CHUNGNAM NATIONAL UNIVERSITY

시스템 프로그래밍

강의 8 : 8.1~8.2 예외적인 제어흐름 - 프로세스

http://eslab.cnu.ac.kr

* Some slides are from Original slides of RBE

전달사항

중간고사 리뷰

폭탄랩 진행현황

http://168.188.127.145/sys02.htmll

http://168.188.127.145/sys03.html

다음주 읽기 숙제

8.4.3~8.4.5

구글 Summer of Codes

https://developers.google.com/open-source/soc/?hl=ko&csw=1

좋은 소식

glassdoor

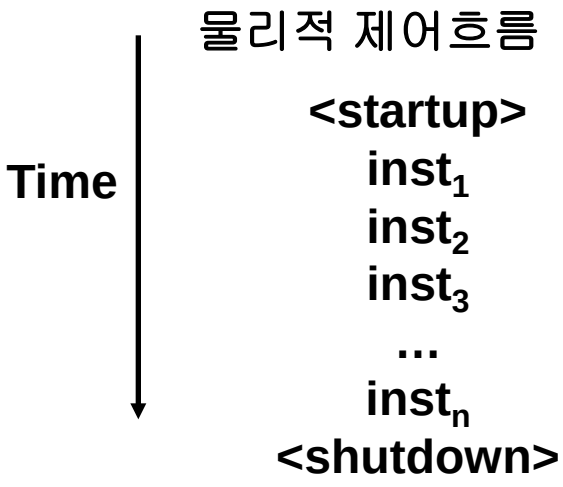
25 HIGHEST PAYING COMPANIES FOR SOFTWARE ENGINEERS 2013

RANK	COMPANY	AVERAGE BASE SALARY
1	JUNIPER NETWORKS	$159,990
2	LinkedIn	$136,427
3	YAHOO!	$130,312
4	Google	$127,143
5		$124,863
6		$124,630
7	ORACLE	$122,905

컴퓨터의 제어 흐름

컴퓨터는 단순한 한가지 일만 한다

- 전원이 들어간 이후에는 명령어(인스트럭션)들만 반복적으로 실행한다. 한번에 한 개씩.
- 이러한 명령어의 실행흐름을 시스템의 물리적인 제어 흐름이라고 한다.



제어흐름의 변경

제어흐름을 변경하는 방법
- Jumps 와 branches 명령어
- 스택을 사용한 Call 과 return 명령어

이 정도로는 쓸만한 시스템을 만들기에는 부족하다
- CPU가 시스템의 상태변화에 대응하도록 하기는 어렵다.
  - 하드디스크나 네트워크 어댑터에 데이터가 수신된 경우
  - 0으로 나누기를 시도할 때
  - 사용자가 CTRL-C를 눌렀을 때
  - 시스템 타이머가 초과되었을 때

시스템은 예외적인 제어흐름을 위한 메커니즘을 필요로 한다 “exceptional control flow”

비동기형 예외(인터럽트)

프로세서의 외부사건으로부터 발생

- 프로세서의 인터럽트 핀을 세팅 해서 발생을 표시
- 핸들러 실행 후, 인터럽트 직전 실행 명령어 다음 명령어로 복귀

예 :

- 입출력 인터럽트
  - 키보드에서 ctrl-c 를 누른다
  - 네트워크에서 패킷이 들어왔다
  - 디스크에서 한 개의 섹터가 읽혀 들어왔다
- 하드 리셋 인터럽트
  - 컴의 리셋 단추를 눌렀다
- 소프트 리셋 인터럽트
  - 컴에서 ctl-alt-del 을 눌렀다

*** 인터럽트 동작 이해 중요**

동기형 예외

명령어를 실행한 결과로 발생하는 사건들

- Traps
  - 명령어의 결과로 발생하는 의도적인 예외
  - 예 :system calls, breakpoint traps, special instructions
  - 처리 후 "다음" 명령어로 복귀
- Faults
  - 핸들러가 정정할 수 있는 에러의 결과로 발생
  - 예: page faults (회복가능), protection faults (회복불가), floating point exceptions.
  - Fault 를 일으킨 명령을 다시 실행하거나, Abort 한다.
- Aborts
  - 하드웨어 오류와 같이 복구 불가능한 에러의 결과로 발생
  - 예: 패러티 에러, 시스템 체크 에러.
  - 응용 프로그램으로 복귀할 수 없다
  - 현재 프로그램을 종료한다

