Operating Systems LAB 8

Wonpyo Kim
skykwp@gmail.com

Outline

- Substance
  - **SIGNAL**

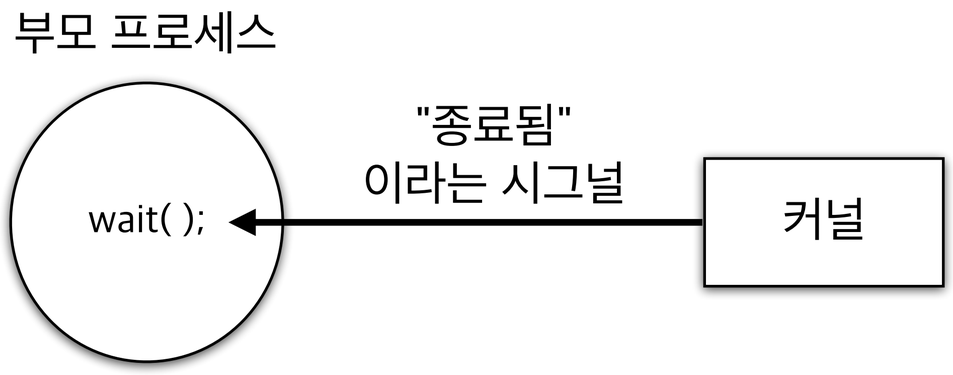
Signal

- 대부분의 프로그램들은 <Ctrl+C> 키(SIGINT 시그널)를 누르면 종료되는데, 쉘은 <Ctrl+C> 키를 눌러도 종료되지 않는다. 시그널에 대한 이해가 있으면 이러한 동작을 쉽게 구현할 수 있다.

- 프로그램의 수행 중 종료되지 않거나 무한루프 등에 빠지게 되면 <Ctrl+C> 키로 강제종료를 한다. 이는 사용자가 <Ctrl+C> 키를 누르면 커널이 종료 신호를 프로그램에 보내 종료되는 것이다. 이러한 신호가 시그널이다.

- 프로그램을 백그라운드로 실행시킨 후 종료되지 않아 강제로 종료를 원할 때는 kill 명령을 사용한다. kill 명령어 역시 시그널을 이용하여 프로그램을 종료시키는 것이다.

- fork( ) 에서 부모 프로세스는 자식 프로세스가 종료되면 깨어나는데, 이 때 시그널이 사용된다. 자식 프로세스가 종료되면 커널이 종료되었다는 내용의 시그널을 부모 프로세스에게 전달함으로써 wait( ) 상태에서 깨어나게 된다.



