10장 시그널과 시그널 처리

- 서론
- 예제 프로그램
- 함수
  - sigemptyset, sigfillset
  - sigaddset, sigdelset, sigismember
  - sigaction
  - sigprocmask
  - kill, raise
  - alarm
  - pause

시그널을 다루기 위해 필요한 시스템 호출/표준 라이브러리 함수

함수	의미
sigemptyset	시그널 집합을 시그널이 없는 비어 있는 상태로 초기화한다.
sigfillset	시그널 집합을 모든 시그널이 포함된 상태로 초기화한다.
sigaddset	시그널 집합에 특정 시그널을 추가한다.
sigdelset	시그널 집합에서 특정 시그널을 삭제한다.
sigaction	특정 시그널에 대한 프로세스의 행동을 설정한다.
sigprocmask	봉쇄할 시그널의 목록을 변경한다.
kill	특정 프로세스에게 특정 시그널을 전달한다.
raise	자기 자신에게 특정 시그널을 전달한다.
alarm	설정된 시간이 경과한 후에 자기 자신에게 시그널을 전달한다.
pause	시그널이 도착할 때까지 대기 상태가 된다.

【예제】 예제 프로그램(2/4)


```
01 #include <unistd.h>
02 #include <signal.h>
03
04 void handler(int signum);
05 int flag = 5;
06
07 main()
08 {
09    struct sigaction act;
10     sigset_t set;
11
12     sigemptyset(&(act.sa_mask));
13    sigaddset(&(act.sa_mask), SIGALRM);
14     sigaddset(&(act.sa_mask), SIGINT);
15    sigaddset(&(act.sa_mask), SIGUSR1);
```

04 시그널 핸들러를 선언한다

09 시그널에 대한 프로세스의 행동을 저장하게 된다

10 시그널의 집합이다

12 시그널 집합을 비어 있는 상태로 생성하고 3개의 시그널을 추가한다

표준입력 스트림

【예제】 예제 프로그램(2/4)


```
16    act.sa_handler = handler;
17   sigaction(SIGALRM, &act, NULL);
18    sigaction(SIGINT, &act, NULL);
19   sigaction(SIGUSR1, &act, NULL);
20
21    printf("call raise(SIGUSR1) before blocking₩n");
22    raise(SIGUSR1);
23
24    sigemptyset(&set);
25    sigaddset(&set, SIGUSR1);
26    sigprocmask(SIG_SETMASK, &set, NULL);
```

16 시그널 핸들러를 등록한다.

17 SIGALRM, SIGINT, SIGUSR1 중에 하나를 수신하면 act에 지정한 방식으로 대응한다.

21 자기 자신에게 SIGUSR1을 전달한다

26 SIGUSR1 시그널을 봉쇄한다. 이후에 SIGUSR1이 발생해도 반응하지 않는다.

표준입력 스트림

【예제】 예제 프로그램(3/4)


```
27   while(flag)
28   {
29      printf("input SIGINT [%d]₩n", flag);
30      sleep(1);
31   }
32
33   printf("call kill(getpid(), SIGUSR1) after blocking₩n");
34   kill(getpid(), SIGUSR1);
35
36   printf("sleep by pause.. zzZZ₩n");
37   printf("pause return %d₩n", pause());
38
39   printf("2 seconds sleeping..zzZ₩n");
40   alarm(2);
41   pause();
42 }
```

34 자기 자신에게 SIGUSR1을 보낸다

37 시그널이 들어올 때까지 대기 상태가 된다

39 2초 뒤에 SIGALRM이 발생한다.

41 시그널이 들어올 때까지 대기 상태가 된다.

표준입력 스트림

【예제】 예제 프로그램(4/4)


```
43 void handler(int signum)
44 {
45     flag--;
46
47     switch(signum) {
48     case SIGINT:
49       printf("SIGINT(%d)₩n", signum);
50         break;
51     case SIGALRM:
52         printf("SIGALRM(%d)₩n", signum);
53         break;
54     case SIGUSR1:
55         printf("SIGUSR1(%d)₩n", signum);
56         break;
57     default:
58         printf("signal(%d)₩n", signum);
59     }
60 }
```

43 시그널 핸들러

47 수신된 시그널에 따라 할 일이 결정된다

표준입력 스트림

실행 결과

```
$ ex10-01
call raise(SIGUSR1) before blocking
SIGUSR1(10)
input SIGINT [4]
input SIGINT [4]
SIGINT(2)
input SIGINT [3]
SIGINT(2)
input SIGINT [2]
input SIGINT [2]
SIGINT(2)
input SIGINT [1]
SIGINT(2)
call kill(getpid(), SIGUSR1) after blocking
sleep by pause.. zzZZ
SIGINT(2)
pause return -1
2 seconds sleeping..zzZ
SIGALRM(14)
$
```

