

# 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 (test1.c)

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); SIGALRM 에 대한 핸들러를 설정.
12
13     alarm(2);
14     printf("Wait...\n");
15     sleep(3); sleep 상태 중 2초가 지나면, 알람 시그널을 받고 핸들러를 수행.
16
17     return 0;
18 }
```

2초 설정

핸들러의 수행이 종료되면, 남은 1초를 기다리지 않고 다음 코드로 넘어감.

# ex7\_9.out

Wait...

Received Signal: Alarm Clock



# 인터벌 타이머

## □ 타이머의 종류

- ITIMER\_REAL : 실제 시간 사용. SIGALRM 시그널 발생
- ITIMER\_VIRTUAL : 프로세스의 가상 시간 사용. SIGVTALRM 시그널 발생  
(프로세스가 동작중인 경우만 작동) = 프로세스가 실행되고 있는 시간
- ITIMER\_PROF : 시스템이 프로세스를 위해 실행중인 시간과 프로세스의 가상 시간을 모두 사용. SIGPROF 시그널 발생 = 프로세스가 실행되고 있는 시간 + 프로세스를 작동시키기 위해 준비하는 시간
- ITIMER\_REALPROF : 실제 시간 사용. 멀티스레드 프로그램의 실제 실행시간 측정시 사용. SIGPROF 시그널 발생  
스레드마다 수행시간이 다름. 스레드의 평균 시간으로 측정할 수도 있고, 맨 처음에 시작한 스레드의 시간에서부터 마지막까지 수행된 스레드의 시간까지 일 수도 있음.

## □ 타이머 정보 검색: getitimer(2)

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

int getitimer(int 타이머 종류 which, struct 시간 간격에 대한 구조체 itimerval *value);
```

## □ 타이머 설정: setitimer(2)

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

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

기존에 있던 시간

# 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;  
};
```

마이크로сек: 1/1,000,000 sec

it\_interval : 간격정보 어떤 간격으로 타이머를 발생시킬 것인가

it\_value : 만료까지 남은 시간. 0이면 기능멈춤



## [예제 7-10] 인터벌 타이머 설정하기 (test2.c)

ex7\_10.c

```
...
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_usec);
32         sleep(1); 1초 슬립
33     }
34
35     return 0;
36 }
```

타이머 간격 : 2초  
타이머에 현재 남은 시간 : 3초

3초 후에 최초 시그널 발생  
이후 2초 간격으로 시그널 발생

interval = 0 이라면,  
3초 후 최초의 시그널을  
발생하고, 기능이 멈추게 됨.

기존에 있던 구조체는 NULL 로 표시.

2.999997 초를 의미.

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.
```

## 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);
```

 인자로 받은 시그널을 블록      시그널을 계속 블록 상태로 둠. 아예 무시.

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

 인자로 받은 시그널을 해제

- 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

아예 민관이 마냥 사람  
같아치우듯이 대체 시켜버림.



```
...
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

블록되어 있는 상태이기 때문에

# ex7\_11.out  
^^C^^C^^C



## [예제 7-12] sigprocmask 함수 사용하기(test4.c)

ex7\_12.c

```
...
05 int main(void) {
06     sigset t new;
07
08     sigemptyset(&new); 일단 비우고
09     sigaddset(&new, SIGINT); 넣고
10     sigaddset(&new, SIGQUIT); 넣고
11     sigprocmask(SIG_BLOCK, &new, (sigset t *)NULL);
12                                     기존 시그셋을 NULL 로 표시
13     printf("Blocking Signals : SIGINT, SIGQUIT\n");
14     printf("Send SIGQUIT\n");
15     kill(getpid(), SIGQUIT);
16     SIGQUIT 이 블록되기 때문에 코어 덤프
17     시킨 다음에 종료가 수행되지 않음.
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

시그널 집합이 블록이 해제되면서 블록되었던  
SIGQUIT 이 여기서 수행됨.  
코어 덤프가 만들어지고 프로세스가 종료됨.

malloc 하면 free 해주는 것처럼,  
블록하고 언블록해줘야 함.

블록해제 후 시그널을 받아  
종료

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

## 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

기다리려는 시그널들을 지정한  
sigset\_t set 에서 O으로 설정된 시그널을 기다림.



## [예제 7-13] sigsuspend 함수 사용하기(test5.c)

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

3초가 지난 후, 알람을 호출.

시그널 기다리기

여기서 3초를 기다림.

알람 시그널을 기다림.

# ex7\_13.out

Wait...

^^Received 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

외울 필요 없음.

*idtype →*

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



## □ 문제1 ~ 문제4

문제 1: 시그널 집합을 이용해 모든 시그널을 집합에 설정하고 이 중 SIGBUS 가 설정되었는지 확인하는 프로그램.

문제 2: 시그널 집합을 정의하고 여기에 SIGQUIT, SIGILL, SIGTRAP 시그널을 설정한 다음에 이 중 SIGILL 시그널이 설정되었는지 확인하는 프로그램. (SUN OS의 경우,) 그리고 시그널 집합에 설정된 비트 패턴을 보고 시그널이 제대로 설정되었는지 확인.

- SIGILL 이 시그널 집합의 멤버임을 알 수 있음.

문제 3: 시그넷 함수를 이용해 SIGQUIT 의 시그널을 붙잡고 이에 대한 처리 함수를 수행하는 프로그램을 작성하라.

문제 4: SIGACTION 을 이용해서 SIGQUIT 의 시그널을 붙잡고 이에 대한 처리 함수를 작성하라. 시그널을 한 번 처리하고 기본 처리 방법으로 바꿔라.  
핸들러를 작성하라는 의미.

- act.sa\_flags = SA\_RESETHAND;

