11장 시그널

11.1 시그널

시그널

- 시그널은 예기치 않은 사건이 발생할 때 이를 알리는 소프트웨어 인터럽트이다.
- 시그널 발생 예
  - SIGFPE 부동소수점 오류
  - SIGPWR 정전
  - SIGALRM 알람시계 울림
  - SIGCHLD 자식 프로세스 종료
  - SIGINT 키보드로부터 종료 요청 (Ctrl-C)
  - SIGSTP 키보드로부터 정지 요청 (Ctrl-Z)

시그널 종류

- 총 31개의 시그널
- /usr/include/asm/siganl.h

```
#define SIGHUP           1
#define SIGINT           2
#define SIGQUIT          3
#define SIGILL           4
#define SIGTRAP          5
#define SIGABRT          6
#define SIGIOT           6
#define SIGBUS           7
#define SIGFPE           8
#define SIGKILL          9
#define SIGUSR1         10
#define SIGSEGV         11
#define SIGUSR2         12
#define SIGPIPE         13
#define SIGALRM         14
#define SIGTERM         15
#define SIGSTKFLT       16
#define SIGCHLD         17
#define SIGCONT         18
#define SIGSTOP         19
#define SIGTSTP         20
#define SIGTTIN         21
#define SIGTTOU         22
#define SIGURG          23
#define SIGXCPU         24
#define SIGXFSZ         25
#define SIGVTALRM       26
#define SIGPROF         27
#define SIGWINCH        28
#define SIGIO           29
#define SIGPOLL         SIGIO
/*
#define SIGLOST         29
*/
#define SIGPWR          30
#define SIGSYS          31
#define SIGUNUSED       31

/* These should not be considered constants from userland.  */
#define SIGRTMIN        32
```

주요 시그널

시그널 이름	의미	기본 처리
SIGABRT	abort()에서 발생되는 종료 시그널	종료(코어 덤프)
SIGALRM	자명종 시계 alarm() 울림 때 발생하는 알람 시그널	종료
SIGCHLD	프로세스의 종료 혹은 중지를 부모에게 알리는 시그널	무시
SIGCONT	중지된 프로세스를 계속시키는 시그널	무시
SIGFPE	0으로 나누기와 같은 심각한 산술 오류	종료(코어 덤프)
SIGHUP	연결 끊김	종료
SIGILL	잘못된 하드웨어 명령어 수행	종료(코어 덤프)
SIGIO	비동기화 I/O 이벤트 알림	종료
SIGINT	터미널에서 Ctrl-C 할 때 발생하는 인터럽트 시그널	종료
SIGKILL	잡을 수 없는 프로세스 종료시키는 시그널	종료
SIGPIPE	파이프에 쓰려는데 리더가 없을 때	종료
SIGPIPE	끊어진 파이프	종료

주요 시그널

SIGPWR	전원고장	종료
SIGSEGV	유효하지 않은 메모리 참조	종료(코어 덤프)
SIGSTOP	프로세스 중지 시그널	중지
SIGSTP	터미널에서 Ctrl-Z 할 때 발생하는 중지 시그널	중지
SIGSYS	유효하지 않은 시스템 호출	종료(코어 덤프)
SIGTERM	잡을 수 있는 프로세스 종료 시그널	종료
SIGTTIN	후면 프로세스가 제어 터미널을 읽기	중지
SIGTTOU	후면 프로세스가 제어 터미널에 쓰기	중지
SIGUSR1	사용자 정의 시그널	종료
SIGUSR2	사용자 정의 시그널	종료

