시그널
Signal

- IPC
- 시그널의 종류
- 시그널 구현 함수

IPC

❑ 프로세스 간 통신 (Inter-Process Communication)
- 실행 중인 프로세스 간에 데이터를 주고 받는 기법

❑ IPC에는 매우 많은 방법이 있다!
- File
- Signal
- Pipe/Named pipe
- Socket
- Shared memory
- Message passing
- Remote Procedure Call (RPC)

시그널

- 특정 이벤트가 발생했을 때 프로세스에게 전달하는 "신호"
  - 연산 오류 발생, 자식 프로세스의 종료, 사용자의 종료 요청 등
  - 굉장히 작은 값이다.
  - 인터럽트(interrupt)라고 부르기도 한다.
  - 용도가 제한적이며 여러 시그널이 겹칠 경우 원치 않는 결과가 발생할 위험이 있다.
- 시그널은 여러 종류가 있고 각각에 유일한 번호가 붙여져 있다.
  - 프로그램 내에서는 매크로 상수를 사용한다.
    예1) Ctrl + C를 누를 때 SIGTERM이 전달된다.
    예2) kill 명령을 사용하면 해당 프로세스에게 SIGTERM이 전달된다.

시그널

- 시그널을 수신한 프로세스의 반응

1. 시그널에 대해 기본적인 방법으로 대응한다. 대부분의 시그널에 대해서 프로세스는 종료하게 된다.
2. 시그널을 무시한다. 단, SIGKILL과 SIGSTOP은 무시될 수 없다.
3. 프로그래머가 지정한 함수(핸들러)를 호출한다.

- 시그널 핸들러 (handler)
  - 프로세스가 특정 시그널을 포착했을 때 수행해야 할 별도의 함수
    - 프로세스는 시그널을 포착하면 현재 작업을 일시 중단하고 시그널 핸들러를 실행
    - 시그널 핸들러의 실행이 끝나면 중단된 작업을 재개
- 시그널 종류
  - "/usr/include/asm/signal.h"

시그널

시그널	설명	시그널	설명
SIGABRT	process aborted	SIGALRM	signal raised by “alarm”
SIGBUS	bus error	SIGCHLD	child process terminated
SIGCONT	continue if stopped	SIGFPE	floating point exception
SIGHUP	hangup	SIGILL	illegal instruction
SIGINT	interrupt	SIGKILL	kill
SIGPIPE	write to pipe	SIGQUIT	quit and dump core
SIGSEGV	segmentation violation	SIGSTOP	stop temporarily
SIGTERM	termination	SIGTSTP	terminal stop signal
SIGTTIN/ SIGTTOU	background process attempting to read/write	SIGUSR1/ SIGUSR2	user defined 1/2
SIGPOLL	pollable event	SIGPROF	profiling timer expired
SIGSYS	bad syscall	SIGTRAP	trace/breakpoint trap
SIGURG	urgent data available	SIGVTALRM	virtual timer expired
SIGXCPU	CPU time limit exceeded	SIGXFSZ	file size limit exceeded

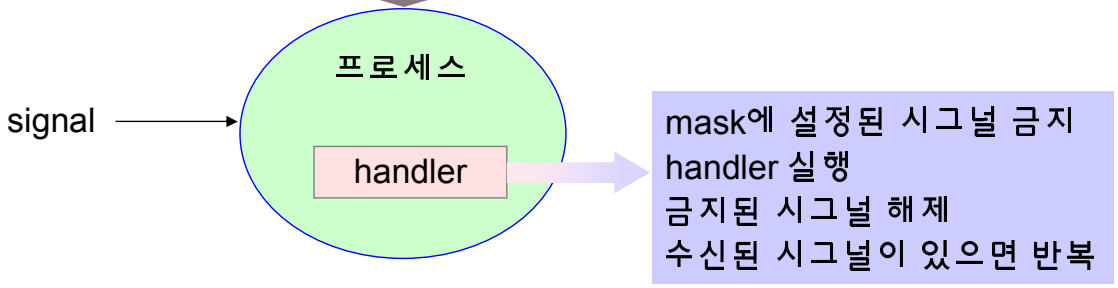
시그널 구현 함수

- 시그널을 다루기 위해 필요한 시스템 호출/표준 라이브러리 함수

함수	의미
sigemptyset	시그널 집합을 시그널이 없는 비어 있는 상태로 초기화
sigfillset	시그널 집합을 모든 시그널이 포함된 상태로 초기화
sigaddset	시그널 집합에 특정 시그널을 추가
sigdelset	시그널 집합에서 특정 시그널을 삭제
sigaction	특정 시그널에 대한 프로세스의 행동을 설정
sigprocmask	봉쇄할 시그널의 목록을 변경
kill	특정 프로세스에게 특정 시그널을 전달
raise	자기 자신에게 특정 시그널을 전달
alarm	설정된 시간이 경과한 후에 자기 자신에게 시그널을 전달
pause	시그널이 도착할 때까지 대기 상태 유지