Signal

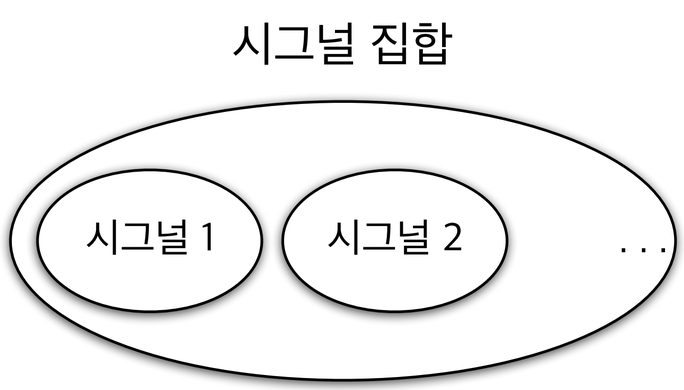
- 시그널의 종류

시그널 이름	의미
SIGABRT	abort 함수를 호출하면 보내지며, 이 시그널을 받으면 코어 덤프하고 종료한다.
SIGALRM	alarm 함수를 호출하면 보내지며, 이 시그널을 받으면 종료한다.
SIGBUS	하드웨어 결함이 탐지되면 보내지며, 이 시그널을 받으면 종료한다.
SIGCHILD	자식 프로세스가 종료되면 부모 프로세스에 보내지며, wait 에서 이 시그널에 의해 깨어난다. 이 시그널을 받으면 무시한다.
SIGCONT	중단되어 있는 프로세스가 이 시그널을 받으면 실행을 하고 실행중인 프로세스가 받으면 무시한다.
SIGFPE	0으로 나누기, 부동소수점 오류 등이 발생했을 때 보내지며, 이 시그널을 받으면 코어 덤프하고 종료한다.
SIGHUP	터미널 연결이 끊어졌을 때, 이 터미널과 연결된 세션 리더 또는 세션에 속한 모든 프로세스들에게 보내지며, 이 시그널을 받으면 종료한다.
SIGILL	불법 명령어를 실행할 때 보내지며, 이 시그널을 받으면 코어 덤프하고 종료한다.
SIGINT	터미널에서 인터럽트 키(일반적으로 <Ctrl+C>)를 눌렀을 때 보내지며, 이 시그널을 받으면 코어 덤프하고 종료한다.
SIGKILL	프로세스를 종료시키기 위해 보내지며, 이 시그널을 받으면 반드시 종료한다.
SIGPIPE	닫힌 파이프에 쓰고자 할 때 보내지며, 이 시그널을 받으면 종료한다.
SIGQUIT	터미널에서 종료 키(일반적으로 <Ctrl+\>)를 눌렀을 때 보내지며, 이 시그널을 받으면 코어 덤프하고 종료한다.
SIGSEGV	잘못된 메모리 주소를 접근하고자 할 때 보내지며, 이 시그널을 받으면 코어 덤프하고 종료한다.
SIGSTOP	프로세스를 멈추기 위해 보내지며, 이 시그널을 받으면 반드시 멈춘다.
SIGSYS	잘못된 시스템 호출을 했을 때 보내지며, 이 시그널을 받으면 코어 덤프하고 종료한다.
SIGTERM	프로세스를 종료시키기 전에 하던 일을 정리하고 종료할 것을 알릴 때 보내진다.
SIGTSTP	터미널에서 일시중지 키(일반적으로 <Ctrl+Z>)를 눌렀을 때 보내지며, 이 시그널을 받으면 멈춘다.
SIGTTIN	백그라운드 작업중인 프로세스가 표준 입력을 하려 할 때 현재 실행중인 프로세스에 보내지며, 이 시그널을 받으면 멈춘다.
SIGTTOU	백그라운드 작업중인 프로세스가 표준 출력을 하려 할 때 현재 실행중인 프로세스에 보내지며, 이 시그널을 받으면 멈춘다.
SIGURS1 SIGURS2	사용자가 정의해서 사용할 수 있는 시그널로, 이 시그널을 받으면 종료한다.

Signal

- 시그널과 관련된 함수 중 많은 함수들은 시그널 각각을 인수로 하지 않고 시그널 집합을 인수로 하는데, 이는 시그널 각각을 인수로 하면 동일한 작업을 수십 번 반복해야 하는 불편함이 있기 때문이다.

- 시그널 집합(signal set)은 말 그대로 시그널을 원소로 하는 집합을 의미한다.



- 시그널 집합의 데이터형은 sigset_t 으로 <signal.h>에 정의되어 있다.

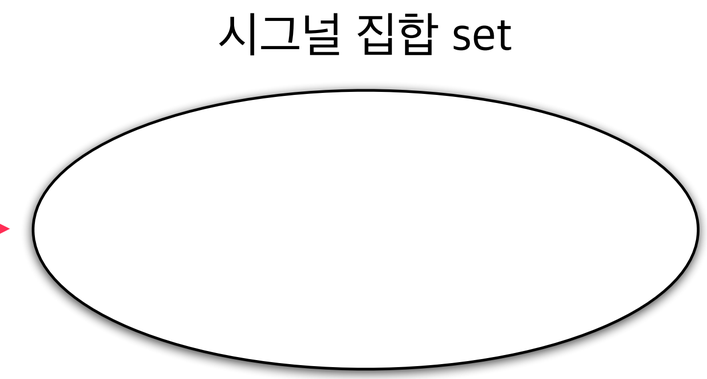
Signal

- 이러한 시그널 집합과 직접적으로 관련된 함수들은 다음과 같다.