시그널 생성

- 터미널에서 생성된 시그널
  - CTRL-C → SIGINT
  - CTRL-Z → SIGSTP
- 하드웨어 예외가 생성하는 시그널
  - 0으로 나누기 → SIGFPE
  - 유효하지 않는 메모리 참조 → SIGSEGV
- `kill()` 시스템 호출
  - 프로세스(그룹)에 시그널 보내는 시스템 호출
  - 프로세스의 소유자이거나 슈퍼유저이어야 한다.
- 소프트웨어 조건
  - SIGALRM: 알람 시계 울림
  - SIGPIPE: 끊어진 파이프
  - SIGCHLD: 자식 프로세스가 끝났을 때 부모에 전달되는 시그널

시그널 처리

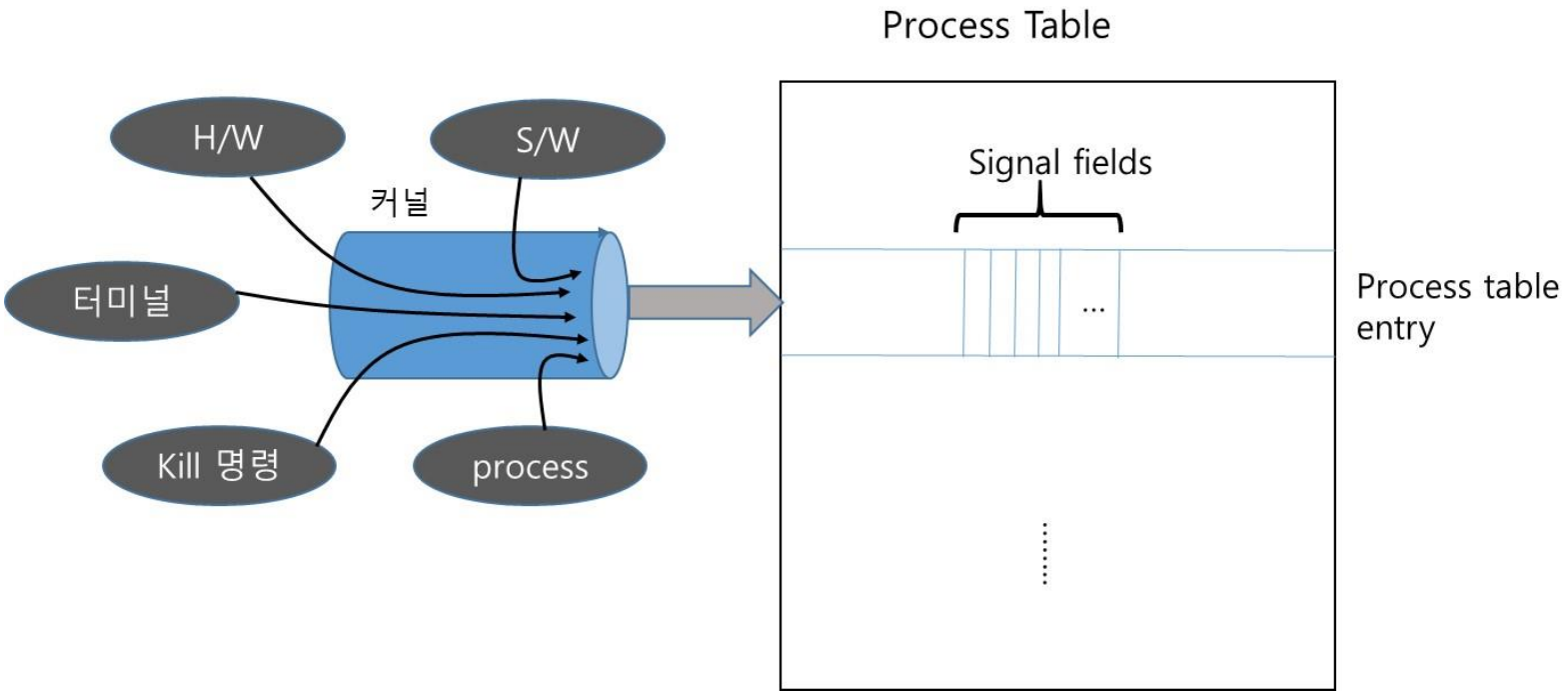
- 기본 처리 동작
  - 프로세스를 종료시킨다(terminate)
  - 시그널 무시(ignore)
  - 프로세스 중지(suspend)
  - 프로세스 계속(resume)