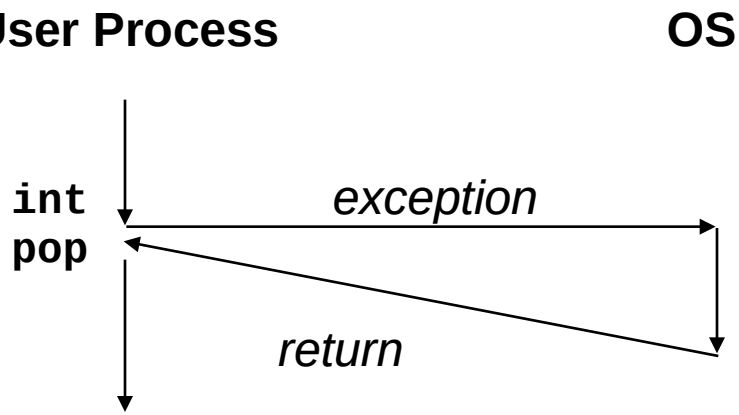
Trap 예제

파일 오픈

- 사용자가 open(filename, options)을 호출하는 경우

```
0804d070 <__libc_open>:
 . . .
 804d082:       cd 80                    int     $0x80
 804d084:       5b                       pop     %ebx
 . . .
```

  - open 함수는 시스템 콜 명령어 int를 실행한다
- OS 는 파일을 찾거나 생성해야 하고, 읽기나 쓰기 동작을 위해 준비작업을 수행
- 정수형의 파일 식별자를 리턴



Fault 예제

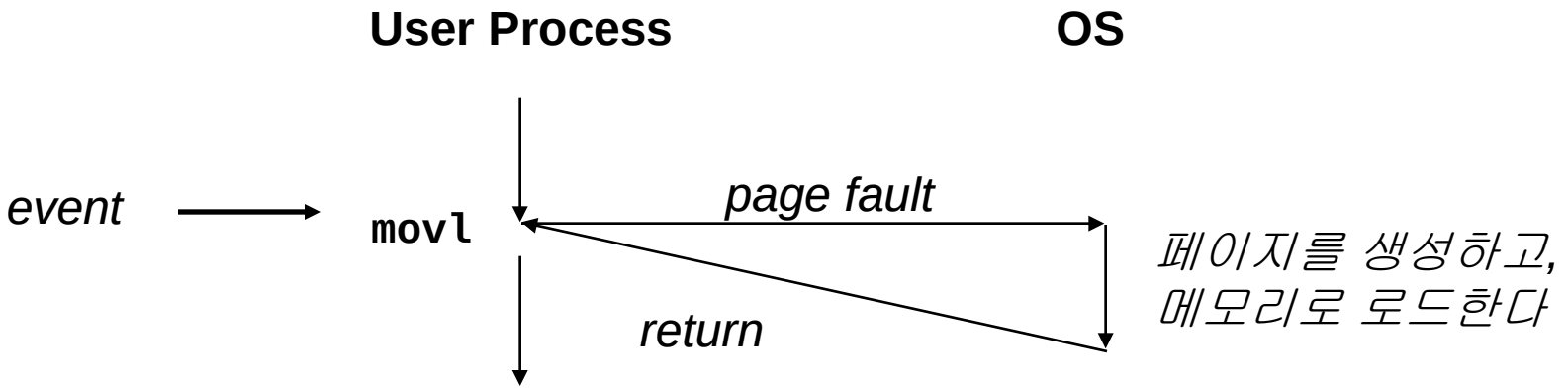
```
int a[1000];
main ()
{
    a[500] = 13;
}
```

