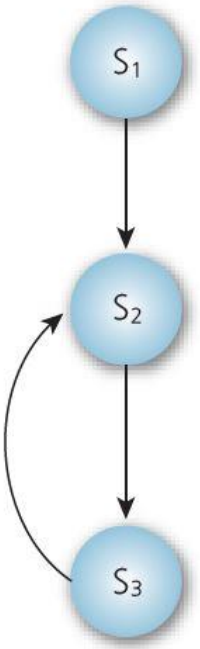


그림 4-4 순환 선행 그래프

s_3는 s_2가 끝난 후에만 수행될 수 있고
s_2는 s_3가 끝난 후에만
수행되므로 모순 발생 !!

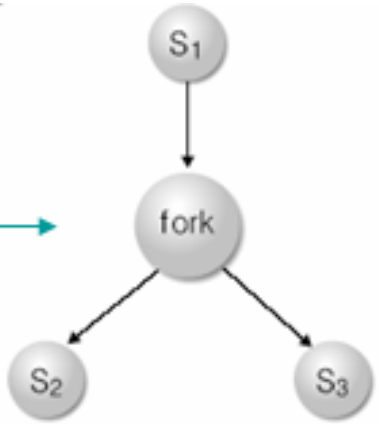
병행프로세스(Concurrent Process)

■ Concurrent Statement in program

☑ fork, join

☑ parbegin, parend

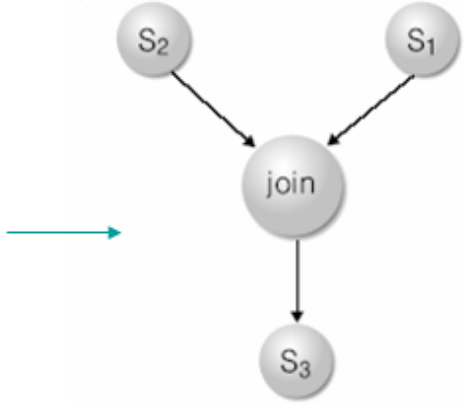
```
    s1 ;
    fork L;
    s2 ;
     .
     .
L : s3 ;
```




```
 :
 :
count := count - 1;
if count ≠ 0 then quit;
```

☑ join [합칠 연산의 개수]

```
    count := 2;
    fork L1;
     :
     :
    S1;
    go to L2;
L1 : S2;
L2 : join count;
     S3;
```



병행프로세스(Concurrent Process)

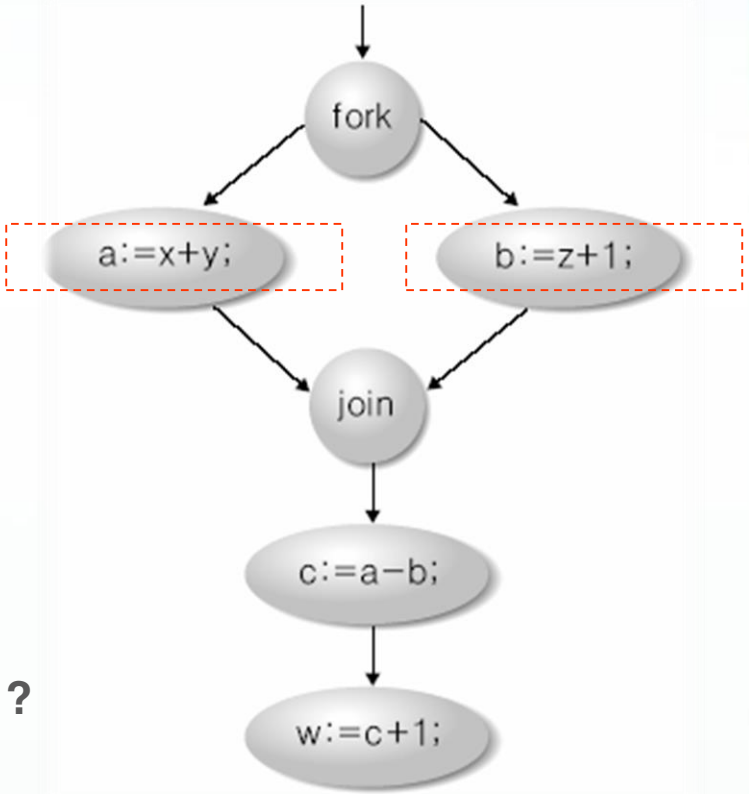
■ fork, join 의 실제 예

[예]

```
a := x + y;
b := z + 1;
c := a - b;
w := c +1;
```

Concurrency

```
   count := 2;
    fork L1;
    a := x + y;
    go to L2;
L1 : b := z +1;
L2 : join count ;
    c := a - b;
    w := c + 1;
```



[문제]

```
sum:= x + y;
test := sum - 2;
p := test * 2 +1;
q := sum * 2;
f:= p + q;
```

1) Precedence Graph ?
2) Fork, join 이용한 코드 ?

