7 시그널

학습목표

- 시그널의 기본 개념을 이해한다.
- 시그널을 보내는 방법을 이해한다.
- 시그널을 받아서 처리하는 기본적인 방법을 이해한다.
- 시그널 집합의 개념과 사용방법을 이해한다.
- sigaction 함수를 사용해 시그널을 처리하는 방법을 이해한다.
- 알람 시그널의 처리방법을 이해한다.
- 시그널 관련 기타 함수들의 사용방법을 이해한다.

목차

- 시그널의 개념
- 시그널의 종류
- 시그널 보내기
- 시그널 핸들러 함수
- 시그널 집합
- sigaction 함수의 활용
- 알람시그널과 인터벌 타이머
- 기타 시그널 관련 함수

시그널의 개념

시그널

- 소프트웨어 인터럽트 프로세스에 뭔가 발생했음을 알리는 간단한 메시지를 비동기적으로 보내는 것

정수

발생사유

- 0으로 나누기처럼 프로그램에서 예외적인 상황이 일어나는 경우
- kill 함수처럼 시그널을 보낼 수 있는 함수를 사용해서 다른 프로세스에 시그널을 보내는 경우
- 사용자가 Ctrl+C와 같이 인터럽트 키를 입력한 경우

시그널 처리방법

- 각 시그널에 지정된 기본 동작 수행. 대부분의 기본 동작은 프로세스 종료
- 시그널을 무시
- 시그널 처리를 위한 함수(시그널 핸들러)를 지정해놓고 시그널을 받으면 해당 함수 호출
- 시그널이 발생하지 않도록 블록처리

시그널의 종류

시그널	번호	기본 처리	발생 요건
SIGHUP	1	종료	행업으로 터미널과 연결이 끊어졌을 때 발생
SIGINT	2	종료	인터럽트로 사용자가 Ctrl+C를 입력하면 발생
SIGQUIT	3	코어 덤프	종료 신호로 사용자가 Ctrl+\를 입력하면 발생
SIGILL	4	코어 덤프	잘못된 명령 사용
SIGTRAP	5	코어 덤프	추적(trace)이나 중단점(breakpoint)에서 트랩 발생
SIGABRT	6	코어 덤프	abort 함수에 의해 발생
SIGEMT	7	코어 덤프	에뮬레이터 트랩으로 하드웨어에 문제가 있을 경우 발생
SIGFPE	8	코어 덤프	산술 연산 오류로 발생
SIGKILL	9	종료	강제 종료로 발생
SIGBUS	10	코어 덤프	버스 오류로 발생
SIGSEGV	11	코어 덤프	세그먼테이션 폴트로 발생
SIGSYS	12	코어 덤프	잘못된 시스템 호출로 발생
SIGPIPE	13	종료	잘못된 파이프 처리로 발생
SIGALRM	14	종료	알람에 의해 발생
SIGTERM	15	종료	소프트웨어적 종료로 발생
SIGUSR1	16	종료	사용자 정의 시그널 1

SIGCONT: continue, 계속하라

이외에도 시그널의 종류는 다양함 (표 7-7 참조)

시그널 보내기[1]

kill 명령

- 프로세스에 시그널을 보내는 명령
- 예 : 3255번 프로세스에 9번 시그널(SIGKILL) 보내기 -> 프로세스 강제 종료

```
# kill -9 3255
```

pid

시그널 보내기: kill(2)

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig);
```

sig

- pid가 0보단 큰 수 : pid로 지정한 프로세스에 시그널 발송
- pid가 -1이 아닌 음수 : 프로세스 그룹ID가 pid의 절대값인 프로세스 그룹에 속하고 시그널을 보낼 권한을 가지고 있는 모든 프로세스에 시그널 발송 -5 = 5번 그룹
- pid가 0 : 특별한 프로세스를 제외하고 프로세스 그룹ID가 시그널을 보내는 프로세스의 프로세스 그룹ID와 같은 모든 프로세스에게 시그널 발송
- pid가 -1 : 시그널을 보낸는 프로세스의 유효 사용자ID가 root가 아니면, 특별한 프로세스를 제외하고 프로세스의 실제 사용자ID가 시그널을 보내는 프로세스의 유효 사용자ID와 같은 모든 프로세스에 시그널 발송

[예제 7-1] kill 함수 사용하기

ex7_1.c