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
```

- sigemptyset( )은 원소가 하나도 없는 빈 시그널 집합을 생성하는 함수이다.

sigemptyset 함수
기능
　　빈 시그널 집합을 생성한다.
기본형
　　int sigemptyset(sigset_t *set);
　　set: 생성하고자 하는 시그널 집합
반환값
　　성공: 0
　　실패: -1
헤더파일
　　<signal.h>



Signal

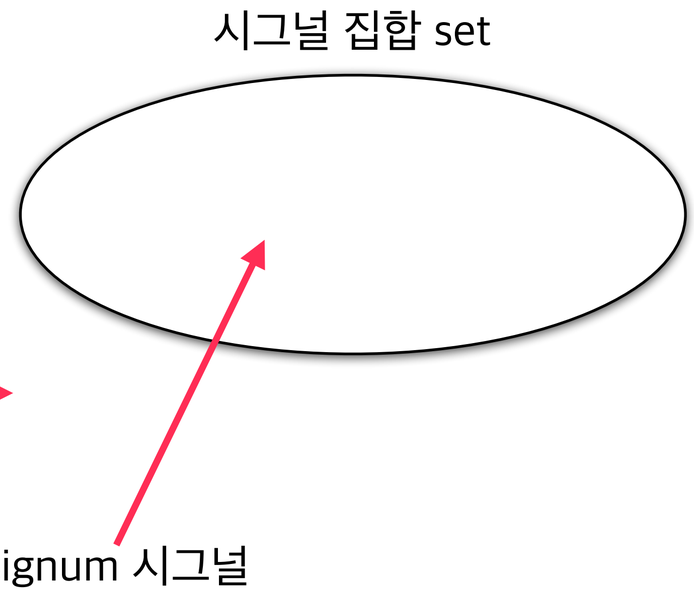
- sigfillset( ) 함수는 **모든 시그널**을 원소로 하는 시그널 집합을 생성하는 함수이다.

```
sigfillset 함수
기능
        모든 시그널을 포함한 시그널 집합을 생성한다.
기본형
        int sigfillset(sigset_t *set);
        set: 생성하고자 하는 시그널 집합
반환값
        성공: 0
        실패: -1
헤더파일
        <signal.h>
```

Signal

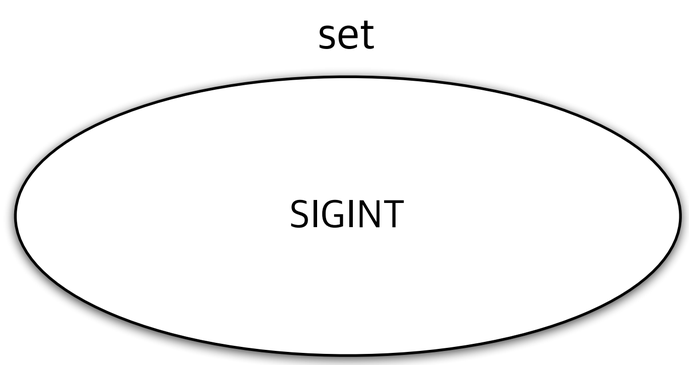
- sigaddset( ) 함수는 이미 존재하는 시그널 집합에 시그널을 추가하는 함수이다.

sigaddset 함수
기능
　　시그널을 시그널 집합에 추가한다.
기본형
　　int sigaddset(sigset_t *set, int signum);
　　set: 시그널 집합
　　signum: 추가하고자 하는 시그널 번호
반환값
　　성공: 0
　　실패: -1
헤더파일
　　<signal.h>



- 다음과 같이 실행하면 sigemptyset( ) 에 의해 빈 시그널 집합 set이 생성되고 sigaddset( ) 에 의해 시그널 집합 set 에 SIGINT 시그널이 추가된다.

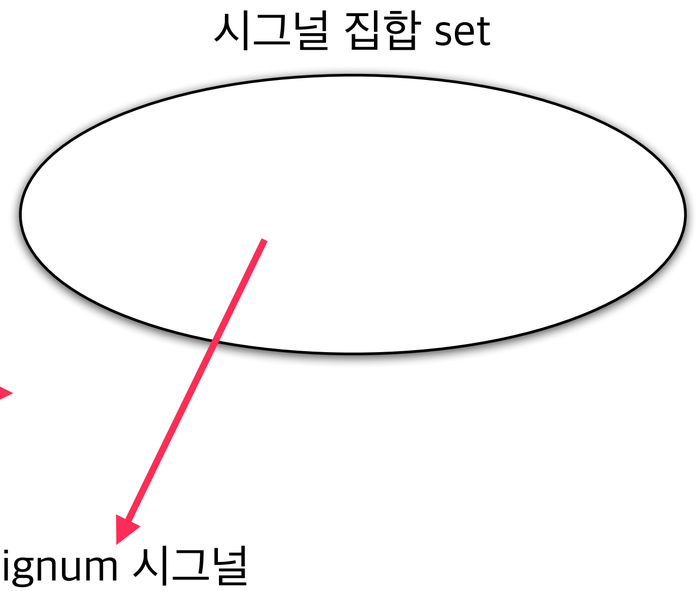
```
sigemptyset(&set);
sigaddset(&set, SIGINT);
```



Signal

- sigdelset( ) 함수는 이미 존재하는 시그널 집합에서 시그널을 제거하는 함수이다.

sigdelset 함수
기능
　　시그널 집합에 시그널을 삭제한다.
기본형
　　int sigdelset(sigset_t *set, int signum);
　　set: 시그널 집합
　　signum: 삭제하고자 하는 시그널 번호
반환값
　　성공: 0
　　실패: -1
헤더파일
　　<signal.h>



Signal

- sigismember( ) 함수는 signum 시그널이 시그널 집합 set 에 속해있는지를 확인하는 함수이다.