메모리 참조시

- 사용자는 메모리에 쓰기작업 수행
- 사용자 메모리의 특정 페이지가 현재 하드디스크에 위치하는 경우

```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```

- 페이지 핸들러는 해당 페이지를 물리메모리에 로드해야 한다
- 이 때 페이지 오류가 발생한다
- 오류 처리후에 오류를 발생시킨 명령어를 다시 실행한다
- 다시 실행할 때에는 접근이 성공한다



프로세스Processes

정의 : 프로세스는 실행하고 있는 프로그램의 한 실행 예이다

- 컴퓨터과학 분야에서 가장 심오한 개념중의 하나
- 프로세스와 프로세서를 혼돈하지 마라

프로세스는 프로그램에 두 개의 중요한 추상화를 제공한다:

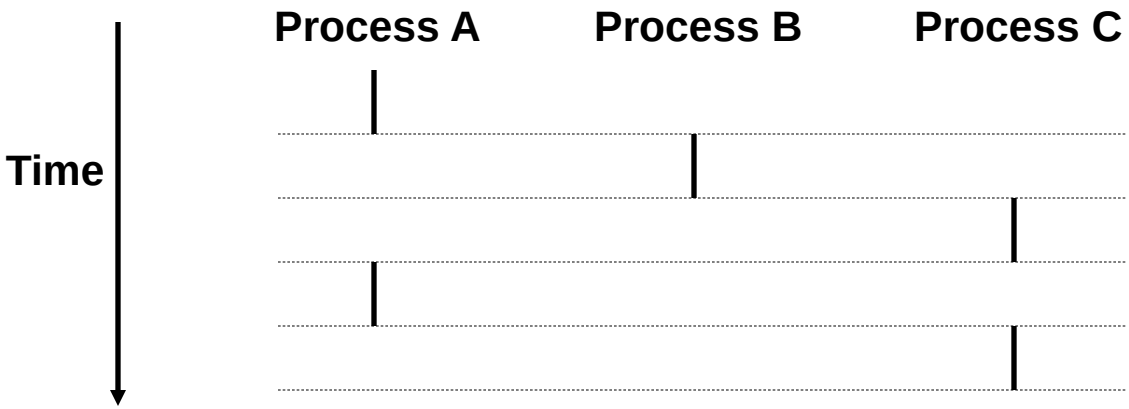
- 논리적인 제어흐름
  - 각 프로그램이 CPU를 독점하는 것처럼 보이도록 한다.
- 사적인 주소공간
  - 각 프로그램이 주 메모리를 독점하는 것처럼 보이도록 한다.

어떻게 이러한 착시가 가능한가?

- 프로세스의 실행이 서로 교대로 실행된다(interleaved , multitasking)
- 주소공간은 가상메모리 시스템에 의해 관리된다

논리적 제어흐름

각 프로세스는 자신만의 논리적인 제어흐름을 갖는다



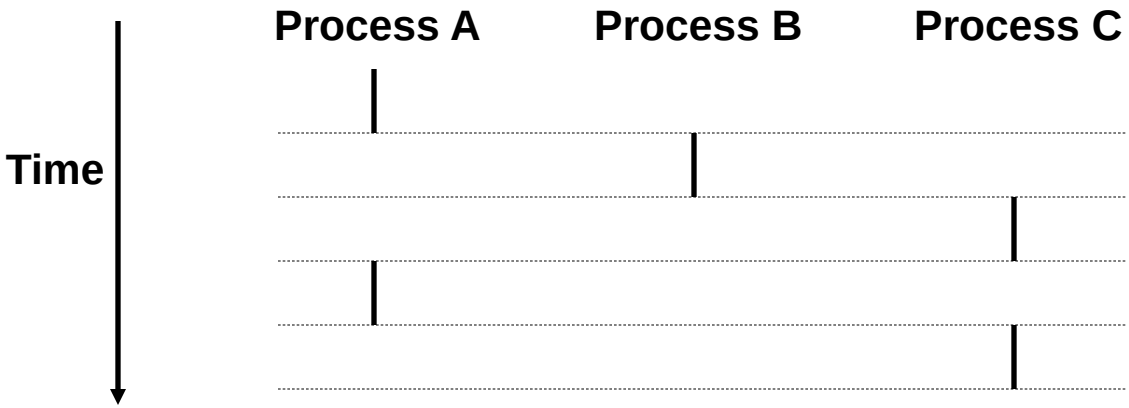
동시성 프로세스

두 프로세스는 그들의 실행 시간이 서로 중첩되면, 동시에 실행된다고 부른다.(*are concurrent)*

