

시그널 보내기[2]

□ 시그널 보내기: raise(2)

```
#include <signal.h>

int raise(int sig);
```

- 함수를 호출한 프로세스에 시그널 발송

□ 시그널 보내기: abort(3)

```
#include <stdlib.h>

void abort(void);
```

- 함수를 호출한 프로세스에 SIGABRT시그널 발송
- SIGABRT 시그널은 프로세스를 비정상적으로 종료시키고 코어덤프 생성



시그널 핸들러 함수[2]

□ 시그널 핸들러 지정: sigset(3)

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

```
void (*sigset(int sig, void (*disp)(int)))(int);
```

- **disp** : **sig**로 지정한 시그널을 받았을 때 처리할 방법
 - 시그널 핸들러 함수의 주소값
 - **SIG_IGN** : 시그널을 무시하도록 지정
 - **SIG_DFL** : 기본 처리 방법으로 처리하도록 지정
- **sigset**함수는 **signal**함수와 달리 시그널 핸들러가 한 번 호출된 후에 기본동작으로 재설정하지 않고, 시그널 핸들러를 자동으로 재정한다.



```
...
07 void handler(int signo) {
08     printf("Signal Handler Signal Number : %d\n", signo);
09     psignal(signo, "Received Signal");
10 }
11
12 int main(void) {
13     if (sigset(SIGINT, handler) == SIG_ERR) {
14         perror("sigset");
15         exit(1);
16     }
17
18     printf("Wait 1st Ctrl+C... : SIGINT\n");
19     pause();
20     printf("After 1st Signal Handler\n");
21     printf("Wait 2nd Ctrl+C... : SIGINT\n");
22     pause();
23     printf("After 2nd Signal Handler\n");
24
25     return 0;
26 }
```

시그널 핸들러를 재지정
하지 않아도 됨

```
# ex7_4.out
Wait 1st Ctrl+C... : SIGINT
^CSignal Handler Signal Number: 2
Received Signal: Interrupt
After 1st Signal Handler
Wait 2nd Ctrl+C... : SIGINT
^CSignal Handler Signal Number: 2
Received Signal: Interrupt
After 2nd Signal Handler
```

시그널 집합

□ 시그널 집합의 개념

- 시그널을 개별적으로 처리하지 않고 복수의 시그널을 처리하기 위해 도입한 개념
- POSIX에서 도입

□ 시그널 집합의 처리를 위한 구조체

- `sigset_t`

```
typedef struct {  
    unsigned int __sigbits[4];  
} sigset_t;
```

- 시그널을 비트 마스트로 표현. 각 비트가 특정 시그널과 1:1로 연결
- 비트값이 1이면 해당 시그널이 설정된 것이고, 0이면 시그널 설정 안된 것임



시그널 집합 처리 함수[1]

□ 시그널 집합 비우기 : sigemptyset(3)

```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

- 시그널 집합에서 모든 시그널을 0으로 설정

□ 시그널 집합에 모든 시그널 설정: sigfillset(3)

```
#include <signal.h>

int sigfillset(sigset_t *set);
```

- 시그널 집합에서 모든 시그널을 1로 설정

□ 시그널 집합에 시그널 설정 추가: sigaddset(3)

```
#include <signal.h>

int sigaddset(sigset_t *set, int signo);
```

- signo로 지정한 시그널을 시그널 집합에 추가



시그널 집합 처리 함수[2]

□ 시그널 집합에서 시그널 설정 삭제: sigdelset(3)

```
#include <signal.h>

int sigdelset(sigset_t *set, int signo);
```

- signo로 지정한 시그널을 시그널 집합에서 삭제

□ 시그널 집합에 설정된 시그널 확인: sigismember(3)

```
#include <signal.h>

int sigismember(sigset_t *set, int signo);
```

- signo로 지정한 시그널이 시그널 집합에 포함되어 있는지 확인



[예제 7-5] 시그널 집합 처리 함수 사용하기(test2.c)

ex7_5.c