시그널 집합을 생성하거나 조작한다.

```
#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 집합이다.
signum	시그널 번호이다.
반환값	호출이 성공하면 0을 반환하고, 실패하면 -1을 반환한다. 단 sigisnumber는 호출이 성공하면 1이나 0을 반환하고 실패하면 -1을 반환한다.

- **시그널을 다루려면 시그널 집합을 만들어야 한다.**

sigemptyset(set)

set으로 주어진 시그널 집합을 아무런 시그널도 포함되어 있지 않은 비어 있는 상태로 초기화한다.

sigfillset(set)

sigemptyset와는 반대로 모든 시그널이 포함된 상태로 시그널 집합을 초기화한다.

sigaddset, sigdelset

시그널 집합에서 지정한 시그널을 추가하거나 제거한다.

sigismember

시그널 집합에 지정한 시그널이 포함되어 있는지를 검사한다.

【예제 10-2】 ex10-02.c(1/2)


```
01 #include <signal.h>
02 #include <unistd.h>
03
04 main()
05 {
06         sigset_t set;
07         int result;
08
09      sigemptyset(&set);
10         result = sigismember(&set, SIGALRM);
11         printf("SIGALRM is %s a member₩n", result ? "" : "not");
12         sigaddset(&set, SIGALRM);
13         result = sigismember(&set, SIGALRM);
14         printf("SIGALRM is %s a member₩n", result ? "" : "not");
```

표준입력 스트림

【예제 10-2】 ex10-02.c(2/2)


```
15                  sigfillset(&set);
16          result = sigismember(&set, SIGCHLD);
17          printf("SIGCHLD is %s a member₩n", result ? "" : "not");
18          sigdelset(&set, SIGCHLD);
19          result = sigismember(&set, SIGCHLD);
20          printf("SIGCHLD is %s a member₩n", result ? "" : "not");
21  }
```

표준입력 스트림

```
$ ex10-02
SIGALRM is not a member
SIGALRM is  a member
SIGCHLD is  a member
SIGCHLD is not a member
$
```

특정한 시그널을 받았을 때 프로세스가 취해야 할 행동을 지정한다

```
#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.

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

sigaction 구조체

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

void (*sa_handler)(int);

시그널 핸들러, signum으로 지정한 특정 시그널이 들어왔을 때 수행해야 할 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 함수이다.

- void (*sa_sigaction)(int, siginfo_t *, void *);
  - 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.

- sigset_t sa_mask;
  - 시그널 마스크 : 시그널 집합을 의미한다.
  - 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.

- **int sa_flags;**

  시그널 처리 절차를 수정하는데 사용된다.

값	의미
SA_SIGINFO	sa_handler 대신에 sa_sigaction을 선택한다.
SA_NOCLDSTOP	signum이 SIGCHLD일 때 자식 프로세스가 종료되거나 중단되더라도 부모 프로세스는 이를 알려고 하지 않는다.
SA_ONESHOT	signum에 대해서 시그널 핸들러를 최초에 한번만 실행하고 그 다음부터는 동일한 시그널에 대해서 SIG_DFL에 해당하는 기본적인 동작만 수행하게 된다.
SA_RESETHAND	SA_ONESHOT과 같다.

【예제 10-3】 ex10-03.c(1/2)


```
01 #include <signal.h>
02 #include <unistd.h>
03
04 int num = 0;
05
06 main()
07 {
08     static struct sigaction act;
09
10     void int_handle(int);
11
12     act.sa_handler = int_handle;
13     sigfillset(&(act.sa_mask));
14     sigaction(SIGINT, &act, NULL);
```

표준입력 스트림

【예제 10-3】 ex10-03.c(2/2)


```
15     while(1)
16     {
17         printf("i'm sleepy..₩n");
18         sleep(1);
19         if(num >= 3)
20             exit(0);
21     }
22 }
23
24 void int_handle(int signum)
25 {
26     printf("SIGINT:%d₩n", signum);
27     printf("int_handle called %d times₩n", ++num);
28 }
```

```
$ ex10-03
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..
SIGINT:2
int_handle called 3 times
$
```

【예제 10-4】 ex10-04.c(1/2)


```
01 #include <signal.h>
02 #include <unistd.h>
03
04 int num = 0;
05
06 main()
07 {
08    static struct sigaction act;
09
10     void int_handle(int);
11
12     act.sa_handler = int_handle;
13     sigfillset(&(act.sa_mask));
14     sigaction(SIGINT, &act, NULL);
```