Signal handling

- Once we get a signal, we can do one of several things:
  - Ignore it. (note: there are some signals which we CANNOT or SHOULD NOT ignore)
    - (단, SIGSTOP과 SIGKILL은 무시할 수 없음)
  - Catch it. That is, have the kernel call a function which we define whenever the signal occurs.
  - Accept the default. Have the kernel do whatever is defined as the default action for this signal

signal을 보내는 (등록시키는) 방법

- signal을 프로세스에 보내는 것은, 결국 process table entry의 시그널 필드에 bit setting을 의미



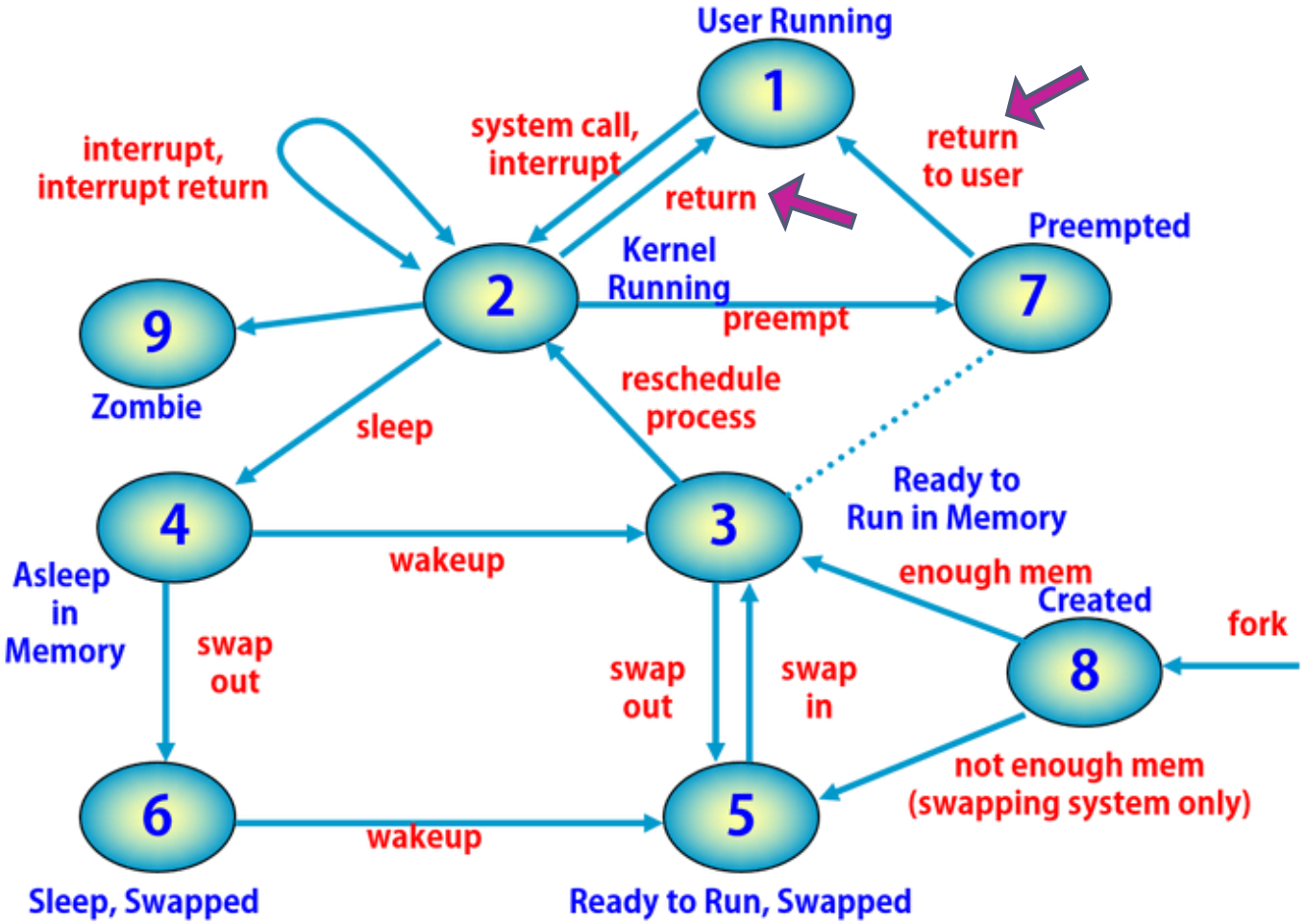
signal의 수신여부 및 처리시기

- signal 처리 시기
  - 프로세스가 커널 모드에서 사용자 모드로 리턴될 때.

- 시그널은 커널 모드에서 수행중인 프로세스에게 즉시적인 효과를 발휘하지 않음 (∵ user mode로 돌아갈 때만 처리하므로)

Process State Transition Disgram

https://codes.pratikkataria.com/process-and-process-management/

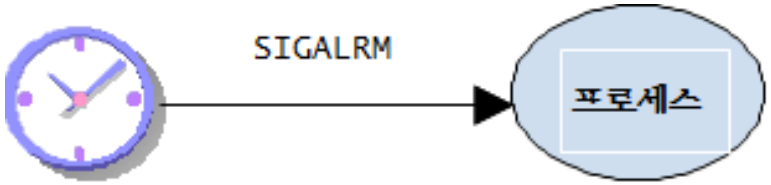


Check for and Handle Signals

alarm()

```
#include <unistd.h>
unsigned int alarm(unsigned int sec)