그렇지 않다면, 순차적으로 실행된다고 정의한다 (*sequential.*)

Examples:

- 동시실행: A & B, A & C
- 순차실행: B & C



연습문제 1. 동시성 프로세스

다음과 같은 세 프로세스의 시작시간과 종료시간이 주어졌다.

Process	Start time	End time
A	0	2
B	1	4
C	3	5

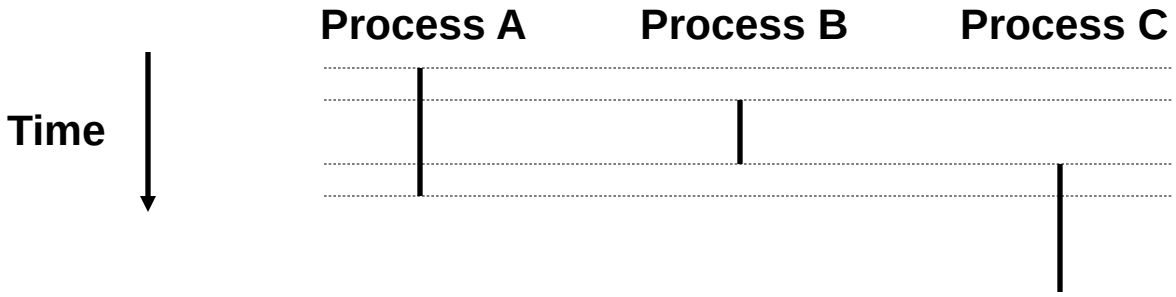
아래와 같은 프로세스들 조합이 동시성 실행이 가능한지 여부를 결정하라

Process pair	Concurrent?
AB	_____
AC	_____
BC	_____

동시프로세스의 사용자 관점

동시 프로세스들을 위한 제어흐름은 시간상으로는 물리적으로 분리된다.

그러나, 동시프로세스들이 서로 병렬로 실행된다고 생각할 수 있다.



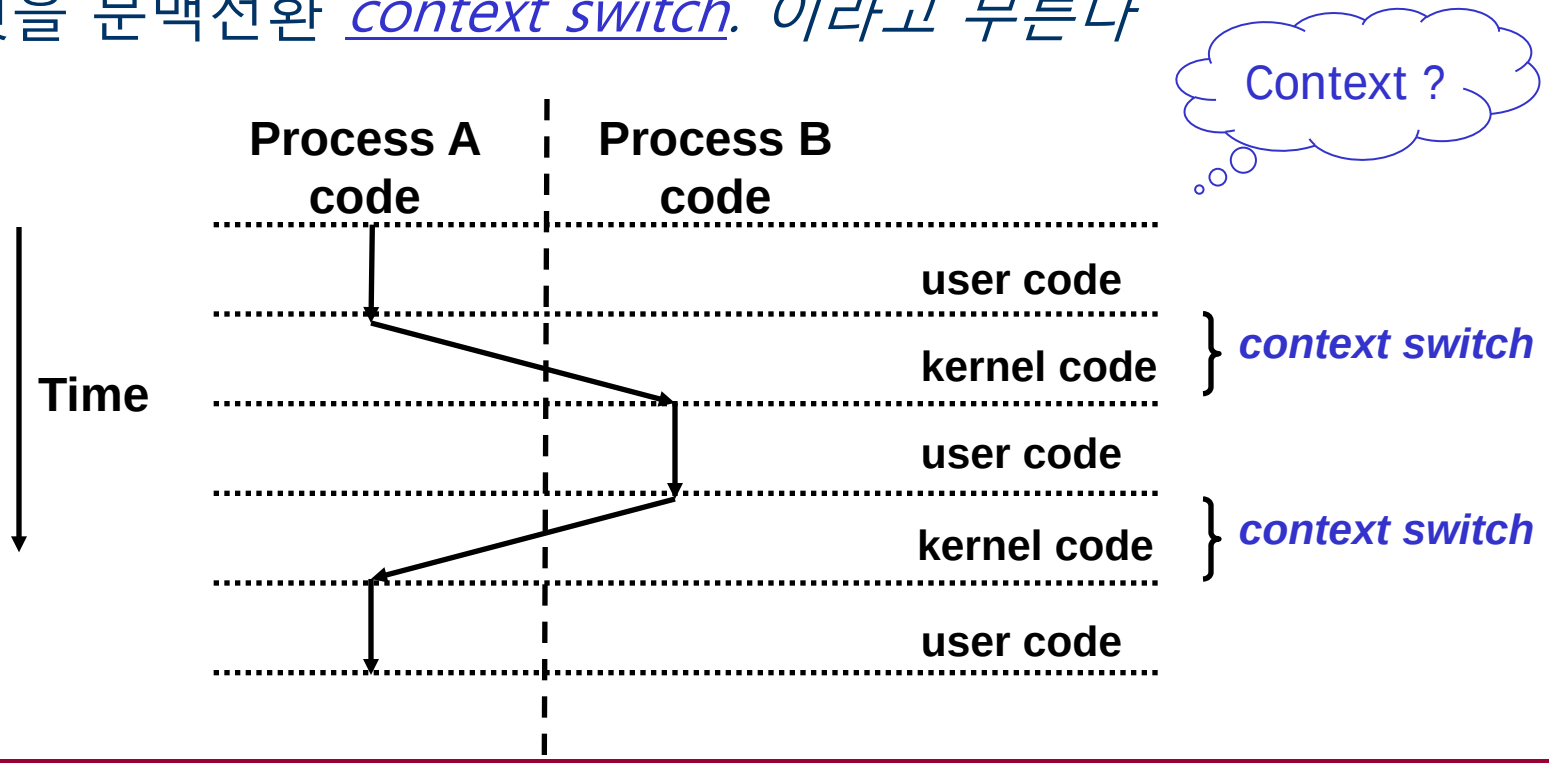
*** 멀티 태스킹 또는 타임 슬라이싱이라고 부름**

문맥전환(Context Switch)

프로세스는 커널이라고 부르는 운영체제에 의해서 관리된다

- 중요 : 커널은 프로세스가 아니며, 유저 프로세스의 일부분으로 실행된다

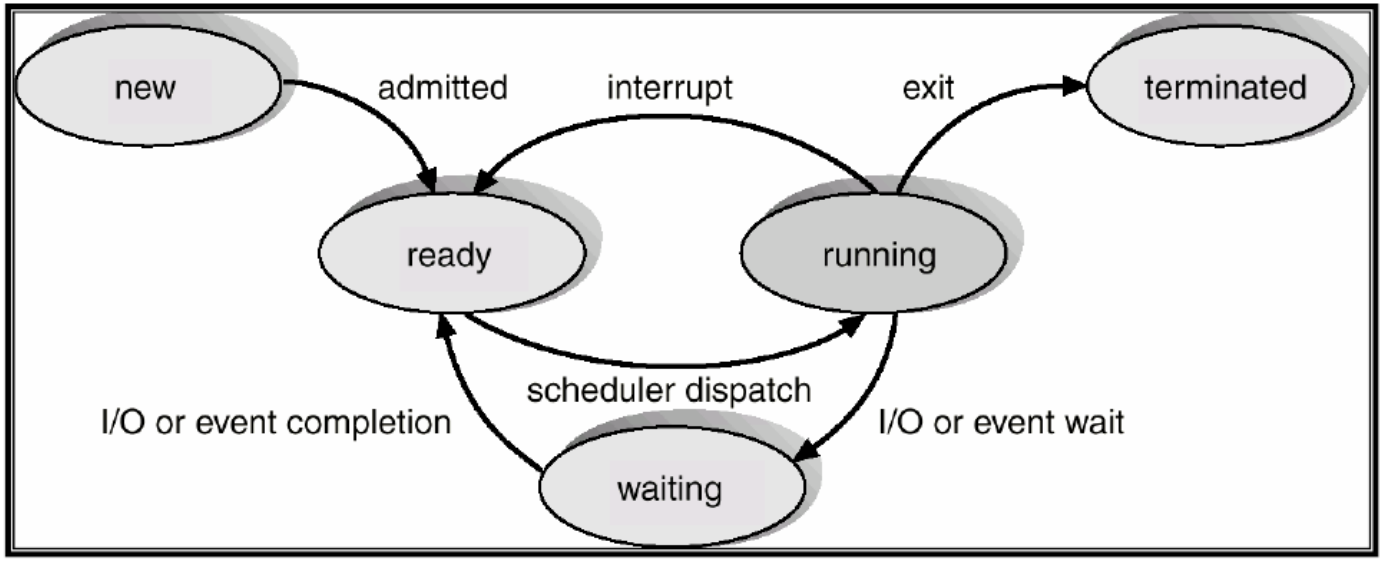
한 개의 프로세스에서 다른 프로세스로 제어흐름이 넘어가는 것을 문맥전환 *context switch. 이라고 부른다*



시스템 콜(System Calls)

- Unix/리눅스 시스템은 시스템 콜을 제공한다
