HALLYM UNIVERSITY

CIC Convergence Information & Communications

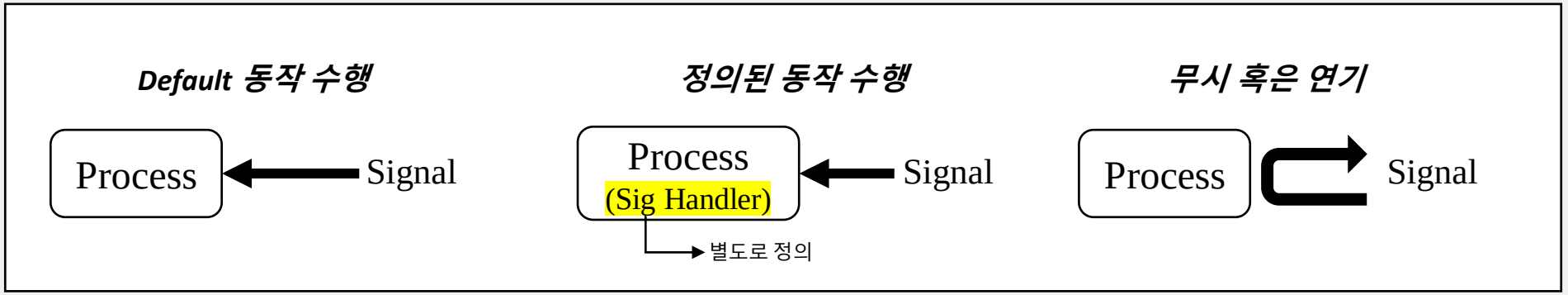
(Operating System) Practice -9-

Signal 2

Index

Signal Overview

- **Signal**
  - 비동기적 사건 (이벤트, 예외 등)의 발생을 프로세스에게 알리기 위해 사용
  - 다양한 종류의 Signal이 존재
    - ✓ 대부분의 Signal은 프로세스를 종료시키기 위해 사용됨
  - Signal을 수신한 프로세스는 다음 중 하나의 동작을 수행
    - ① 수신한 Signal의 Default 동작을 수행
    - ② 프로세스에 정의된 동작 수행
    - ③ Signal 무시 혹은 연기
  - 이전 실습 내용: Signal Handler, Pause, Kill, Raise & Alarm

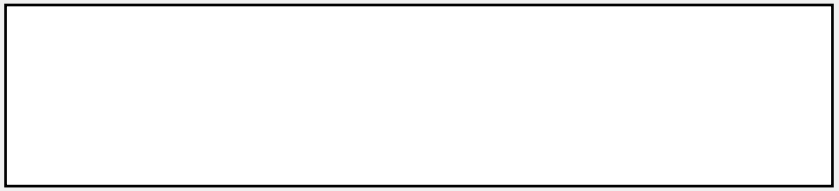


Signal Practice -5 (Signal Set)

- **Signal Set**
  - ✓ Signal은 보통 묶어서 set으로 다룬다. → Signal의 종류가 많고, 비슷한 의미의 signal이 많아서
    - Signal 각각을 처리하면, 동일한 동작을 반복함 → 비효율
  - ✓ sigemptyset → 빈 signal set을 생성하는 함수

유형	설명
헤더 파일	<signal.h>
형태	int sigemptyset(sigset_t *set);
인수	sigset_t *set : signal set 변수
반환 값	성공: 0 실패: -1

Signal set



Signal Practice -5 (Signal Set)

- **Signal Set**
  - ✓ sigfillset → 모든 signal을 포함한 signal set 생성

유형	설명
헤더 파일	<signal.h>
형태	int sigfillset(sigset_t *set);
인수	sigset_t *set : signal set 변수
반환 값	성공: 0 실패: -1