sec초 후에 프로세스에 SIGALRM 시그널이 발생되도록 설정한다.
```

- sec초 후에 프로세스에 SIGALRM 시그널이 발생한다.
  - 이 시그널을 받으면 "자명종 시계" 메시지를 출력하고 프로그램은 종료된다.



- 한 프로세스 당 오직 하나의 알람만 설정할 수 있다.
  - 이전에 설정된 알람이 있으면 취소되고 남은 시간(초)을 반환한다.
  - 이전에 설정된 알람이 없다면 0을 반환한다.
- alarm(0)
  - 이전에 설정된 알람은 취소된다.

alarm.c

```
#include <stdio.h>
/* 알람 시그널을 보여주는 프로그램 */
int main( )
{
    alarm(5);
    printf("무한 루프 ₩n");
    while (1) {
        sleep(1);
        printf("1초 경과 ₩n");
    }
    printf("실행되지 않음 ₩n");
}
```

```
[hansung@localhost chap12]$ ./a.out
무한 루프
1초 경과
1초 경과
1초 경과
1초 경과
Alarm clock
[hansung@localhost chap12]$
```

11.2 시그널 처리

시그널 처리기

- 시그널에 대한 처리함수 지정
  - signal() 시스템 호출
  - "이 시그널이 발생하면 이렇게 처리하라"
- signal() 시스템 호출

```
#include <signal.h>
signal(int signo, void (*func)( ))
```

signo에 대한 처리 함수를 func으로 지정하고 기존의 처리함수를 리턴한다

  - 시그널 처리 함수 func
    - SIG_IGN : 시그널 무시(단, SIGSTOP과 SIGKILL은 무시할 수 없음)
    - SIG_DFL : 기본 처리(대부분의 시그널에 대해서 프로세스는 종료)
    - 사용자 정의 함수 이름