```
01  #include <sys/types.h>
02  #include <unistd.h>
03  #include <signal.h>
04  #include <stdio.h>
05
06  int main(void) {
07      printf("Before SIGCONT Signal to parent.\n");
08
09      kill(getppid(), SIGCONT);
10
11      printf("Before SIGQUIT Signal to me.\n");
12
13      kill(getpid(), SIGQUIT);
14
15      printf("After SIGQUIT Signal.\n");
16
17      return 0;
18  }
```

부모에 SIGCONT 보내기

나에게 SIGQUIT 보내기

SIGQUIT의 기본동작은 코어덤프

종료되어서

```
# ex7_1.out
Before SIGCONT Signal to parent.
Before SIGQUIT Signal to me.
끝(Quit)(코어 덤프)
```

After

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[1]

시그널 핸들러

- 시그널을 받았을 때 이를 처리하기 위해 지정된 함수
- 프로세스를 종료하기 전에 처리할 것이 있거나, 특정 시그널에 대해 종료하고 싶지 않을 경우 지정

시그널 핸들러 지정: signal(3)

```
#include <signal.h>
void (*signal(int sig, void (*disp)(int)))(int);
```

받은신호 할 동작

- disp : sig로 지정한 시그널을 받았을 때 처리할 방법
  - 시그널 핸들러 함수명
  - SIG_IGN : 시그널을 무시하도록 지정
  - SIG_DFL : 기본 처리 방법으로 처리하도록 지정
- signal함수는 시그널이 들어올 때마다 시그널 핸들러를 호출하려면 매번 시그널 핸들러를 재지정해야함. 그러나 옛방식임

[예제 7-2] signal 함수 사용하기

ex7_2.c

```
...                                          <unistd.h> <signal.h> <stdlib.h> <stdio.h>
07  void handler(int signo) {                신호 받고
08      printf("Signal Handler Signal Number : %d\n", signo);
09      psignal(signo, "Received Signal");   받았다고 표시
10  }   원하고 싶다면 여기에 exit(1) 이나
11      signal(SIGINT, SIG_DFL); 넣기
12  int main(void) {
13      void (*hand)(int);
14
15      hand = signal(SIGINT, handler);
16      if (hand == SIG_ERR) {
17          perror("signal");
18          exit(1);
19      }SIGINT가 발생하면 handler 작동시켜라
20      그리 다른 신호가 와도 멈춤
21      printf("Wait 1st Ctrl+C... : SIGINT\n");
22      pause(); 신호가 올때까지 중단
23      printf("After 1st Signal Handler\n");
24      printf("Wait 2nd Ctrl+C... : SIGINT\n");
25      pause();
26      printf("After 2nd Signal Handler\n");
28      return 0;
29  }
```

```
# ex7_2.out
Wait 1st Ctrl+C... : SIGINT
^CSignal Handler Signal Number : 2
Received Signal: Interrupt
After 1st Signal Handler
Wait 2nd Ctrl+C... : SIGINT
^C#
```

두번째 Ctrl+C는 처리못함

일회용 handler,
그러나 현재는
결과가 끝까지
잘 나온다

[예제 7-3] 시그널 핸들러 재지정하기

ex7_3.c

```
...
07  void handler(int signo) {
08       void (*hand)(int);
09       hand = signal(SIGINT, handler);
10       if (hand == SIG_ERR) {
11            perror("signal");
12            exit(1);
13       }
14
15       printf("Signal Handler Signal Number: %d\n", signo);
16       psignal(signo, "Received Signal");
17  }
...
```

시그널 핸들러 재지정

```
# ex7_3.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
```

두번째 Ctrl+C도 처리

같은 결과, 현재는 같이?

시그널 핸들러 함수[2]

- 시그널 핸들러 지정: sigset(3) 우분투에서 지원 안함, 그리고 쿨이?

```
#include <signal.h>

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

  - disp : sig로 지정한 시그널을 받았을 때 처리할 방법
    - 시그널 핸들러 함수명
    - SIG_IGN : 시그널을 무시하도록 지정
    - SIG_DFL : 기본 처리 방법으로 처리하도록 지정

  - sigset함수는 signal함수와 달리 시그널 핸들러가 한 번 호출된 후에 기본동작으로 재설정하지 않고, 시그널 핸들러를 자동으로 재정한다.