```
sigismember 함수
기능
        시그널이 시그널 집합에 속하는지를 확인한다.
기본형
        int sigismember(const sigset_t *set, int signum);
        set: 시그널 집합
        signum: 확인하고자 하는 시그널 번호
반환값
        성공:
            속하면: 1
            속하지 않으면: 0
        실패: -1
헤더파일
        <signal.h>
```

Signal

- 예제

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

main()
{
    sigset_t set;

    if (sigemptyset(&set) == -1) {
        perror("sigemptyset failed");
        exit(1);
    }

    if (sigaddset(&set, SIGINT)  == -1) {
        perror("sigaddset failed");
        exit(1);
    }

    if (sigismember(&set, SIGINT)) {
        printf("SIGINT is a member\n");
    }

    if (!sigismember(&set, SIGPIPE)) {
        printf("SIGPIPE is not a member\n");
    }

    exit(0);
}
```

- 본 예제는 시그널 집합을 생성하고 SIGINT 시그널을 추가한 후 SIGINT, SIGPIPE 시그널 집합에 속해있는지 확인하는 예제이다.

- SIGPIPE는 속해있지 않으므로 if 조건에 **만족하여** 출력된다.

Signal

- 각 프로세스는 시그널과 관련해서 크게 다음의 세 가지 일을 한다.

- 첫째, 시그널이 도착했을 때 그에 대한 응답을 한다. 응답 방법에는 다음과 같은 세 가지 방법이 존재한다.
  - 시스템에서 기본적으로 설정한 동작을 한다.
  - 시그널을 무시한다. 단, SIGKILL과 SIGSTOP 시그널은 무시할 수 없다.
  - 특정 루틴(함수)을 실행한다.

- 둘째, 시그널이 도착했을 때 시그널을 블록화한다. 중요한 부분을 실행하고 있을 때 시그널이 도착하면 이를 블록화해 뒤로 미룰 수가 있는데, 블록을 해제하면 블록화되었던 시그널이 전달된다.

- 셋째, 프로세스에 시그널을 보낸다. kill( ), raise( ) 와 같은 함수를 이용하여 다른 또는 자기 자신의 프로세스에 시그널을 보낼 수 있다. 관련 함수는 다음과 같다.

```
void (*signal(int signum, void (*sighandler)(int)))(int);
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

Signal

- 예제

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
    while (!0) {
        printf("Hello world\n");
        sleep(1);
    }
}
```

- 본 예제는 "Hello world"를 무한히 반복 출력한다.

- 종료를 위해서는 실행 중인 화면에서 <Ctrl+C>(인터럽트 키)를 눌러 SIGINT 시그널을 보내 프로세스를 종료시킨다.

Signal

- signal( ) 함수는 시그널을 받았을 때, 종료되지 않고 무시하거나 특정 함수가 실행되도록 할 수 있도록 해준다.
- 함수를 호출한 후에 signum 시그널을 받게되면 sighandler 에 설정한 일을 하게 된다.

signal 함수
기능
　　시그널 처리를 설정한다.
기본형
　　void (*signal(int signum, void (*sighandler)(int)))(int);
　　signum: 시그널 번호
　　sighandler: 설정할 시그널 핸들러
반환값
　　성공: 이전의 시그널 핸들러 포인터
　　실패: -1 (SIG_ERR)
헤더파일
　　<signal.h>

- 두 번째 인수인 sighandler 는 인수로 동작할 함수 이름만이 아니라 SIG_IGN 또는 SIG_DFL 이 될 수 있다.

시그널 이름	의미
함수 이름	시그널을 받으면 "함수 이름" 함수가 실행된다.
SIG_IGN	시그널을 받으면 무시한다.
SIG_DFL	시그널을 받으면 시스템에서 기본적으로 설정한 동작을 한다.