  - 커널의 서비스를 이용할 때, 응용 프로그램에서 호출하는 함수
  - 예. 파일을 읽을 때, 새로운 프로세스를 만들때
  - 리눅스에는 약 160여개의 시스템 콜이 있다
  - `man syscalls`
- 시스템 콜의 호출방법
  - C에서, `_syscall` 매크로를 이용하면 직접 호출 가능
  - 시스템 콜은 어떻게 구현되어 있는가?
  - 시스템 콜을 직접 호출하는 것은 바람직하지 않다. => Why ?
  - 표준 C 라이브러리는 랩퍼 함수들을 제공한다.

프로세스 상태



- 정지 또는 대기 **Stopped or waiting**
  - **SIGSTOP, SIGSTP, SIGTTIN or SIGTTOU** 과 같은 시그널을 받으면, 프로세스는 정지하며, **SIGCONT** 시그널을 받으면, 다시 실행된다
  - 시그널은 일종의 소프트웨어 인터럽트다
- 종료 **Terminated**
  - 종료 시그널을 수신했을 때
  - 메인 함수에서 리턴했을 때
  - **exit** 함수를 호출했을 때

프로세스의 제어

Unix 는 C 프로그램을 이용해서 다음과 같은 프로세스 제어 기능을 제공한다

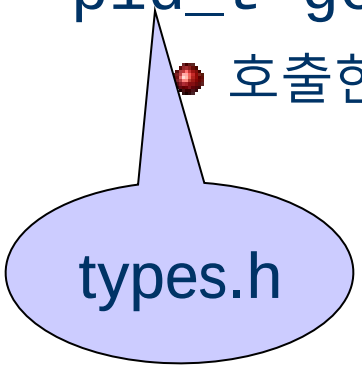
- process ID 를 가져온다
- 프로세스를 만들거나 종료한다
- 자식 프로세스를 제거한다 Reaping child processes
- 프로그램의 로딩 및 실행

Process ID 가져오기

각 프로세스는 프로세스 ID를 갖는다