[예제 7-4] sigset 함수 사용하기

ex7_4.c

```
...
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 않아도됨

```
# 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 마스크로 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

ex7_5.c

```
01  #include <signal.h>
02  #include <stdio.h>
03
04  int main(void) {
05      sigset_t st;
06
07      sigemptyset(&st);
08
09      sigaddset(&st, SIGINT);
10      sigaddset(&st, SIGQUIT);
11
12      if (sigismember(&st, SIGINT))
13          printf("SIGINT is setting.\n");
14
15      printf("** Bit Pattern: %x\n",st.__sigbits[0]);
16
17      return 0;
18  }
```

시그널 집합 비우기

시그널 추가

시그널 설정 확인

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

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

sigaction 함수의 활용[1]

sigaction 함수 Signal 상위 호환

- 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 { union : 위를 쓸건지 아래를 쓸건지 정해짐
      ①void (*sa_handler)();
      ②void (*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 재시작하게 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 new
- oact : 기존에 시그널을 처리하던 방법을 저장할 구조체 주소 old

- 첫번째 인자로 SIGKILL과 SIGSTOP을 제외한 어떤 시그널도 올 수 있음

SIGKILL 과 SIGSTOP 은 첫번째 인자가 될 수 없다

[예제 7-6] sigaction 함수 사용하기(1)

ex7_6.c

```
... <unistd.h> <signal.h> <stdlib.h> <stdio.h>
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 = 0;
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  }
```

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)
```

^C 하면 곧바로 나옴

[예제 7-7] sigaction 함수 사용하기(SA_RESETHAND)

ex7_7_arg.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 후에 시그널 처리 방법이 기본처리 방법으로 재설정

SA_RESETHAND 지정

이것을 땐 계속 했는데, 이전 handler 실행 완시키고 종료함

```
# 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

[예제 7-8] 시그널 발생원인 검색하기

ex7_8.c

```
...
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
16    act.sa_flags = SA_SIGINFO;
17    act.sa_sigaction = (void (*)(int, siginfo_t *, void *))handler;
18    sigemptyset(&act.sa_mask);
19    if (sigaction(SIGUSR1, &act, (struct sigaction *)NULL) < 0) {
20        perror("sigaction");
21        exit(1);
22    }
23
24    pause();
25
26    return 0;
27  }
```

오류 메시지 출력

SA_SIGINFO 플래그 설정

sigaction 함수 설정

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

SIGUSR1 시그널 보내기

알람 시그널

알람 시그널

- 일정한 시간이 지난 후에 자동으로 시그널이 발생하도록 하는 시그널
- 일정 시간 후에 한 번 발생시키거나, 일정 간격을 두고 주기적으로 발송 가능

알람 시그널 생성: alarm(2)

```
#include <unistd.h>

unsigned int alarm(unsigned int sec);
```

- sec : 알람이 발생시킬 때까지 남은 시간(초 단위)
- 일정 시간이 지나면 SIGALRM 시그널 발생
- 프로세스별로 알람시계가 하나 밖에 없으므로 알람은 하나만 설정 가능

[예제 7-9] alarm 함수 사용하기

ex7_9.c

```
01  #include <unistd.h>
02  #include <signal.h>
03  #include <siginfo.h>
04  #include <stdio.h>
05
06  void handler(int signo) {
07      psignal(signo, "Received Signal");
08  }
09
10  int main(void) {
11      sigset(SIGALRM, handler);
12
13      alarm(2);
14      printf("Wait...\n");
15      sleep(3);
16
17      return 0;
18  }
```

알람신호 받으면 handler 실행

2초 설정

3초를 자고 싶지만 2초뒤에 알람이 울릴 것임

```
# ex7_9.out
Wait...
Received Signal: Alarm Clock
```