【예제 10-4】 ex10-04.c(2/2)


```
15    while(1) {
16       printf("i'm sleepy..₩n");
17       sleep(1);
18       if(num >= 2) {
19          act.sa_handler = SIG_DFL;
20          sigaction(SIGINT, &act, NULL);
21       }
22    }
23 }
24
25 void int_handle(int signum)
26 {
27       printf("SIGINT:%d₩n", signum);
28       printf("int_handle called %d times₩n", ++num);
29 }
```

```
$ ex10-04
i'm sleepy..
SIGINT:2
int_handle called 1 times
i'm sleepy..
SIGINT:2
int_handle called 2 times
i'm sleepy..

$
```

시그널 집합 단위로 봉쇄될 시그널들의 목록을 설정한다.

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

how	sigprocmask 함수의 행동 방식을 결정한다.
set	새롭게 적용하는 시그널 마스크이다.
oldset	이전에 적용되어 있는 시그널 마스크를 저장한다.
반환값	호출이 성공할 경우 0을 반환하고, 실패하면 -1을 반환한다.

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

시그널 봉쇄 → 중요한 작업 수행 → 시그널 봉쇄 해제

- **int 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.

【예제 10-5】 ex10-05.c(1/2)


```
01 #include <unistd.h>
02 #include <signal.h>
03
04 main()
05 {
06     sigset_t set;
07     int count = 3;
08
09     sigemptyset(&set);
10     sigaddset(&set, SIGINT);
11
12     sigprocmask(SIG_BLOCK, &set, NULL);
```

표준입력 스트림

【예제 10-5】 ex10-05.c(2/2)


```
13    while(count)
14    {
15        printf("don't disturb me (%d)₩n", count--);
16        sleep(1);
17    }
18
19    sigprocmask(SIG_UNBLOCK, &set, NULL);
20
21    printf("you did not disturb me!!₩n");
22 }
```

표준입력 스트림

실행 결과

```
$ ex10-05
don't disturb me (3)
don't disturb me (2)
don't disturb me (1)
you did not disturb me!!
$ ex10-05
don't disturb me (3)
don't disturb me (2)                    ← 이쯤에서 Ctrl+C를 입력한다.
don't disturb me (1)

$
```

- **kill은 특정 프로세스나 프로세스 그룹에게 지정한 시그널을 전달하고, raise는 자기 자신에게 지정한 시그널을 전달한다.**

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

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

pid	프로세스의 식별 번호이다.
sig	시그널 번호이다.
반환값	호출이 성공하면 0을 반환하고, 실패하면 -1을 반환한다.

kill의 pid의 값에 따른 의미

pid	의미
pid > 0	프로세스의 식별 번호이다. 해당 프로세스에게만 시그널을 보낸다
pid = 0	자신과 같은 그룹에 있는 모든 프로세스에게 시그널을 보낸다.
pid = -1	자신과 같은 그룹에 있는 모든 프로세스에게 시그널을 보낸다. 단, 프로세스 식별 번호가 1인 프로세스를 제외한다.
pid < -1	프로세스 그룹 식별 번호가 -pid인 모든 프로세스에게 시그널을 보낸다. pid가 -100이라면 그룹 식별 번호가 100인 모든 프로세스를 의미한다.

kill과 raise

```
raise(sig);
는 아래와 같다.
kill(getpid(), sig);
```

【예제 10-6】 ex10-06.c(1/2)


```
01 #include <unistd.h>
02 #include <signal.h>
03 #include <sys/types.h>
04
05 main()
06 {
07     pid_t pid;
08     int count = 5;
09
10     if((pid = fork()) > 0)
11     {
12         sleep(2);
13         kill(pid, SIGINT);
14         raise(SIGINT);
15         printf("[parent] bye!₩n");
16     }
```

표준입력 스트림

【예제 10-6】 ex10-06.c(2/2)


```
17    else if(pid == 0)
18    {
19       while(count)
20       {
21          printf("[childe] count is %d₩n", count--);
22          sleep(1);
23       }
24    }
25    else
26       printf("fail to fork₩n");
27 }
```

표준입력 스트림

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

$
```

지정한 시간이 경과한 후에 자신에게 SIGALRM 시그널을 보낸다

```
#include <unistd.h>

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

seconds	초 단위의 시간이다.
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.