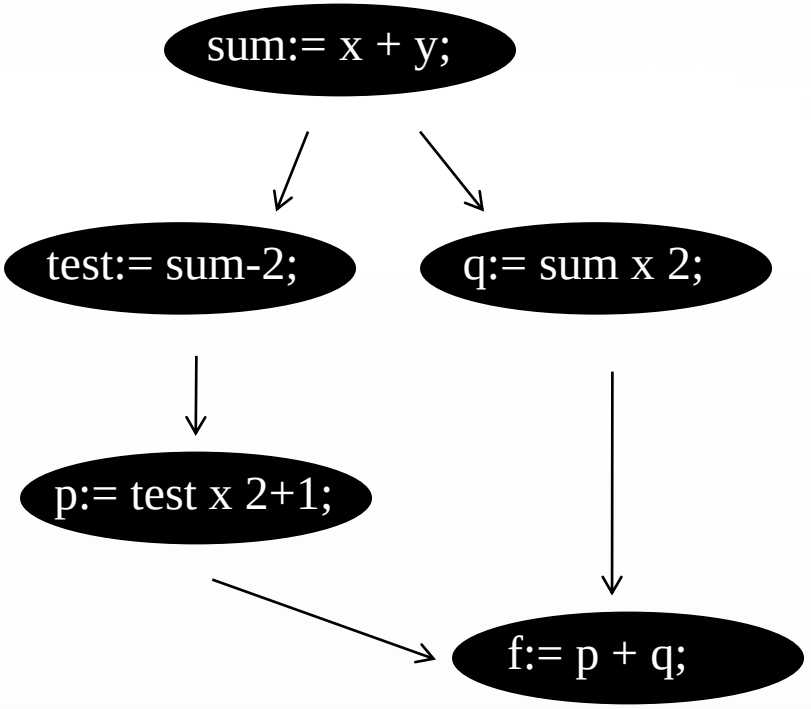
병행프로세스(Concurrent Process)

■ fork, join 예

[문제]

```
sum:= x + y;
test := sum - 2;
p := test x 2 +1;
q := sum x 2;
f:= p + q;
```

```
   count := 2;
     fork  L1;
     test := sum - 2;
     p:= test x 2 +1;
    go to L2;
L1 : q := sum x 2;
L2 : join count ;
    f := p + q;
```



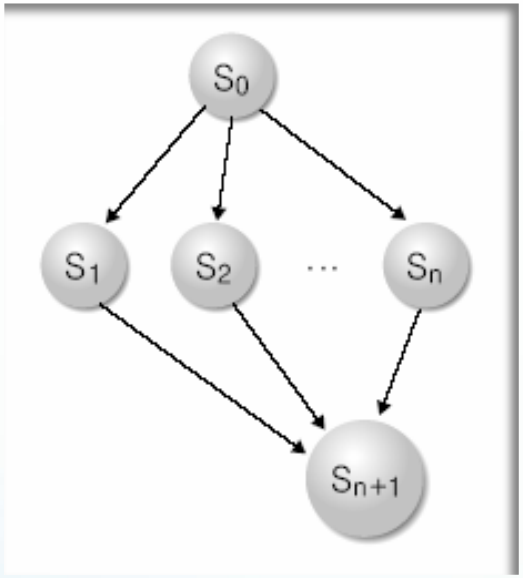
병행프로세스(Concurrent Process)

■ Concurrent Statement in program

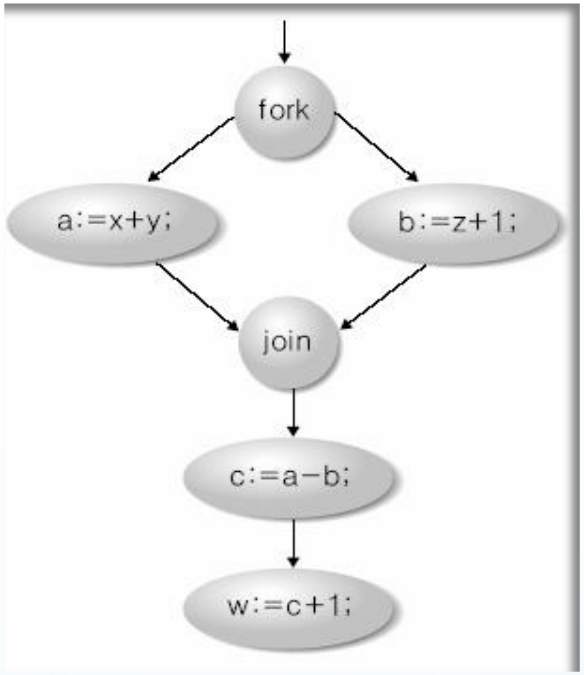
☑ fork, join

☑ **parbegin, parend**

S_0
parbegin $s_1; s_2; \cdots; S_n;$ parend
S_{n+1}



문제) 아래 그림을 parbegin, parend로 작성하여라.