```
pid_t getpid(void)
pid_t getppid(void)
```

- 호출한 프로세스 또는 그 부모 프로세스의 PID 를 리턴



fork: 프로세스 만들기

int fork(void)

- 호출하는 프로세스(부모 프로세스)와 동일한 새 프로세스(자식 프로세스)를 생성
- 자식 프로세스는 0을 리턴
- 부모 프로세스는 pid 을 리턴

```
if (fork() == 0) {
    printf("hello from child\n");
} else {
    printf("hello from parent\n");
}
```

Fork 함수는 한번 호출하지만, 리턴은 두번 된다는 점이 특이하다

Fork Example #1

Key Points

- 부모와 자식은 동일한 코드를 실행한다
  - fork 로부터의 리턴 값으로 부모와 자식을 구분
- 부모와 자식은 동일한 상태로 시작하지만, 각각의 사본을 갖는다
  - 출력 파일 식별자도 공유
  - 각각의 출력문의 실행 순서는 랜덤

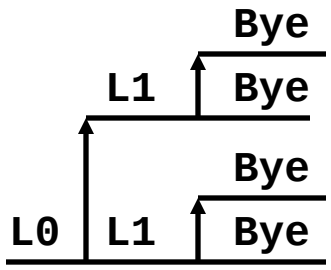
```
void fork1()
{
    int x = 1;
    pid_t pid = fork();
    if (pid == 0) {
       printf("Child has x = %d\n", ++x);
    } else {
       printf("Parent has x = %d\n", --x);
    }
    printf("Bye from process %d with x = %d\n", getpid(), x);
}
```

Fork Example #2

Key Points

- 부모와 자식이 계속 fork 하는 경우

```
void fork2()
{
    printf("L0\n");
    fork();
    printf("L1\n");
    fork();
    printf("Bye\n");
}
```



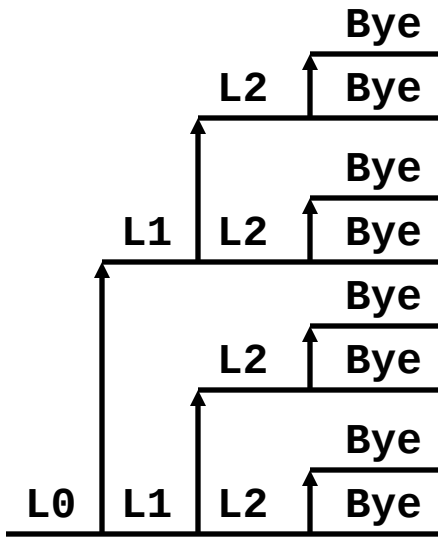
프로세스 그래프를 그려서 생각하면 편리

Fork Example #3

Key Points

- 부모와 자식이 계속 fork 하는 경우