Signal

- 예제

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

main()
{
    int count = 0;

    signal(SIGINT, SIG_IGN);

    while (!0) {
        printf("Hello world\n");
        sleep(1);

        if (++count == 5) {
            signal(SIGINT, SIG_DFL);
        }
    }
}
```

- 본 예제는 SIGINT 시그널을 처음에는 무시했다가 5번의 카운트가 되었을 때, 시스템에서 기본으로 하는 동작으로 다시 복귀한다.

- 따라서, 처음 5번의 출력 동안에 <Ctrl+C>는 반응이 없다가 5번 후에 프로그램이 <Ctrl+C>를 인지하여 종료한다.

Signal

- 예제

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

void signal_handler(int signo);

main()
{
    signal(SIGINT, signal_handler);

    while(!0) {
        printf("Hello world\n");
        sleep(1);
    }
}

void signal_handler(int signo)
{
    printf("I'm your signal %d\n", signo);
    signal(SIGINT, SIG_DFL);
}
```

- 본 예제는 앞선 예제의 응용으로 signal_handler 라는 함수를 임의로 생성하고 SIGINT 에 등록시켰다.

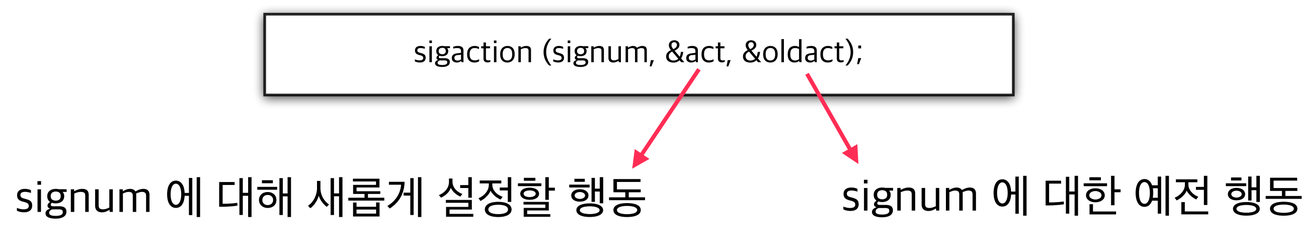
- signal_handler 가 수행됨에 따라 내부에서 SIGINT를 원래대로 돌려놓기 때문에, 2번째 <Ctrl+C> 부터는 원래 시스템 시그널의 기능을 하게 된다.

Signal

- sigaction( ) 함수는 signal( ) 함수에 비해 좀 더 향상된 기능을 제공한다.

sigaction 함수
기능
　　시그널 처리를 설정한다.
기본형
　　int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
　　signum: 시그널 번호
　　act: 설정할 행동
　　oldset: 이전 행동
반환값
　　성공: 0
　　실패: -1
헤더파일
　　<signal.h>

- signum 시그널에 대해 act 행동을 설정하고 oldact 에는 이 함수가 호출되기 전의 행동 정보가 저장된다.



Signal

- act 와 oldact 의 데이터형인 struct sigaction 은 /usr/include/x86_64-linux-gnu/bits/sigaction.h 에 선언되어있다.

```
/* Structure describing the action to be taken when a signal arrives.  */
struct sigaction
  {
    /* Signal handler.  */
#if defined __USE_POSIX199309 || defined __USE_XOPEN_EXTENDED
    union
      {
        /* Used if SA_SIGINFO is not set.  */
        __sighandler_t sa_handler;
        /* Used if SA_SIGINFO is set.  */
        void (*sa_sigaction) (int, siginfo_t *, void *);
      }
    __sigaction_handler;
# define sa_handler     __sigaction_handler.sa_handler
# define sa_sigaction   __sigaction_handler.sa_sigaction
#else
    __sighandler_t sa_handler;
#endif

    /* Additional set of signals to be blocked.  */
    __sigset_t sa_mask;

    /* Special flags.  */
    int sa_flags;

    /* Restore handler.  */
    void (*sa_restorer) (void);
  };
```

- sa_handler 는 시그널을 받았을 때 취할 행동을 설정하는 것으로, 함수이름, SIG_IGN, SIG_DFL이 올 수 있다.

- sa_sigaction 은 sa_flags 가 SA_SIGINFO 일 때, sa_handler 대신 동작하는 핸들러로 sa_handler 보다 여러 정보를 전달받아 동작한다.