인터벌 타이머

- 타이머의 종류 which
  - ITIMER_REAL : 실제 시간 사용. SIGALRM 시그널 발생
  - ITIMER_VIRTUAL : 프로세스의 가상 시간 사용. SIGVTALRM 시그널 발생
  - ITIMER_PROF : 시스템이 프로세스를 위해 실행중인 시간과 프로세스의 가상 시간을 모두 사용. SIGPROF 시그널 발생
  - ITIMER_REALPROF : 실제 시간 사용. 멀티스레드 프로그램의 실제 실행시간 측정시 사용. SIGPROF 시그널 발생

- 타이머 정보 검색: getitimer(2)

```
#include <sys/time.h>

int getitimer(int which, struct itimerval *value);
```

which 구조체

- 타이머 설정: setitimer(2)

```
#include <sys/time.h>

int setitimer(int  which,  const  struct  itimerval  *value,
              struct itimerval *ovalue);
```

old/null

인터벌 타이머

- which : 타이머 종류
- value : 타이머정보 구조체 포인터

```
struct itimerval {
    struct timeval it_interval;
    struct timeval it_value;
};
```

```
struct timeval {
    time_t tv_sec;
    suseconds_t tv_usec;
};
```

구조체 안에 구조체가 있다

[예제 7-10] 인터벌 타이머 설정하기

ex7_10.c

```
...<sys/time.h><unistd.h><signal.h><stdlib.h><stdio.h>
11  int main(void) {
12      struct itimerval it;
13
14      sigset(SIGALRM, handler);
15      it.it_value.tv_sec = 3;
16      it.it_value.tv_usec = 0;
17      it.it_interval.tv_sec = 2;
18      it.it_interval.tv_usec = 0;
19
20      if (setitimer(ITIMER_REAL, &it, (struct itimerval *)NULL) == -1) {
21          perror("setitimer");
22          exit(1);
23      }
24
25      while (1) {
26          if (getitimer(ITIMER_REAL, &it) == -1) {
27              perror("getitimer");
28              exit(1);
29          }
30          printf("%d sec, %d msec.\n", (int)it.it_value.tv_sec
31                                       (int)it.it_value.tv_u
32          sleep(1);
33      }
34
35      return 0;
36  }
```

타이머 간격 : 2초
타이머에 현재 남은 시간 : 3초

3초 후에 최초 시그널 발생
이후 2초 간격으로 시그널 발생

3초 후 2초 간격으로 작동하라

남은 시간 정보 출력

```
# ex7_10.out
2 sec, 999997 msec.
1 sec, 999998 msec.
0 sec, 992047 msec.
Timer Invoked..handler
1 sec, 991565 msec.
0 sec, 982071 msec.
Timer Invoked..
1 sec, 991433 msec.
0 sec, 981829 msec.
Timer Invoked..
1 sec, 991218 msec.
```

기타 시그널 처리 함수[1]

시그널 정보 출력: psignal(3)

```
#include <siginfo.h>

void psignal(int sig, const char *s);
```

- s에 지정한 문자열을 붙여 정보 출력

시그널 정보 출력: strsignal(3)

```
#include <string.h>

char *strsignal(int sig);
```

- 인자로 받은 시그널을 가리키는 이름을 문자열로 리턴

기타 시그널 처리 함수[2]

시그널 블록킹과 해제

```
#include <signal.h>

int sighold(int sig);
int sigrelse(int sig);
```

- 인자로 받은 시그널을 시그널 마스크에 추가하거나 해제

시그널 집합 블록과 해제: sigprocmask(2)

```
#include <signal.h>

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

- how : 시그널을 블록할 것인지, 해제할 것인지 여부
  - SIG_BLOCK : set에 지정한 시그널 집합을 시그널 마스크에 추가
  - SIG_UNBLOCK : set에 지정한 시그널 집합을 시그널 마스크에서 제거
  - SIG_SETMASK : set에 지정한 시그널 집합으로 현재 시그널 마스크 대체
- set : 블록하거나 해제할 시그널 집합 주소
- oset : NULL 또는 이전 설정값을 저장한 시그널 집합주소

[예제 7-11] 시그널 블록함수 사용하기

ex7_11.c