```
01  #include <signal.h>
02  #include <stdio.h>
03
04  int main(void) {
05      sigset_t st;
06      sigemptyset(&st);
07      sigaddset(&st, SIGINT);
08      sigaddset(&st, SIGQUIT);
09
10      if (sigismember(&st, SIGINT))
11          printf("SIGINT is setting.\n");
12
13      printf("** Bit Pattern: %x\n", st.__sigbits[0]);
14
15      return 0;
16  }
```

시그널 집합 비우기

시그널 추가

시그널 설정 확인

6은 2진수로 00000110이므로 오른쪽에서 2
번, 3번 비트가 1로 설정
SIGINT는 2번, SIGQUIT는 3번 시그널

ex7_5.out
SIGINT is setting.
** Bit Pattern: 6

sigaction 함수의 활용[1]

□ sigaction 함수

- signal이나 sigset 함수처럼 시그널을 받았을 때 이를 처리하는 함수 지정
- signal, sigset 함수보다 다양하게 시그널 제어 가능

□ sigaction 구조체

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

- **sa_flags** : 시그널 전달 방법을 수정할 플래그(다음 쪽 참조)
- **sa_handler/sa_sigaction** : 시그널 처리를 위한 동작 지정
 - sa_flags에 SA_SIGINFO가 설정되어 있지 않으면 sa_handler에 시그널 처리동작 지정
 - sa_flags에 SA_SIGINFO가 설정되어 있으면 sa_sigaction 멤버 사용
- **sa_mask** : 시그널 핸들러가 수행되는 동안 블록될 시그널을 지정한 시그널 집합



sigaction 함수의 활용[2]

□ sa_flags에 지정할 수 있는 값(sys/signal.h)

플래그	설명
SA_ONSTACK (0x00000001)	이 값을 설정하고 시그널을 받으면 시그널을 처리할 프로세스에 sigaltstack 시스템 호출로 생성한 대체 시그널 스택이 있는 경우에만 대체 스택에서 시그널을 처리한다. 그렇지 않으면 시그널은 일반 스택에서 처리된다.
SA_RESETHAND (0x00000002)	이 값을 설정하고 시그널을 받으면 시그널의 기본 처리 방법은 SIG_DFL로 재설정되고 시그널이 처리되는 동안 시그널을 블록하지 않는다.
SA_NODEFER (0x00000010)	이 값을 설정하고 시그널을 받으면 시그널이 처리되는 동안 유닉스 커널에서 해당 시그널을 자동으로 블록하지 못한다.
SA_RESTART (0x00000004)	이 값을 설정하고 시그널을 받으면 시스템은 시그널 핸들러에 의해 중지된 기능을 재 시작하게 한다.
SA_SIGINFO (0x00000008)	이 값이 설정되지 않은 상태에서 시그널을 받으면 시그널 번호(sig 인자)만 시그널 핸들러로 전달된다. 만약 이 값을 설정하고 시그널을 받으면 시그널 번호 외에 추가 인자 두 개가 시그널 핸들러로 전달된다. 두 번째 인자가 NULL이 아니면 시그널이 발생한 이유가 저장된 siginfo_t 구조체를 가리킨다. 세 번째 인자는 시그널이 전달될 때 시그널을 받는 프로세스의 상태를 나타내는 ucontext_t 구조체를 가리킨다.
SA_NOCLDWAIT (0x00010000)	이 값이 설정되어 있고 시그널이 SIGCHLD면 시스템은 자식 프로세스가 종료될 때 좀비 프로세스를 만들지 않는다.
SA_NOCLDSTOP (0x00020000)	이 값이 설정되어 있고 시그널이 SIGCHLD면 자식 프로세스가 중지 또는 재시작할 때 부모 프로세스에 SIGCHLD 시그널을 전달하지 않는다.



sigaction 함수의 활용[3]

□ sigaction 함수

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

```
int sigaction(int sig, const struct sigaction *restrict act,  
              struct sigaction *restrict oact);
```

- **sig** : 처리할 시그널
- **act** : 시그널을 처리할 방법을 지정한 구조체 주소
- **oact** : 기존에 시그널을 처리하던 방법을 저장할 구조체 주소
- 첫번째 인자로 SIGKILL과 SIGSTOP을 제외한 어떤 시그널도 올 수 있음
- 성공하면 0, 실패시 -1



[예제 7-6] sigaction 함수 사용하기(1) test3.c

ex7_6.c