- sa_mask 는 시그널을 처리하는 동안 블록화할 시그널 집합으로 시그널을 처리하는 동안 sa_mask 에 의해 지정된 시그널이 도착하면 이를 블록화하고 시그널 처리가 끝나면 블록이 해제되어 시그널을 받는다.

Signal

- sa_flags 는 시그널 처리에 관련된 다음과 같은 옵션을 지정하는데, OR 연산(|)을 통해 여러 옵션을 동시에 지정할 수도 있다.

옵션	의미
SA_NOCLDSTOP	signum 이 SIGCHILD 인 경우, 자식 프로세스가 멈추었을 때 부모 프로세스로 SIGCHILD 가 전달되지 않는다.
SA_ONESHOT 또는 SA_RESETHAND	시그널을 받으면 설정된 행동을 취하고, 시스템 기본 설정인 SIG_DFL 로 재설정한다.
SA_RESTART	시그널 처리에 의해 방해받은 시스템 호출은 시그널 처리가 끝나면 재시작한다.
SA_NOMASK 또는 SA_NODEFER	시그널을 처리하는 동안에 전달되는 시그널은 블록화되지 않는다.
SA_SIGINFO	이 옵션을 설정하지 않으면 시그널 번호만을 인수로 하는 sa_handler 가 동작하며, 설정하면 세 개를 인수로 하는 sa_sigaction 이 동작한다. 세 개의 인수는 시그널 번호, 시그널이 만들어진 이유, 시그널을 받는 프로세스의 정보이다.

Signal

- 예제

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

main()
{
    struct sigaction act;

    act.sa_handler = SIG_IGN;

    sigemptyset(&act.sa_mask);

    act.sa_flags = 0;

    sigaction(SIGINT, &act, NULL);

    while (!0) {
        printf("Hello world\n");
        sleep(1);
    }
}
```

- 본 예제는 act.sa_handler 를 SIG_IGN 로 설정하므로 시그널을 무시한다.

- 그리고 act.sa_mask 시그널 집합을 비워서 시그널을 처리하는 동안 발생한 모든 시그널은 블록화시키지 않는다.

- sigaction( ) 함수를 통해 SIGINT 시그널을 받아도 무시하도록 설정한다. 즉, SIGINT <Ctl+C> 로는 프로세스를 종료할 수 없다.

Signal

- 예제

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

void signal_handler(int signo);
int signal_count = 0;

main()
{
    struct sigaction new_act;
    struct sigaction old_act;

    new_act.sa_handler = signal_handler;

    sigemptyset(&new_act.sa_mask);

    new_act.sa_flags = 0;

    sigaction(SIGINT, &new_act, &old_act);

    while (!0) {
        printf("waiting signal\n");
        sleep(1);
        if (signal_count == 3) {
            sigaction(SIGINT, &old_act, NULL);
        }
    }
}

void signal_handler(int signo)
{
    signal_count++;
    printf("I'm your signal %d\n", signal_count);
}
```

- 본 예제는 sigaction( ) 함수의 세 번째 인수를 활용하여 호출 이전의 상태로 되돌리는 것이다.
- old_act 에 SIGINT 가 저장된다. 그러므로 signal_count 가 3이되면 SIGINT 시그널에 대해 old_act 가 설정되므로 다시 원래대로 <Ctrl+C> 로 프로세스를 종료할 수 있다.

Signal

- 다른 프로세스에 시그널을 보낼 수 있다. signal( ) 과 sigaction( ) 은 자신의 프로세스에게 전달되는 시그널을 어떻게 처리할지를 결정하는 함수이다.

- 이와는 대조적으로 내가 다른 프로세스에 시그널을 보내려면, kill( ) 함수를 사용한다. 프로세스ID가 pid 인 프로세스에 sig 변 시그널을 보냄을 의미한다.

kill 함수
기능
　　프로세스에 시그널을 보낸다.
기본형
　　int kill(pid_t pid, int sig);
　　pid: 시그널을 받을 프로세스의 프로세스 ID
　　sig: 보내고자 하는 시그널 번호
반환값
　　성공: 0
　　실패: -1
헤더파일
　　<sys/types.h>
　　<signal.h>

Signal

- pause( ) 함수는 호출한 프로세스를 임의의 시그널이 도착할 때까지 실행을 중단시킨다.

```
pause 함수
기능
        시그널이 도착할 때까지 실행을 중단시킨다.
기본형
        int pause(void);
반환값
        항상 -1
헤더파일
        <unistd.h>
```

