9주차 3차시 시그널 관련 함수

【학습목표】

1. 시그널 관련 처리 함수를 사용할 수 있다.
2. 인터벌 타이머의 기본 개념을 이해하고 사용할 수 있다.

학습내용1 : 시그널 관련 함수

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
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

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() 함수

첫번째 인자로 SIGKILL과 SIGSTOP을 제외한 어떤 시그널도 올 수 있음

```
#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

* sigaction 함수 사용하기(1)

```
...
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)
```

* 시그널 발생 원인 검색

sa_flags에 SA_SIGINFO 플래그를 지정하면 시그널 발생원인을 알 수 있다.

```
void handler (int sig, siginof_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

* 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

2. 시그널 발생 원인 출력: psiginfo(3)

시그널 발생 원인을 출력 : si_code 값이 양수면 시스템에서 시스널 생성

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

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

* 시그널 발생원인 검색하기

```
...
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

학습내용2 : 인터벌 타이머

1. 알람 시그널

* 알람 시그널의 개념

일정한 시간이 지난 후에 자동으로 시그널이 발생하도록 하는 시그널
일정 시간 후에 한 번 발생시키거나, 일정 간격을 두고 주기적으로 발송 가능
일정 시간이 지나면 SIGALRM 시그널 발생
프로세스별로 알람시계가 하나 밖에 없으므로 알람은 하나만 설정 가능

* 알람 시그널 생성: alarm(2)

```
#include <unistd.h>

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

sec : 알람이 발생시킬 때까지 남은 시간(초 단위)

* alarm 함수 사용하기

```
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  }
```

2초 설정

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

2. 인터벌 타이머 : 정해진 시간에 한번 시그널 보냄

* 타이머의 종류
① ITIMER_REAL : 실제 시간 사용. SIGALRM 시그널 발생
② ITIMER_VIRTUAL : 프로세스의 가상 시간 사용. SIGVTALRM 시그널 발생
③ ITIMER_PROF : 시스템이 프로세스를 위해 실행중인 시간과 프로세스의 가상 시간을 모두 사용. SIGPROF 시그널 발생
④ ITIMER_REALPROF : 실제 시간 사용. 멀티스레드 프로그램의 실제 실행시간 측정시 사용. SIGPROF 시그널 발생

* 타이머 정보 검색: getitimer(2)
타이머 정보 검색 함수

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

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

which : 검색할 타이머의 종류
value : 타이머 정보를 저장할 구조체 포인터

- itimerval / timeval 구조체

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

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

it_interval : 타이머 간격 정보 저장
it_value : 타이머가 만료될까지 남은 시간 저장.
tv_sec : 초 단위 시간 저장
tv_usec : 마이크로초 단위 저장

* 타이머 설정: setitimer(2)

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

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

which : 설정할 타이머의 종류
value : 설정할 타이머 정보를 저장한 구조체 포인터
ovalue : 이전 타이머 정보를 저장할 구좇체 포인터

* 인터벌 타이머 설정하기

```
...
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

남은 시간 정보 출력

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

3. 기타 시그널 처리 함수

* 시그널 블록킹과 해제

인자로 받은 시그널을 시그널 마스크에 추가하거나 해제

```
#include <signal.h>

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

int sig : 블록하거나 해제할 시그널

* 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

* 시그널 블록함수 사용하기

```
...
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
```

* sigprocmask 함수 사용하기

```
...
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

시그널 집합 블록 해제

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

블록해제 후 시그널을 받아 종료

* 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

* sigsuspend 함수 사용하기 ex7_13.c

```
...
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
```

* 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.

* 시그널 무시처리 : sigignore(3)
인자로 지정한 시그널의 처리방법을 SIG_IGN으로 설정

```
#include <signal.h>

int sigignore(int sig);
```

sig : 무시할 시그널 번호

【학습정리】

1. 시그널 관련 함수
* sigaction 함수
- signal이나 sigset 함수처럼 시그널을 받았을 때 이를 처리하는 함수 지정
- signal, sigset 함수보다 다양하게 시그널 제어 가능
* 시그널 발생 원인 출력: psiginfo(3)
- 시그널 발생 원인을 출력 : si_code 값이 양수면 시스템에서 시스널 생성

2. 인터벌 타이머
* 알람 시그널의 개념
- 일정한 시간이 지난 후에 자동으로 시그널이 발생하도록 하는 시그널
- 일정 시간 후에 한 번 발생시키거나, 일정 간격을 두고 주기적으로 발송 가능
- 일정 시간이 지나면 SIGALRM 시그널 발생
- 프로세스별로 알람시계가 하나 밖에 없으므로 알람은 하나만 설정 가능

- 알람 시그널 생성: alarm(2)

```
#include <unistd.h>

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

- sec : 알람이 발생시킬 때까지 남은 시간(초 단위)

; 정해진 시간에 한번 시그널 보냄
* 타이머의 종류
- 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)
- - 타이머 설정: setitimer(2)