시그널 처리 과정

프로그램에서 처리할 시그널 선정 (sigfillset, sigaddset, …)
시그널을 처리할 동안 방해받지 않도록 mask 설정 (sigprocmask)
각 시그널에 대한 처리 행동 지정 (handler 함수 구현)



시그널 집합 조작

❑ 시그널 집합을 생성하거나 조작

- 시그널을 다루려면 시그널 집합을 만들어야 한다.

```
#include <signal.h>

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);
```

set	sigset_t형의 시그널 집합
signum	시그널 번호
반환값	호출이 성공하면 0을 반환, 실패하면 -1을 반환한다. sigismember는 호출이 성공하면 1이나 0을 반환하고 실패하면 -1을 반환한다.

시그널 집합 조작

❑ 사용 예

```
sigset_t set;
int result;

sigemptyset(&set);
sigfillset(&set);
sigdelset(&set, SIGALRM);
sigaddset(&set, SIGALRM);
result = sigismember(set, SIGALRM);
```


sigaction

❑ 특정한 시그널을 받았을 때 프로세스가 취해야 할 행동을 지정

- signum으로 지정한 시그널에 대해서 act로 지정한 행동을 취한다.
- 새로운 행동인 act가 등록되면서 기존의 행동은 oldact에 저장된다.

```
#include <signal.h>

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

signum	시그널 번호이다. SIGKILL과 SIGSTOP은 적용할 수 없다.
act	프로세스가 지정한 시그널에 대해서 취해야 할 행동에 관한 정보가 담겨져 있다.
oldact	이전에 지정되어 있는 시그널에 대한 행동이 저장된다. 보통 NULL 값을 사용한다.
반환값	호출이 성공할 경우 0을 반환하고, 실패하면 -1을 반환한다.

sigaction

struct sigaction

- 시그널 핸들러, signum으로 지정한 특정 시그널이 들어왔을 때 수행해야 할 행동을 저장하는 구조체

```
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
}
```

- sa_handler: 시그널에 대응하는 행동

SIG_DFL	기본적으로 설정된 행동. 대부분의 시그널에 대해서 프로세스는 종료.
SIG_IGN	해당 시그널을 무시. 지정한 시그널에 대한 행동 없음. (단, SIGSTOP과 SIGKILL은 무시할 수 없음)
함수의 포인터	사용자가 지정한 함수

sigaction

sa_sigaction

- sa_handler 대신에 사용할 수 있다.
- sa_handler에 비해 추가 정보를 알 수 있다.
  - sa_sigaction과 sa_handler 중에 하나만 사용한다.
- sa_sigaction을 사용하려면 sa_flags를 SA_SIGINFO로 지정한다.

sa_mask

- 시그널 마스크 : 봉쇄된 시그널들의 집합
- sa_mask에 등록된 시그널은 시그널 핸들러가 실행되는 동안 봉쇄된다.
  - 봉쇄 (blocking)
    - 무시가 아니라 시그널 핸들러 실행이 완료될 때까지 처리가 미뤄진다.
    - 현재 처리 중인 시그널도 봉쇄된다.

sigaction

❑ sa_flags

- 시그널 처리 절차를 수정하는데 사용된다.
- 여러 플래그를 사용하고 싶으면 '|'(bitwise-OR) 사용

값	의미
SA_SIGINFO	sa_handler 대신에 sa_sigaction을 선택
SA_NOCLDSTOP	signum이 SIGCHLD일 때 자식 프로세스가 종료되거나 중단되더라도 부모 프로세스는 이를 알려고 하지 않는다.
SA_RESETHAND	signum에 대해서 시그널 핸들러를 최초에 한번만 실행하고 그 다음부터는 동일한 시그널에 대해서 SIG_DFL에 해당하는 기본적인 동작만 수행하게 된다.
SA_NODEFER	시그널 핸들러 내에서 시그널 받는 것을 금지하지 않는다. 즉, 마스크를 사용하지 않는다.

sigaction

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

int num=0;

int main (void)
{
    static struct sigaction act;

    void int_handle(int);

    act.sa_handler = int_handle;
    sigfillset(&act.sa_mask);
    sigaction(SIGINT, &act, NULL);

    while(1) {
        printf("I'm sleepy...₩n");
        sleep(1);
        if (num >= 2) {
            act.sa_handler = SIG_DFL;
            sigaction(SIGINT, &act, NULL);
        }
    }
}

void int_handle(int signum)
{
    printf("SIGINT:%d₩n", signum);
    printf("int_handle called %d times₩n", ++num);
}
```