Signal

- 예제

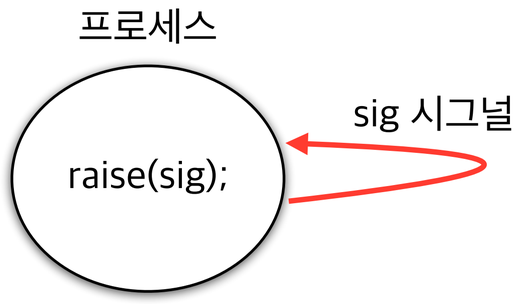
```c
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>

static int signal_flag = 0;
void signal_handler(int signo);

main()
{
    struct sigaction act;

    if (fork()) {
        printf("parent process\n");

        act.sa_handler = signal_handler;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        sigaction(SIGINT, &act, NULL);

        pause();

        if (signal_flag) {
            printf("signal fired\n");
        }

        exit(0);
    } else {
        sleep(5);
        kill(getppid(), SIGINT);

        exit(0);
    }
}

void signal_handler(int signo)
{
    signal_flag = 1;
}
```

- 본 예제의 부모 프로세스는 SIGINT 시그널에 대해 signal_handler( )가 실행되고 pause( )에 의해 실행을 중단한다.
- 자식 프로세스는 5초 동안 가만히 있다가 kill( ) 함수를 이용해 부모 프로세스에 SIGINT 시그널을 보내고 종료한다.
- 부모 프로세스는 SIGINT에 의해 signal_handler( )를 실행시키고 종료한다.

Signal

- raise( ) 와 alarm( ) 은 자신에게 시그널을 보내는 함수이다. raise( ) 함수는 다음과 같이 sig번 시그널을 자기 자신에게 보낸다.

raise 함수
기능
　　자기 자신에게 시그널을 보낸다.
기본형
　　int raise(int sig);
　　sig: 보내고자 하는 시그널 번호
반환값
　　성공: 0
　　실패: 0 이외
헤더파일
　　<signal.h>



Signal

- 예제

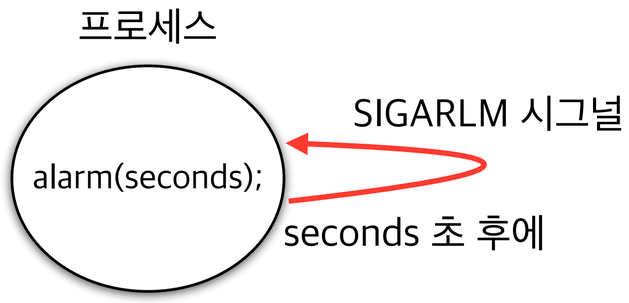
```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

main()
{
    int count = 0;

    while (!0) {
        printf("Hello world\n");
        count++;
        if (count == 3) {
            raise(SIGINT);
        }
        sleep(2);
    }
}
```

- 본 예제는 아주 간단한 예제로 2초씩 sleep( ) 하면서 "Hello world"를 출력하다가 count가 3이 되면 자기 자신에게 raise( ) 함수를 통해 SIGINT 시그널을 보내 종료한다.

Signal

- alarm( ) 함수는 seconds 초 후에 SIGALRM 시그널을 자기 자신에게 보낸다.

alarm 함수
기능
　　자기 자신에게 SIGALARM 시그널을 보낸다.
기본형
　　unsigned int alarm(unsigned int seconds);
　　seconds: 시그널을 보낼 시간, 단위는 초
반환값
　　이전에 호출한 alarm이 없으면: 0
　　이전에 호출한 alarm이 있으면: 남은 시간
헤더파일
　　<unistd.h>



Signal

- 예제

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

void alarm_handler(int signo);

main()
{
    int status;
    struct sigaction act;

    act.sa_handler = alarm_handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    sigaction(SIGALRM, &act, NULL);

    alarm(3);

    while (!0);