```
#include <unistd.h><signal.h><stdlib.h><stdio.h><string.h>
07  void handler(int signo) {
08      char *s;
09
10      s = strsignal(signo);
11      printf("Received Signal : %s\n", s);
12  }
13
14  int main(void) {
15      if (sigset(SIGINT, handler) == SIG_ERR) {
16          perror("sigset");
17          exit(1);
18      }
19
20      sighold(SIGINT);
21
22      pause();
23
24      return 0;
25  }
```

시그널 이름 리턴

시그널 핸들러 설정

SIGINT 블록설정

SIGINT 시그널을 안받는다

```
# ex7_11.out
^C^C^C^C^C
```

[예제 7-12] sigprocmask 함수 사용하기

ex7_12.c

```
#include <unistd.h><signal.h><stdio.h>
05  int main(void) {
06      sigset_t new;
07
08      sigemptyset(&new);
09      sigaddset(&new, SIGINT);
10      sigaddset(&new, SIGQUIT);
11      sigprocmask(SIG_BLOCK, &new, (sigset_t *)NULL);
12
13      printf("Blocking Signals : SIGINT, SIGQUIT\n");
14      printf("Send SIGQUIT\n");
15      kill(getpid(), SIGQUIT);
16
17      printf("UnBlocking Signals\n");
18      sigprocmask(SIG_UNBLOCK, &new, (sigset_t *)NULL);
19
20      return 0;
21  }
```

시그널 집합에 SIGINT, SIGQUIT 설정

시그널 집합 블록설정

SIGQUIT 시그널 보내기

kill을 막음

시그널 집합 블록 해제

막음 해제하여 kill 실행됨

```
# ex7_12.out
Blocking Signals : SIGINT, SIGQUIT
Send SIGQUIT
UnBlocking Signals
끝(Quit)(코어 덤프)
```

블록해제 후 시그널을 받아 종료

기타 시그널 처리 함수[3]

시그널 대기 : sigpause(3)

```
#include <signal.h>

int sigpause(int sig);
```

- sig : 시그널이 올 때까지 대기할 시그널

시그널 기다리기: sigsuspend(2)

```
#include <signal.h>

int sigsuspend(const sigset_t *set);
```

- set : 기다리려는 시그널을 지정한 시그널 집합

[예제 7-13] sigsuspend 함수 사용하기

ex7_13.c

```
#include <unistd.h><signal.h><stdio.h> //<siginfo.h>
06  void handler(int signo) {
07      psignal(signo, "Received Signal:");
08  }
09
10  int main(void) {
11      sigset_t set;
12
13      sigset(SIGALRM, handler);
14
15      sigfillset(&set);
16      sigdelset(&set, SIGALRM);
17
18      alarm(3);
19
20      printf("Wait...\n");
21
22      sigsuspend(&set);
23
24      return 0;
25  }
```

기다릴 시그널 설정

알람시그널 설정

시그널 기다리기

```
# ex7_13.out
Wait...
^C^CReceived Signal:: Alarm Clock
```

기타 시그널 처리 함수[4]

시그널 보내기: sigsend(2)