병행프로세스(Concurrent Process)

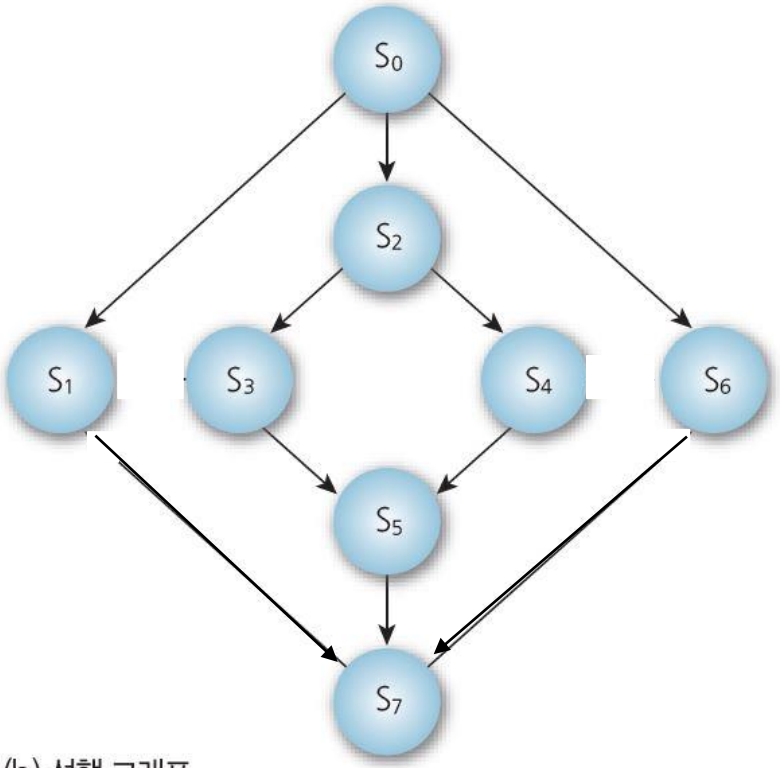
■ Concurrent Statement in program

☑ fork, join

☑ **parbegin, parend**

```
S0;
PARBEGIN
   S1;
   BEGIN
      S2;
      PARBEGIN
         S3;
         S4;
      PAREND;
      S5;
   END;
   S6;
PAREND;
S7;
```

(a) 알고리즘



(b) 선행 그래프

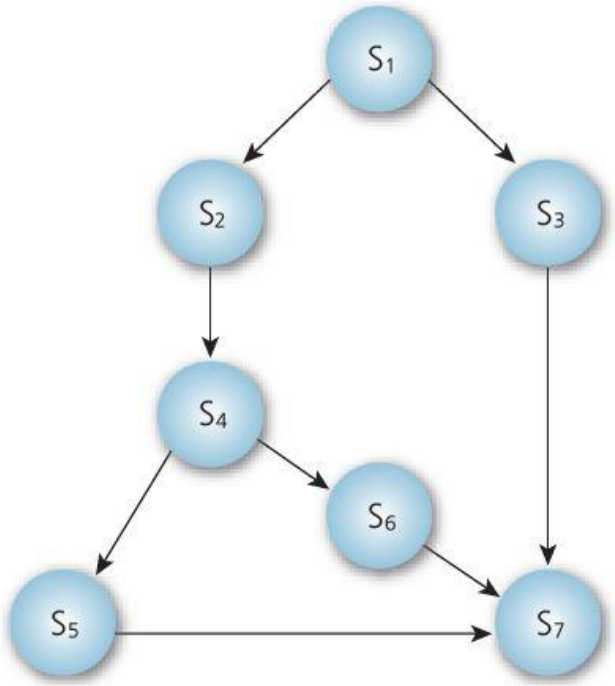
그림 4-10 복잡한 구조의 병행 문장과 선행 그래프

병행프로세스(Concurrent Process)

■ Concurrent Statement in program

☑ fork, join

☑ parbegin, parend



(a) 선행 그래프

?

(b) 알고리즘

그림 4-12 그림 4-3 선행 그래프의 parbegin/parend 구조 알고리즘

Q&A