실행 결과:
프로그램 실행 도중 ^C로 시그널을 보낸다.

```
juyoon@ce:~/system/programming/si
[juyoon:signal/68]$ ex04
I'm sleepy...
I'm sleepy...
SIGINT:2
int_handle called 1 times
I'm sleepy...
I'm sleepy...
SIGINT:2
int_handle called 2 times
I'm sleepy...
I'm sleepy...

[juyoon:signal/69]$
```

sigprocmask

❑ 시그널 집합 단위로 봉쇄될 시그널 목록 설정

- 프로세스가 대단히 중요한 코드를 실행 중일 때 작업에 방해를 받지 않기 위해 시그널을 무시할 수도 있다.

#include <signal.h> int sigprocmask (int *how*, const sigset_t **set*, sigset_t **oldset*);	
how	sigprocmask 함수의 행동 방식을 결정
set	새롭게 적용하는 시그널 마스크
oldset	이전에 적용되어 있는 시그널 마스크를 저장
반환값	호출이 성공할 경우 0을 반환하고, 실패하면 -1을 반환한다.

시그널 봉쇄 → 중요한 작업 수행 → 시그널 봉쇄 해제

sigprocmask

❑ how

값	의미
SIG_BLOCK	현재 봉쇄 설정된 시그널의 목록에 두 번째 인자 set에 포함된 시그널을 추가한다.
SIG_UNBLOCK	현재 봉쇄 설정된 시그널의 목록에서 두 번째 인자 set에 포함된 시그널을 제외한다.
SIG_SETMASK	현재 봉쇄 설정된 시그널의 목록을 두 번째 인자 set가 가진 목록으로 대체한다.

sigprocmask

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

int main (void)
{
    sigset_t set;
    int count = 5;

    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigprocmask(SIG_BLOCK, &set, NULL);

    while (count) {
        printf("Don't disturb me (%d)₩n", count--);
        sleep(1);
    }
    sigprocmask(SIG_UNBLOCK, &set, NULL);
    printf("You did not disturb me!₩n");
    return 0;
}
```

실행 결과:
프로그램 실행 도중 ^C로 시그널을 보낸다.

```
juyoon@ce:~/system/programming/sig
[juyoon:signal/73]$ ex05
Don't disturb me (5)
Don't disturb me (4)
Don't disturb me (3)
Don't disturb me (2)
Don't disturb me (1)

[juyoon:signal/74]$
```


kill/raise

❑ 프로세스에 시그널 전달

- kill: 특정 프로세스나 프로세스 그룹에게 지정한 시그널을 전달
- raise: 자기 자신에게 지정한 시그널을 전달한다.

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

#include <signal.h>
int raise (int sig);
```

pid	프로세스의 식별 번호
sig	시그널 번호
반환값	호출이 성공하면 0을 반환, 실패하면 -1을 반환

kill/raise

❑ kill에서 pid 값에 따른 의미

pid	의미
pid > 0	특정 프로세스에 시그널 전달
pid = 0	자신과 같은 그룹에 있는 모든 프로세스에 시그널 전달
pid = -1	시그널 전달이 허용된 모든 프로세스에 시그널 전달. 단, 프로세스 식별 번호가 1인 프로세스(init)는 제외.
pid < -1	프로세스 그룹 식별 번호가 -pid인 모든 프로세스에게 시그널 전달

❑ raise는 kill의 특정한 형태

```
raise(sig);  ⇔  kill (getpid(), sig);
```


alarm

❑ 지정한 시간이 경과한 후 자신에게 SIGALRM 시그널을 보낸다.

```
#include <unistd.h>

unsigned int alarm (unsigned int seconds);
```

seconds	초 단위의 시간
반환값	지정한 시간 중 남은 시간을 반환한다. 0 이상의 값이다.