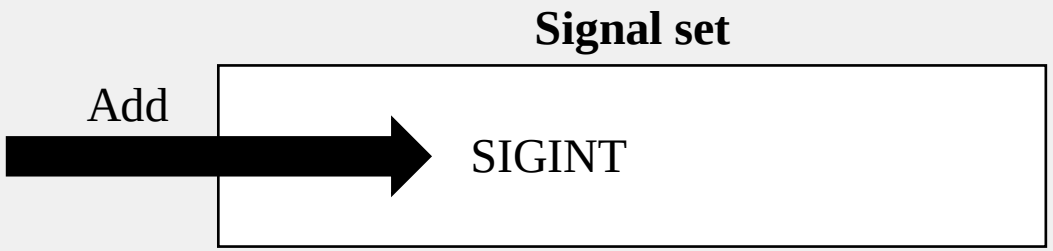
Signal set

SIGINT, SIGALRM, …… (*All signals*)

Signal Practice -5 (Signal Set)

- **Signal Set**
  - ✓ sigaddset → 특정 Signal을 Signal set에 추가

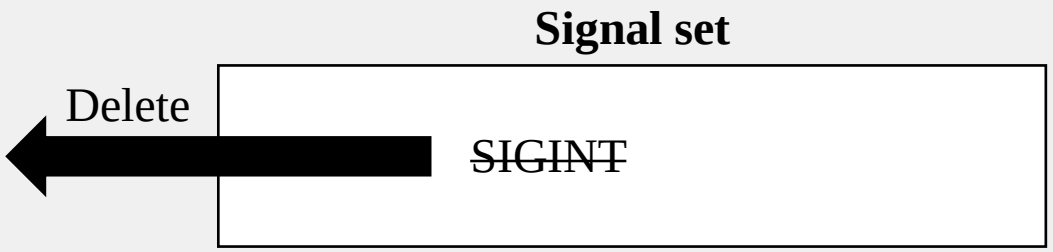
유형	설명
헤더 파일	<signal.h>
형태	int sigaddset(sigset_t *set, int signum);
인수	sigset_t *set : signal set 변수 int signum : 추가할 signal 번호
반환 값	성공: 0 실패: -1



Signal Practice -5 (Signal Set)

- **Signal Set**
  - ✓ sigdelset → 특정 Signal을 Signal set에서 삭제

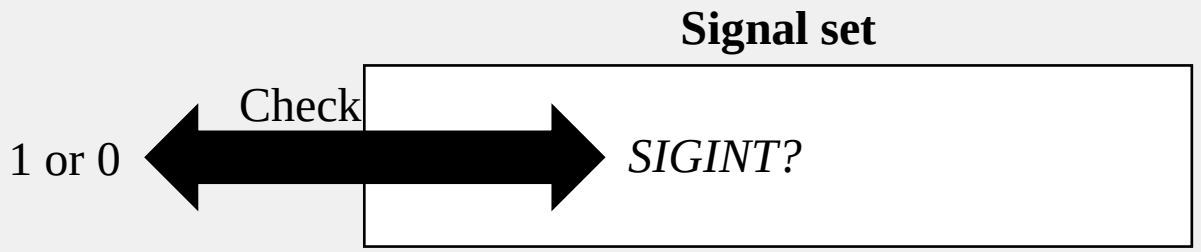
유형	설명
헤더 파일	<signal.h>
형태	int sigdelset(sigset_t *set, int signum);
인수	sigset_t *set : signal set 변수 int signum : 추가할 signal 번호
반환 값	성공: 0 실패: -1



Signal Practice -5 (Signal Set)

- **Signal Set**
  - ✓ sigismember → 특정 Signal이 Signal set에 포함되어 있는지 확인

유형	설명
헤더 파일	<signal.h>
형태	int sigismember(sigset_t *set, int signum);
인수	sigset_t *set : signal set 변수 int signum : 확인할 signal 번호
반환 값	signum이 집합의 멤버일 경우 : 1 signum이 집합의 멤버가 아닐 경우 : 0 에러: -1



Signal Practice -5 (Signal Set)