```
...
07 void handler(int signo) {
08     psignal(signo, "Received Signal:");
09     sleep(5);
10     printf("In Signal Handler, After Sleep\n");
11 }
12
13 int main(void) {
14     struct sigaction act;
15     sigemptyset(&act.sa_mask);
16     sigaddset(&act.sa_mask, SIGQUIT);
17     act.sa_flags = 0;
18     act.sa_handler = handler;
19     if (sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0) {
20         perror("sigaction");
21         exit(1);
22     }
23
24     fprintf(stderr, "Input SIGINT: ");
25     pause();
26     fprintf(stderr, "After Signal Handler\n");
27
28     return 0;
29 }
30 }
```

sa_mask 초기화

SIGQUIT 시그널을 블록시키기 위해 추가

시그널핸들러 지정

시그널 받기 위해 대기(pause함수)

ex7_6.out
Input SIGINT: ^CReceived Signal:: Interrupt
^\\In Signal Handler, After Sleep
끝(Quit)(코어덤프)

[예제 7-7] sigaction 함수 사용하기(SA_RESETHAND) test4.c

```
...
07 void handler(int signo) {
08     psignal(signo, "Received Signal:");
09     sleep(5);
10     printf("In Signal Handler, After Sleep\n");
11 }
12
13 int main(void) {
14     struct sigaction act;
15
16     sigemptyset(&act.sa_mask);
17     sigaddset(&act.sa_mask, SIGQUIT);
18     act.sa_flags = SA_RESETHAND;
19     act.sa_handler = handler;
20     if (sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0) {
21         perror("sigaction");
22         exit(1);
23     }
24
25     fprintf(stderr, "Input SIGINT: ");
26     pause();
27     fprintf(stderr, "After Signal Handler\n");
28
29     return 0;
30 }
```

시그널 핸들러가
한번 호출된 후
에 시그널 처리
방법이 기본처리
방법으로 재설정

SA_RESETHAND 지정

```
# ex7_7.out
Input SIGINT: ^CReceived Signal:: Interrupt
^CIn Signal Handler, After Sleep
#
```

sigaction 함수의 활용[4]

□ sa_flags에 SA_SIGINFO 플래그를 지정하면 시그널 발생원인을 알 수 있다.

■ 시그널 핸들러의 형식

```
void handler (int sig, siginfo_t *sip, ucontext_t *ucp);
```

- sip : 시그널이 발생한 원인을 담은 siginfo_t 구조체 포인터
- ucp : 시그널을 받는 프로세스의 내부상태를 나타내는 구조체 포인터

■ siginfo_t 구조체

```
typedef struct {  
    int si_signo;  
    int si_errno;  
    int si_code;  
    union sigval si_value;  
    union {  
        ...  
    } __data;  
} siginfo_t;
```

si_signo : 시그널 번호
si_errno : 0 또는 오류번호
si_code : 시그널 발생 원인 코드
__data : 시그널의 종류에 따라 값 저장



sigaction 함수의 활용[5]

□ 시그널 발생 원인 코드

[표 7-8] 사용자 프로세스가 시그널을 생성했을 때 si_code 값

코드	값	의미
SI_USER	0	kill(2), sigsend(2), raise(3), abort(3) 함수가 시그널을 보냄
SI_LWP	-1	_lwp_kill(2) 함수가 시그널을 보냄
SI_QUEUE	-2	sigqueue(3) 함수가 시그널을 보냄
SI_TIMER	-3	timer_settime(3) 함수가 생성한 타이머가 만료되어 시그널을 보냄
SI_ASYNCIO	-4	비동기 입출력 요청이 완료되어 시그널을 보냄
SI_MESGQ	-5	빈 메시지 큐에 메시지가 도착했음을 알리는 시그널을 보냄

□ 시그널 발생 원인 출력: psiginfo(3)

```
#include <siginfo.h>
void psiginfo(siginfo_t *pinfo, char *s);
```

- pinfo : 시그널 발생원인 정보를 저장한 구조체, s: 출력할 문자열



```
...
08 void handler(int signo, siginfo_t *sf, ucontext_t *uc) {
09     psiginfo(sf, "Received Signal:");
10     printf("si_code : %d\n", sf->si_code);
11 }
12
13 int main(void) {
14     struct sigaction act;
15     act.sa_flags = SA_SIGINFO;
16     act.sa_sigaction = (void (*)(int, siginfo_t *, void *))handler;
17     sigemptyset(&act.sa_mask);
18     if (sigaction(SIGUSR1, &act, (struct sigaction *)NULL) < 0) {
19         perror("sigaction");
20         exit(1);
21     }
22
23     pause();
24
25     return 0;
26 }
27 }
```

오류 메시지 출력

SA_SIGINFO 플래그 설정

sigaction 함수 설정

SIGUSR1 시그널 보내기

```
# ex7_8.out&
[1]      2515
# kill -USR1 2515
# Received Signal: : User Signal 1 ( from process 1579 )
si_code : 0
```