예제: almhandler.c

```
#include <stdio.h>
#include <signal.h>
void alarmHandler();
/* 알람 시그널을 처리한다. */
int main( )
{
   signal(SIGALRM,alarmHandler);
   alarm(5);     /* 알람 시간 설정 */
   printf("무한 루프 ₩n");
   while (1) {
      sleep(1);
      printf("1초 경과 ₩n");
   }
   printf("실행되지 않음 ₩n");
}
```

```
void alarmHandler()
{
   printf("일어나세요₩n");
   exit(0);
}
```

```
[hansung@localhost ~]$ ./a.out
무한 루프
1초 경과
2초 경과
3초 경과
4초 경과
일어나세요
[hansung@localhost ~]$
```

예제: almhandler2.c

```
#include <stdio.h>
#include <signal.h>
void alarmHandler();
/* 알람 시그널을 처리한다. */
int main( )
{
   signal(SIGALRM,alarmHandler);
   alarm(5);     /* 알람 시간 설정 */
   printf("무한 루프 ₩n");
   while (1) {
      sleep(1);
      printf("1초 경과 ₩n");
   }
   printf("실행되지 않음 ₩n");
}
```

```
void alarmHandler()
{
    printf("일어나세요₩n");
  // exit(0);
}
```

```
[hansung@localhost ~]$ ./a.out
무한 루프
1초 경과
2초 경과
3초 경과
4초 경과
일어나세요
5초 경과
6초 경과
7초 경과
8초 경과
^C
[hansung@localhost ~]$
```

Int_handler.c

```
#include <signal.h>
#include <stdio.h>
void int_handler();
int num=0;

int main()
{
        signal(SIGINT, int_handler);
        while(1)
        {
                printf("I'm sleepy..₩n");
                sleep(1);
                if(num >=3)
                        exit(0);
        }
        return 0;
}

void int_handler()
{
    printf("int_handle called %d times₩n", ++num);
}
```

```
$ ./a.out
i'm sleepy..
int_handle called 1 times
i'm sleepy..
i'm sleepy..
int_handle called 2 times
i'm sleepy..
int_handle called 3 times
$
```

Int_handler2.c

```
#include <signal.h>
#include <stdio.h>
void int_handler();
int num=0;

int main()
{
        signal(SIGINT, int_handler);
        while(1)
        {
                printf("I'm sleepy..₩n");
                sleep(1);
                if(num >=2)
                        signal(SIGINT, SIG_DFL);

        }
        return 0;
}

void int_handler()
{
    printf("int_handle called %d times₩n", ++num);
}
```

```
$ ./a.out
i'm sleepy..
int_handle called 1 times
i'm sleepy..
i'm sleepy..
int_handle called 2 times
i'm sleepy..
$
```

pause

- **시그널이 전달될 때까지 대기한다.**

#include <unistd.h> int pause(void);	
반환값	항상 -1을 반환한다.

pause를 실행하면 프로세스는 임의의 시그널이 수신할 때까지 대기 상태가 된다.

- **아무 시그널이나 상관없다.**
- **수신된 시그널이 프로세스를 종료시키는 것이라면 프로세스는 pause 상태에서 벗어나자마자 종료된다.**
- **무시하도록 설정된 시그널에 대해서는 반응하지 않는다.**
- **시그널 핸들러가 등록된 시그널이라면 시그널 핸들러를 실행하고 나서 pause 상태를 벗어난다.**

예제: inthandler.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
void intHandler();
/* 인터럽트 시그널을 처리한다. */
Int main( )
{
   signal(SIGINT, intHandler);
   while (1)
      pause();
   printf("실행되지 않음 ₩n");
}
```

```
void intHandler(int signo)
{
   printf("인터럽트 시그널 처리₩n");
   exit(0);    // 생략하면???
}
```