- **Signal Set Practice**

Sigset1.c

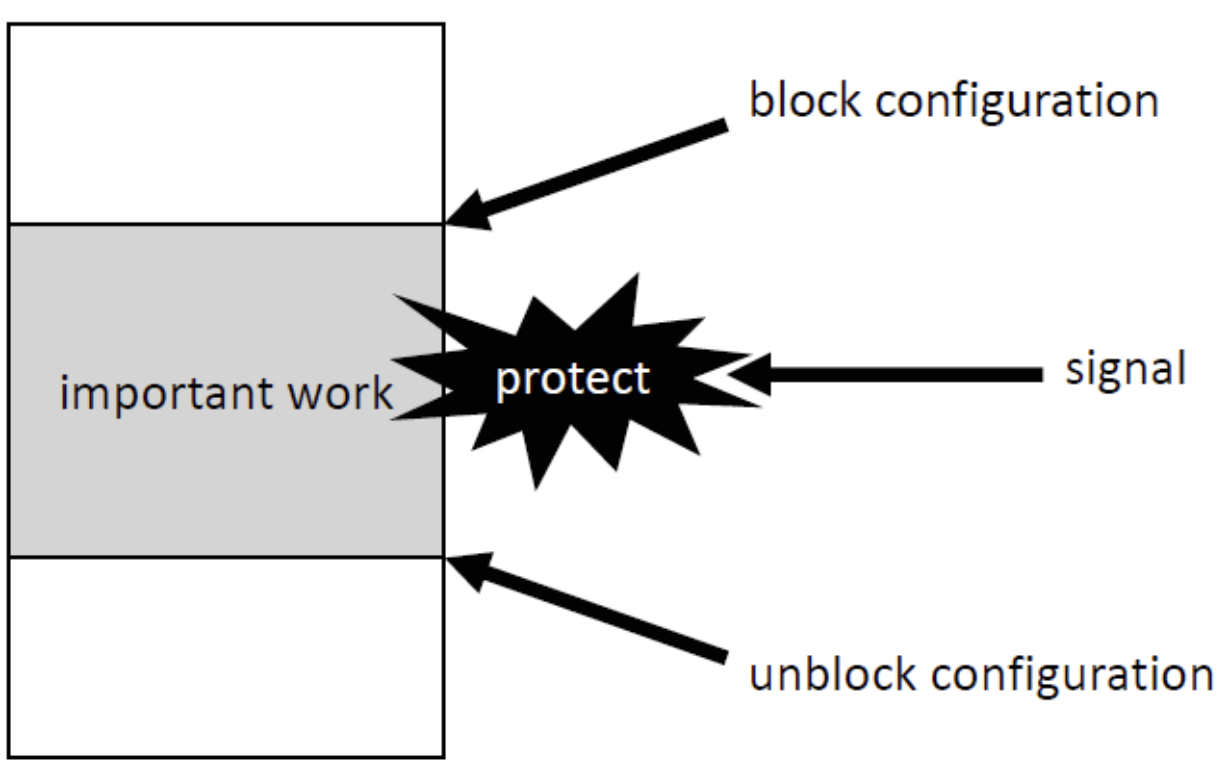
```c
#include <stdio.h>
#include <signal.h>

int main(int argc, char **argv){
    sigset_t set;

    if(sigemptyset(&set)==-1){
        perror("fail to run sigemptyset\n");
    }

    if(sigaddset(&set, SIGINT)==-1){
        perror("fail to run sigaddset\n");
    }

    if(sigismember(&set, SIGINT)){
        printf("SIGINT is a member.\n");
    }
    else{
        printf("SIGINT is not a member.\n");
    }

    if(sigismember(&set, SIGKILL)){
        printf("SIGKILL is a member.\n");
    }
    else{
        printf("SIGKILL is not a member.\n");
    }
}
```

Sigset2.c

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

int main(int argc, char **argv){
    sigset_t set;

    if(sigfillset(&set)==-1){
        perror("fail to run sigemptyset\n");
    }

    if(sigdelset(&set, SIGPIPE)==-1){
        perror("fail to run sigaddset\n");
    }