【예제 10-7】 ex10-07.c(1/2)


```
01 #include <unistd.h>
02 #include <signal.h>
03
04 void timeover(int signum)
05 {
06     printf("₩n₩ntime over!!₩n₩n");
07     exit(0);
08 }
09
10 main()
11{
12     char buf[1024];
13     char *alpha = "abcdefghijklmnopqrstuvwxyz";
14
15     int timelimit;
16     struct sigaction act;
17
18     act.sa_handler = timeover;
19     sigaction(SIGALRM, &act, NULL);
```

표준입력 스트림

【예제 10-7】 ex10-07.c(2/2)


```
20      printf("input timelimit (sec)..₩n");
21      scanf("%d", &timelimit);
22
23      alarm(timelimit);
24
25      printf("START!!₩n > ");
26      scanf("%s", buf);
27
28      if(!strcmp(buf, alpha))
29          printf("well done.. you succeed!₩n");
30      else
31          printf("sorry.. you fail!₩n");
32  }
```

표준입력 스트림

```
$ ex10-07
input timelimit (sec)..
100
START!!
> abcdefghijklmnopqrstuvwxyz
well done.. you succeed!
$ ex10-07
input timelimit (sec)..
3
START!!
> abcdefghijk

time over!!

$
```

시그널이 전달될 때까지 대기한다

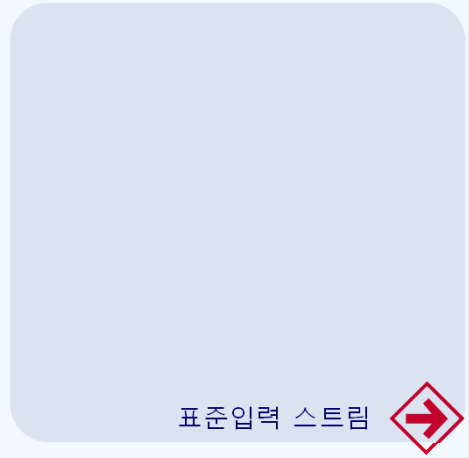
```
#include <unistd.h>

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 10-8】 ex10-08.c


```
01 #include <unistd.h>
02 #include <signal.h>
03
04 main()
05 {
06         printf("pause return %d₩n", pause());
07 }
```

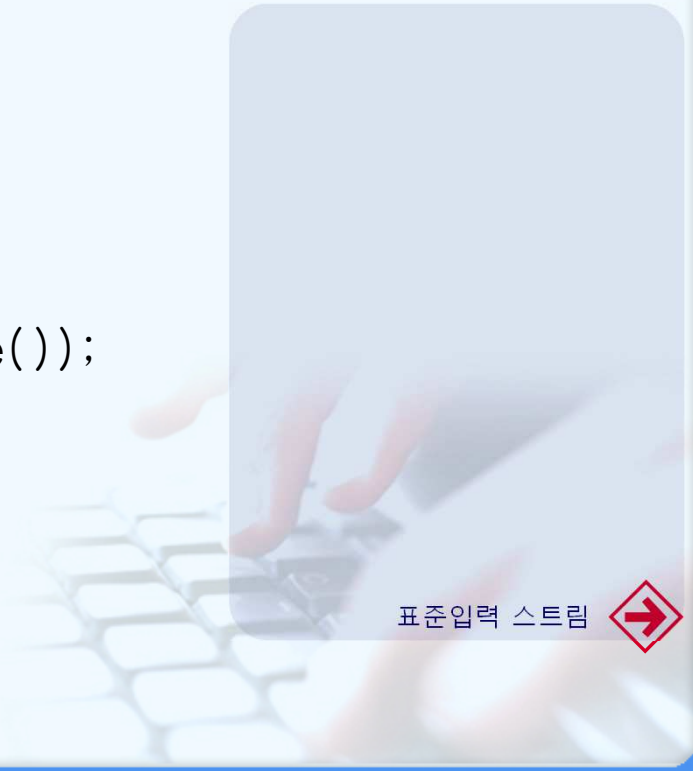



```
$ ex10-08
                                        ← Ctrl+C를 입력한다.
$
```

【예제 10-9】 ex10-09.c


```
01 #include <unistd.h>
02 #include <signal.h>
03
04 void handler(int signum);
05
06 main()
07 {
08         struct sigaction act;
09
10         sigfillset(&(act.sa_mask));
11         act.sa_handler = handler;
12         sigaction(SIGINT, &act, NULL);
13
14         printf("pause return %d₩n", pause());
15 }
16
17 void handler(int signum)
18 {
19         printf("₩nSIGINT coughtt₩n₩n");
20 }
```

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

pause return -1
$
```



표준입력 스트림