```
[hansung@localhost ~]$ ./a.out
^C인터럽트 시그널 처리
[hansung@localhost ~]$
```

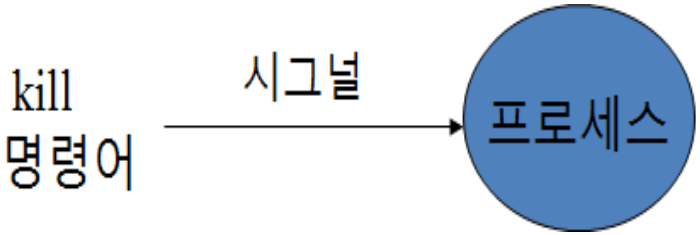
```
$ ex10-09
                                        ← Ctrl+C를 입력한다.
SIGINT cought

pause return -1
$
```

```
#include <unistd.h>
#include <signal.h>

void handler();

main()
{
        signal(SIGINT, handler);

        printf("pause return %d₩n", pause());
}

void handler()
{
        printf("₩nSIGINT coughtt₩n₩n");
}
```

11.3 시그널 보내기

시그널 보내기: kill 명령어

- killl 명령어
  - 한 프로세스가 다른 프로세스를 제어하기 위해 특정 프로세스에 임의의 시그널을 강제적으로 보낸다.



- $ kill [-시그널] 프로세스ID

- $ kill –l // 시그널 리스트

  HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH POLL PWR SYS ...

시그널 보내기: kill()

- kill() 시스템 호출
  - 특정 프로세스 pid에 원하는 임의의 시그널 signo를 보낸다.
  - 보내는 프로세스의 소유자가 프로세스 pid의 소유자와 같거나 혹은 보내는 프로세스의 소유자가 슈퍼유저이어야 한다.

```
#include <sys/types.h>
#include <signal.h>
int kill(int pid, int signo);
```

프로세스 pid에 시그널 signo를 보낸다. 성공하면 0 실패하면 -1를 리턴한다.

kill()

- *pid* > 0 :
  - signal to the process whose process ID is pid
- *pid* == 0 :
  - signal to the processes whose process group ID equals that of sender
- *pid* < 0 :
  - signal to the processes whose process group ID equals abs. of pid
- *pid* == -1 :
  - POSIX.1 leaves this condition unspecified (used as a broadcast signal in SVR4, 4.3+BSD)

```c
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>

main()
{
    pid_t pid;
    int count = 5;
    if((pid = fork()) > 0)
    {
        sleep(2);
        kill(pid, SIGINT);
        kill(getpid(), SIGINT);
        printf("[parent] bye!₩n");
    }
    else if(pid == 0)
    {
        while(count)
        {
            printf("[childe] count is %d₩n", count--);
            sleep(1);
        }
    }
    else
        printf("fail to fork₩n");
}
```

```
$ ex10-06
[childe] count is 5
[childe] count is 4

$
```

예제: 제한 시간 명령어 실행

- tlimit.c 프로그램
  - 명령줄 인수로 받은 명령어를 제한 시간 내에 실행
  - execute3.c 프로그램을 알람 시그널을 이용하여 확장

- 프로그램 설명
  - 자식 프로세스가 명령어를 실행하는 동안 정해진 시간이 초과되면 SIGALRM 시그널이 발생
  - SIGALRM 시그널에 대한 처리함수 alarmHandler()에서 자식 프로세스를 강제 종료
  - kill(pid, SIGINT) 호출을 통해 자식 프로세스에 SIGINT 시그널을 보내어 강제 종료
  - 만약 SIGALRM 시그널이 발생하기 전에 자식 프로세스가 종료하면 이 프로그램은 정상적으로 끝남

tlimit.c

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
int pid;
void alarmHandler();
/* 명령줄 인수로 받은 명령어 실행에
   제한 시간을 둔다. */
int main(int argc, char *argv[])
{
   int child, status, limit;
   signal(SIGALRM, alarmHandler);
   sscanf(argv[1], "%d", &limit);
   alarm(limit);
   pid = fork( );
   if (pid == 0) {
      execvp(argv[2], &argv[2]);
      fprintf(stderr, "%s:실행 불가₩n", argv[1]);
   } else {
      child = wait(&status);
      printf("[%d] 자식 프로세스 %d 종료 ₩n",
            getpid(), pid);
   }
}
void alarmHandler()
{
   printf("[알람] 자식 프로세스 %d 시간 초과
   ₩n", pid);
   kill(pid,SIGINT);
}
```