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

    if(sigismember(&set, SIGINT)){
        printf("SIGINT is a member.\n");
    }
    else{
        printf("SIGINT is not a member.\n");
    }
}
```

Signal Practice -6 (Signal Block)

- **Signal Block**
  - ✓ 프로세스가 중요한 작업을 할 때, Signal을 수신하더라도 강제 종료되지 않게 보호해야 함
  - ✓ 즉, 특정상황에서는 Signal로 인한 종료를 연기해야 됨.



Signal Practice -6 (Signal Block)

- **Signal Block**
  - ✓ sigprocmask → Block될 Signal을 설정

유형	설명
헤더 파일	<signal.h>
형태	int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
인수	int how : 취할 동작 const sigset_t *set : 설정할 signal set sigset_t *oldset : 이전에 블록화된 signal set
반환 값	성공 : 0 실패 : -1

How 인수의 값과 의미

how	설명
SIG_BLOCK	기존에 블록화된 signal set에 두번째 인수 signal set을 추가
SIG_UNBLOCK	기존에 블록화된 signal set에서 두번째 인수 signal set에 있는 signal 제거
SIG_SETMASK	이전에 블록화된 signal set을 모두 지우고 두번째 인수인 signal set으로 설정

Signal Practice -6 (Signal Block)

- **Signal Block Practice**

block.c

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

int main(){
    sigset_t set1, set2;
    sigfillset(&set1);
    sigemptyset(&set2);

    sigaddset(&set2, SIGINT);

    sigprocmask(SIG_BLOCK, &set1, NULL);
    printf("---------------------Block ALL Signals\n");
    sleep(2);
    printf("2s\n");
    sleep(2);
    printf("4s\n");
    printf("---------------------Unblock SIGINT\n");
    sigprocmask(SIG_UNBLOCK, &set2, NULL);
    sleep(2);
    printf("6s\n");
    sleep(2);
    printf("8s\n");
}
```

Signal Practice -7 (Sigaction)

- **Sigaction**
  - ✓ 일괄적이고 유연한 Signal 처리를 수행
  - ✓ 기존 Signal functions보다 더욱 향상된 기능 제공
  - ✓ sigaction → sigaction 구조체를 사용한 일괄적인 signal 처리

유형	설명
헤더 파일	<unistd.h>
형태	int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
인수	int signum : signal 번호 const struct sigaction *act : 설정할 action struct sigaction *oldact : 이전 action
반환 값	성공 : 0 실패 : -1

sigaction 구조체

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

1. **sa_handler(int)**
   - signal이 발생했을 때 실행될 함수
2. **sa_mask**
   - sa_handle가 실행되는 동안 블록되어야 하는 signals
3. **sa_ flags**
   - signal 처리 과정에서의 동작 옵션

Signal Practice -7 (Sigaction)

- **Sigaction**
  - ✓ sa_flags 옵션

옵션	설명
SA_NOCLDSTOP	signum이 SIGCHILD일 경우, 자식 프로세스가 멈추었을 때, 부모 프로세스에 SIGCHILD가 전달되지 않음
SA_ONESHOT 또는 SA_RESETHAND	signal을 받으면 설정된 action을 취하고 시스템 기본 설정인 SIG_DFL로 재설정됨
SA_RESTART	signal 처리에 의해 방해 받은 시스템 호출은 signal 처리가 끝나면 재시작함
SA_NOMASK 또는 SA_NODEFER	signal을 처리하는 동안에 전달되는 signal은 블록되지 않음
SA_SIGINFO	sa_handler 대신에 sa_sigaction이 동작되며, sa_handler보다 더 다양한 인수를 받을 수 있음 sa_sigaction이 받는 인수는 signal 번호, signal이 만들어진 이유, signal을 받는 프로세스의 정보임

Signal Practice -7 (Sigaction)

- **Sigaction**

sigaction.c

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

struct sigaction act_new;
struct sigaction act_old;

void sigint_handler(){
    printf("Ctrl + C pressed\n");
    printf("Next time, it makes the process end.\n");
    sigaction(SIGINT, &act_old, NULL);
}

void main(){
    act_new.sa_handler = sigint_handler;

    sigaction(SIGINT, &act_new, &act_old);
    while(1){
        printf("Running\n");
        sleep(2);
    }
}
```