```
void fork3()
{
    printf("L0\n");
    fork();
    printf("L1\n");
    fork();
    printf("L2\n");
    fork();
    printf("Bye\n");
}
```



exit: 프로세스 종료하기

void exit(int status)

- 종료 상태 status 값을 가지고 종료
  - 정상 리턴시 status 0
- atexit() 함수는 exit 할 때 실행할 함수를 등록

```
void cleanup(void) {
   printf("cleaning up\n");
}

void fork6() {
   atexit(cleanup);
   fork();
   exit(0);
}
```

연습문제 2. fork

code/ecf/forkprob0.c

```
1   #include "csapp.h"
2
3   int main()
4   {
5       int x = 1;
6
7       if (Fork() == 0)
8           printf("printf1: x=%d\n", ++x);
9       printf("printf2: x=%d\n", --x);
10      exit(0);
11  }
```

code/ecf/forkprob0.c

a) 위 프로그램에서 자식 프로세스의 출력을 쓰시오

b) 위 프로그램에서 부모 프로세스의 출력을 쓰시오.

좀비(Zombie)

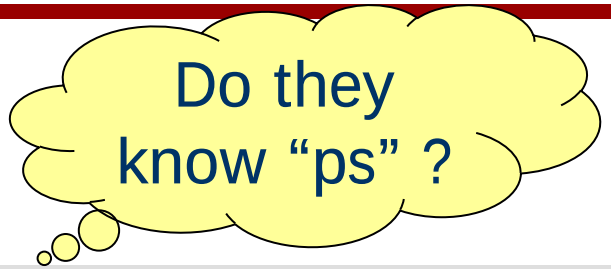
Ide

프로

부모

why ?

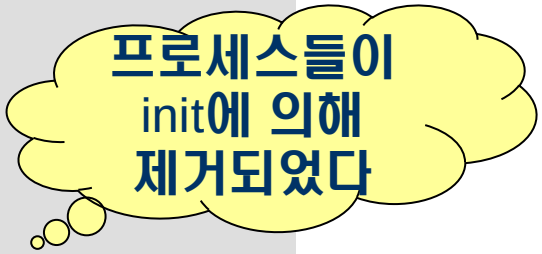
Zombie Example 1




```
void fork7()
{
    if (fork() == 0) {
        /* Child */
        printf("Terminating Child, PID = %d\n",
               getpid());
        exit(0);
    } else {
        printf("Running Parent, PID = %d\n",
               getpid());
        while (1)
            ; /* Infinite loop */
    }
}
```

```
linux> ./fork7 &
[1] 6639
Running Parent, PID = 6639
Terminating Child, PID = 6640
linux> ps
  PID TTY          TIME CMD
 6585 ttyp9    00:00:00 tcsh
 6639 ttyp9    00:00:03 fork7
 6640 ttyp9    00:00:00 fork7 <defunct>
 6641 ttyp9    00:00:00 ps
linux> kill 6639
[1]    Terminated
linux> ps
  PID TTY          TIME CMD
 6585 ttyp9    00:00:00 tcsh
 6642 ttyp9    00:00:00 ps
```

- ps 명령어를 치면, 자식 프로세스가 “defunct” 로 나옴
- 부모 프로세스를 죽이면, 자식 프로세스가 제거된다



Zombie Example 2

부모가 종료되었지만, 자식 프로세스가 여전히 살아있는 경우

- 직접 삭제하지 않으면, 무한 동작한다

```
linux> ./fork8
Terminating Parent, PID = 6(
Running Child, PID = 6676
linux> ps
  PID TTY          TIME CMD
 6585 ttyp9    00:00:00 tcsl
 6676 ttyp9    00:00:06 forl
 6677 ttyp9    00:00:00 ps
linux> kill 6676
linux> ps
  PID TTY          TIME CMD
 6585 ttyp9    00:00:00 tcsl
 6678 ttyp9    00:00:00 ps
```

```
void fork8()
{
    if (fork() == 0) {
        /* Child */
        printf("Running Child, PID = %d\n",
               getpid());
        while (1)
            ; /* Infinite loop */
    } else {
        printf("Terminating Parent, PID = %d\n",
               getpid());
        exit(0);
    }
}
```