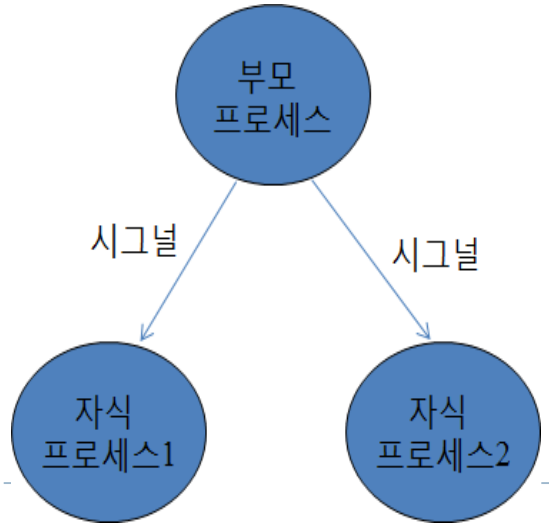
```
[hansung@localhost chap12]$ ./a.out 3  cat
[알람] 자식 프로세스 2452 시간 초과
[2451] 자식 프로세스 2452 종료
        종료 코드 0
[hansung@localhost chap12]$
```

시그널을 이용한 프로세스 제어

- 시그널을 이용하여 다른 프로세스를 제어할 수 있다.

SIGCONT	프로세스 재개
SIGSTOP	프로세스 중지
SIGKILL	프로세스 종료
SIGTSTP	Ctrl-Z에서 발생
SIGCHLD	자식 프로세스 중지 혹은 종료 시 부모 프로세스에 전달

- 예제: 시그널을 이용한 자식 프로세스 제어



control.c

```
#include <signal.h>
#include <stdio.h>
/* 시그널을 이용하여 자식 프로세스
   들을 제어한다. */
int main( )
{
   int pid1, pid2;

   pid1 = fork( );
   if (pid1 == 0) {
      while (1) {
         sleep(1);
         printf("프로세스 [1] 실행₩n");
      }
   }

   pid2 = fork( );
   if (pid2 == 0) {
      while (1) {
         sleep(1);
         printf("프로세스 [2] 실행₩n");
      }
   }
   sleep(2);
   kill(pid1, SIGSTOP);
   sleep(2);
   kill(pid1, SIGCONT);
   sleep(2);
   kill(pid2, SIGSTOP);
   sleep(2);
   kill(pid2, SIGCONT);
   sleep(2);
   kill(pid1, SIGKILL);
   kill(pid2, SIGKILL);
}
```

```
[hansung@localhost chap12]$ ./a.out
프로세스 [2] 실행
프로세스 [1] 실행
프로세스 [2] 실행
프로세스 [2] 실행
프로세스 [2] 실행
프로세스 [2] 실행
프로세스 [1] 실행
프로세스 [2] 실행
프로세스 [1] 실행
프로세스 [2] 실행
프로세스 [1] 실행
프로세스 [1] 실행
프로세스 [1] 실행
프로세스 [1] 실행
프로세스 [2] 실행
프로세스 [1] 실행
프로세스 [2] 실행
프로세스 [1] 실행
[hansung@localhost chap12]$
```

alarm과 pause로 mysleep 구현하기

// mysleep함수와 alrmhandler는 각자 구현해볼 것!

```
#include <signal.h>
#include <stdio.h>
void mysleep(int);
void alrmhandler();

int main()
{
        int time = 10;
        printf("Before sleep\n");
        mysleep(time);
        printf("After sleep\n");
}
```