```
#include <signal.h>

int sigsend(idtype_t idtype, id_t id, int sig);
```

- idtype : id에 지정한 값의 종류
- id : 시그널을 받을 프로세스나 프로세스 그룹
- sig : 보내려는 시그널

값	의미
P_PID	프로세스 ID가 id인 프로세스에 시그널을 보낸다.
P_PGID	프로세스 그룹 ID가 id인 모든 프로세스에 시그널을 보낸다.
P_SID	세션 ID가 id인 모든 프로세스에 시그널을 보낸다.
P_TASKID	태스크 ID가 id인 모든 프로세스에 시그널을 보낸다.
P_UID	유효 사용자 ID(EUID)가 id인 모든 프로세스에 시그널을 보낸다.
P_GID	유효 그룹 ID(EGID)가 id인 모든 프로세스에 시그널을 보낸다.
P_PROJID	프로젝트 ID가 id인 모든 프로세스에 시그널을 보낸다.
P_CID	스케줄러 클래스 ID가 id인 모든 프로세스에 시그널을 보낸다.
P_CTID	프로세스 콘트랙트 ID가 id인 모든 프로세스에 시그널을 보낸다.
P_ALL	id를 무시하고 모든 프로세스에 시그널을 보낸다.
P_MYID	함수를 호출하는 자신에게 시그널을 보낸다.

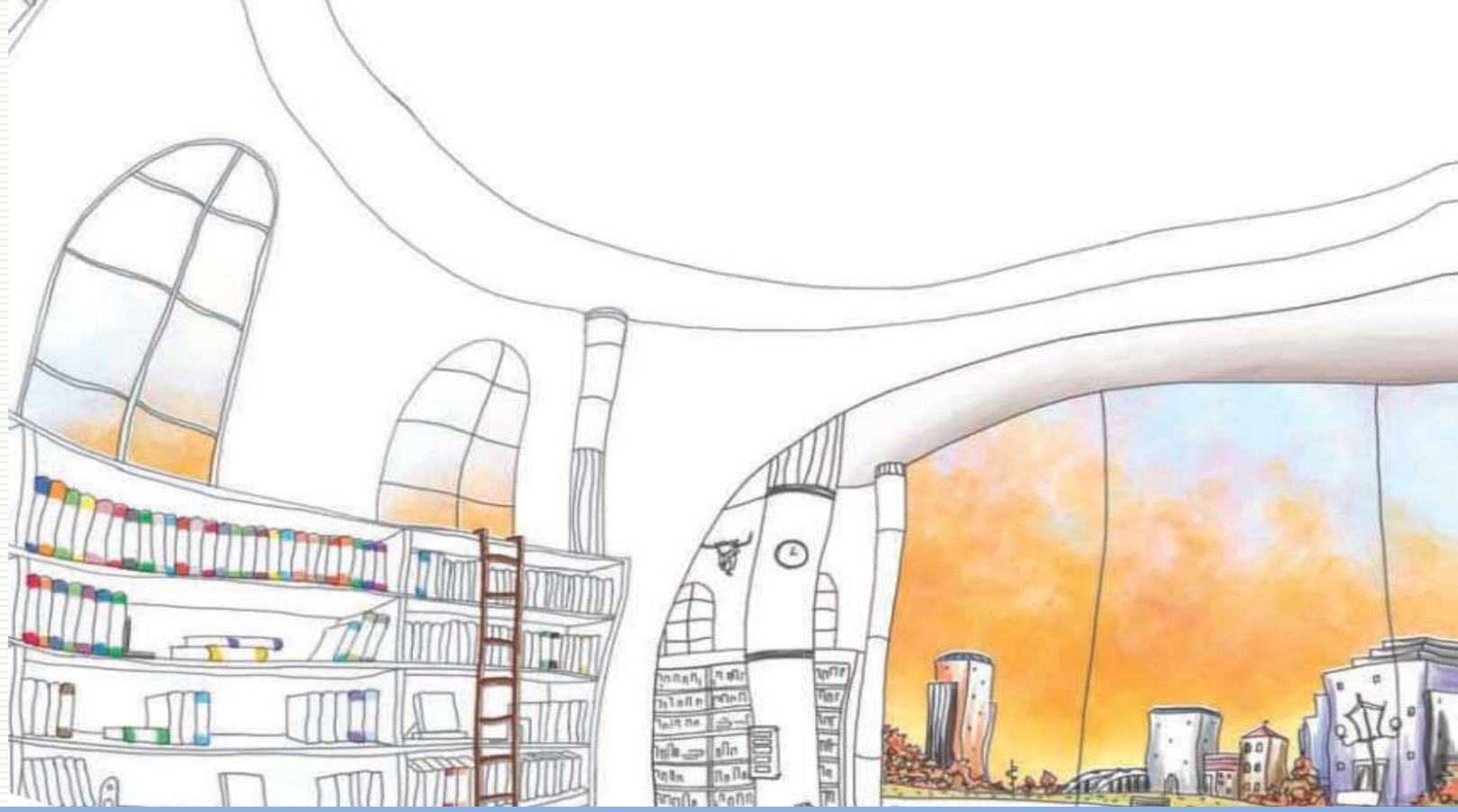
기타 시그널 처리 함수[5]

- 시그널 무시처리 : sigignore(3)

```
#include <signal.h>

int sigignore(int sig);
```

sig : 무시할 시그널 번호
인자로 지정한 시그널의 처리방법을 SIG_IGN으로 설정



Thank You !

IT CookBook, 유닉스 시스템 프로그래밍

한빛미디어 Hanbit Media, Inc.