- alarm 설정은 한번에 하나만 등록할 수 있다.
  - 여러 개를 누적해서 등록할 수 없다.
  - 마지막에 등록한 하나의 alarm만 유효하다.
    - 이전에 등록된 alarm은 취소된다.
- 지금까지 사용했던 sleep 함수는 alarm을 사용하여 구현되었다.
  - sleep과 alarm은 함께 사용하지 않는 것이 좋다.

alarm

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

void timeover (int signum)
{
    printf("₩n₩ntime over!₩n₩n");
    exit(0);
}

int main (void)
{
    char buf[1024];
    char *alpha = "abcdefghijklmnopqrstuvwxyz";

    int timelimit;
    struct sigaction act;

    act.sa_handler=timeover;
    sigaction(SIGALRM, &act, NULL);
    printf("input time limit (sec): ₩n");
    scanf("%d", &timelimit);
    alarm(timelimit);
    printf("START₩n > ");
    scanf("%s", buf);

    if (!strcmp(buf, alpha))
        printf("₩ell done.. you succeeded!₩n");
    else printf("Sorry... you failed₩n");
    return 0;
}
```

```
juyoon@ce:~/system/programming/signal
[juyoon:signal/77]$ ex07
input time limit (sec):
5
START
 > abcdefghijklmnopqrs

time over!

[juyoon:signal/78]$ abcdefghijklmnopqrstu
```

실행 결과

pause

❑ 시그널이 전달될 때까지 대기

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

- 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.

pause

juyoon@ce:~/system/programming/signal

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

void handler(int signam);

int main (void)
{
    struct sigaction act;

    sigfillset(&act.sa_mask);
    act.sa_handler = handler;
    sigaction (SIGINT, &act, NULL);
    printf("pause return %d₩n", pause());
    return 0;
}

void handler (int signum)
{
    printf("₩nSIGINT caught!₩n");
}
```

실행 결과:
프로그램 실행 도중 ^C로 시그널을 보낸다.

juyoon@ce:~/system/programming/signal

```
[juyoon:signal/81]$ ex09

SIGINT caught!
pause return -1
[juyoon:signal/82]$
```


예 제

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

void handler(int signum);
int flag = 5;

int main (void)
{
    struct sigaction act;
    sigset_t set;

    sigemptyset(&(act.sa_mask));
    sigaddset(&(act.sa_mask), SIGALRM);
    sigaddset(&act.sa_mask, SIGINT);
    sigaddset(&act.sa_mask, SIGUSR1);
    act.sa_handler = handler;
    sigaction(SIGALRM, &act, NULL);
    sigaction(SIGINT, &act, NULL);
    sigaction(SIGUSR1, &act, NULL);

    printf("call raise(SIGUSR1) before blocking₩n");
    raise(SIGUSR1);

    sigemptyset(&set);
    sigaddset(&set, SIGUSR1);
    sigprocmask(SIG_SETMASK, &set, NULL);

    while (flag) {
        printf("input SIGINT [%d]₩n", flag);
        sleep(1);
    }
```

처리할 시그널 수

시그널 추가 및 핸들러 설정

SIGUSR1 제한

flag가 0이 될 때까지 1초 간격으로 출력

예 제

juyoon@ce:~/system/programming/signal

```
    printf("call kill(getpid(), SIGUSR1) after blocking₩n");
    kill(getpid(), SIGUSR1);

    printf("sleep by pause.. zzZz₩n");
    printf("pause return %d₩n", pause());
    printf("2 seconds sleeping ..zzZ₩n");
    alarm(2);
    pause();
}

void handler (int signum)
{
    flag--;

    switch(signum) {
        case SIGINT:
            printf("SIGINT(%d)₩n", signum);
            break;
        case SIGALRM:
            printf("SIGALRM(%d)₩n", signum);
            break;
        case SIGUSR1:
            printf("SIGUSR1(%d)₩n", signum);
            break;
        default:
            printf("signal(%d)₩n", signum);
    }
}
```

SIGUSR1을 자신에게 전달. 효과는?

각 시그널에 대해 번호를 출력