}

void alarm_handler(int signo)
{
    printf("I'm your signal %d\n", signo);
    exit(0);
}
```

- SIGALRM 시그널에 의해 alarm_handler( ) 가 실행되고 alarm( )에 의해 자신에게 시그널을 보낸다.

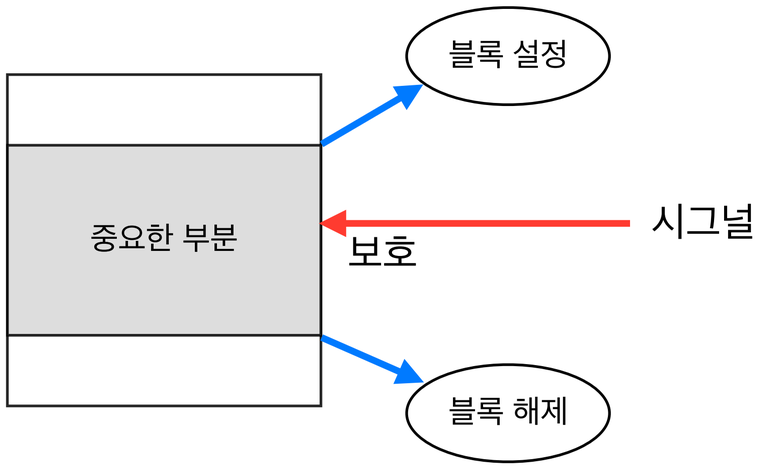
Signal

- 중요한 작업을 할 경우, 시그널로부터 보호받는 것이 바람직하다. 이 경우에는 시그널 블록을 설정하면 된다. 시그널 블록이 설정된 상태에서는 시그널이 올 경우 시그널 블록이 해제될 때 까지 처리를 미루게 된다.

sigprocmask 함수
기능
	블록화될 시그널을 설정한다.
기본형
	int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
	how: 취할 동작
	set: 설정할 시그널 집합
	oldset: 이전에 블럭화된 시그널 집합
반환값
	성공: 0
	실패: -1
헤더파일
	<signal.h>

Signal

- 블록화를 도식화하면 다음과 같다.



- sigprocmask( ) 함수의 how 인수의 값과 의미는 다음과 같다.

how	의미
SIG_BLOCK	기존에 블록화된 시그널 집합에 set의 시그널이 추가된다.
SIG_UNBLOCK	기존에 블록화된 시그널 집합에서 set의 시그널이 제외된다.
SIG_SETMASK	set의 시그널이 블록화된 시그널 집합으로 교체된다.

Signal

- 예제

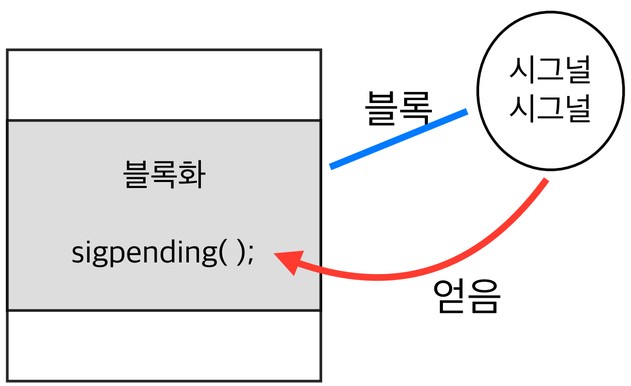
```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

main()
{
    sigset_t set1, set2;

    sigfillset(&set1);
    sigemptyset(&set2);

    sigaddset(&set2, SIGINT);
    sigprocmask(SIG_BLOCK, &set1, NULL);

    printf("block start\n");
    sleep(10);

    sigprocmask(SIG_UNBLOCK, &set2, NULL);

    printf("SIGINT unblock\n");
    while (!0) {
        printf("hello world\n");
        sleep(2);
    }
}
```

- 본 예제는 전체 시그널에 대해 블록화를 설정하여 실행하다가, SIGINT 를 통해서 블록을 해제하는 프로그램이다.

Signal

- 블록화된 시그널을 얻어오는 방법은 sigpending( ) 함수를 사용하는 것이다.

sigpending 함수
기능
　　블록화된 시그널을 얻어온다.
기본형
　　int sigpending(sigset_t *set);
　　set: 블록화된 시그널을 저장할 시그널 집합
반환값
　　성공: 0
　　실패: -1
헤더파일
　　<signal.h>



Signal

- sigsuspend( ) 함수는 mask 의 시그널이 블록화된 시그널 집합으로 교체됨과 동시에 mask 에 포함되지 않은 시그널이 도착할 때까지 실행이 중단된다.

sigsuspend 함수
기능
	시그널 블록을 설정함과 동시에 시그널이 도착할 때까지 중단한다.
기본형
	int sigsuspend(const sigset_t *mask);
	mask: 블록화될 시그널 집합
반환값
	성공: 0
	실패: -1
헤더파일
	<signal.h>