

Signal part 1

한양대학교 컴퓨터공학과

***Dept. of Computer Science & Engineering
Hanyang University***

Signal

- Signal은 프로세스에게 어떤 사건의 발생을 알릴 때 사용
- 각 신호는 관련 default 동작이 있음 : exit, core, stop, ignore
- 신호이름은 파일 <signal.h>에 매크로로 정의되어 있음
- 목록 확인 방법
 - #) kill -l

```
[TA3@localhost lab9]$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS     8) SIGFPE     9) SIGKILL    10) SIGUSR1
11) SIGSEGV    12) SIGUSR2   13) SIGPIPE   14) SIGALRM   15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD  18) SIGCONT   19) SIGSTOP   20) SIGTSTP
21) SIGTTIN    22) SIGTTOU   23) SIGURG    24) SIGXCPU   25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF  28) SIGWINCH  29) SIGIO      30) SIGPWR
31) SIGSYS     34) SIGRTMIN  35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
[TA3@localhost lab9]$
```

Process 종료

Process의 정상 종료

exit() 시스템 호출 사용

Process의 비정상 종료

abort() 시스템 호출 사용

이 시스템 호출은 자기 프로세스에게 SIGABRT 신호를 보냄

Signal	Description
SIGHUP (1)	터미널을 잃어 버렸을때 발생
SIGINT (2)	Ctrl-C 나 Ctrl-Break 키를 입력했을 때 발생
SIGABRT (6)	프로그램의 비정상 종료 시 발생
SIGKILL (9)	프로세스를 죽이기 위해서
SIGUSR1 (10)	사용자를 위해 정의된 시그널
SIGSEGV (11)	잘못된 메모리 참조에 의해 발생
SIGPIPE (13)	단절된 파이프에 write한 경우 발생
SIGSTOP (19)	프로세스의 일시 중단(Ctrl-Z)시 발생
SIGIO (29)	비 동기적인 입출력이 발생했을 때 발생

Signal 처리

- 프로세스가 signal을 받았을 때의 행동
 - 종료 (exit, abort)
 - 대부분의 signal은 signal 수신 시에 프로세스가 정상 종료 (normal termination)한다.
 - SIGABRT, SIGBUS, SIGSEGV, SIGQUIT, SIGILL, SIGTRAP, SIGSYS, SIGXCPU, SIGXFSZ, SIGFPE 시그널은 비정상종료(abnormal termination)을 발생시킴
 - 시그널 무시 (ignore)
 - 수행 중지 (stop)
 - 사용자 수준 signal 처리 함수 (user level catch function) 수행

Signal 발생

- 터미널에서 특수키를 누르는 경우
 - Ex) Ctrl-C -> SIGINT 발생
- 하드웨어의 오류
 - 0으로 나눈 경우, 잘못된 메모리를 참조하는 경우 등
- kill 함수의 호출
 - 특정 프로세스나 프로세스 그룹에게 원하는 시그널을 발생하거나 전달한다.
 - 대상 프로세스에 대한 권한이 있어야 한다.
- kill 명령의 실행
 - 내부적으로 kill 함수를 호출한다.
- 소프트웨어적조건
 - 네트워크에서의 데이터오류(SIGURG), 파이프 작업에서의 오류(SIGPIPE), 알람 시계의 종료(SIGALRM) 등

Signal set

- sigset_t 를 사용하여 정의
- 사용하고자 하는 시그널의 리스트를 지정
- 시그널을 다루는 시스템 호출에 전달되는 주요 인수 중의 하나

사용법

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

<초기화>

```
int sigemptyset(sigset_t *set);    //빈집합 생성
```

```
int sigfillset(sigset_t *set)      //풀 집합 생성
```

<조작>

```
int sigaddset(sigset_t *set, int signo)    // 추가
```

```
int sigdelset(sigset_t *set, int signo)    // 삭제
```

Example – signal set

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

int main(void )
{
    sigset_t mask1, mask2;

    /* create empty set */
    sigemptyset(&mask1);

    /* add signal */
    sigaddset(&mask1, SIGINT);
    sigaddset(&mask1, SIGQUIT);

    /* create full set */
    sigfillset(&mask2);

    /* remove signal */
    sigdelset(&mask2, SIGCHLD);

    /* check signal */
    if (sigismember(&mask1, SIGINT))
        printf("SIGINT is member of mask1\n");
    else
        printf("SIGINT is not member of mask1\n");

    if (sigismember(&mask2, SIGCHLD))
        printf("SIGCHLD is member of mask2\n");
    else
        printf("SIGCHLD is not member of mask2\n");

    return 0;
}
```

```
[TA3@localhost lab9]$ ./signalset
SIGINT is member of mask1
SIGCHLD is not member of mask2
[TA3@localhost lab9]$
```

sigaction

사용법

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

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

특정 시그널의 수신에 대해서 취할 액션을 설정하거나 변경하기 위해서 사용된다

signo

- 행동을 지정하고자 하는 개개 시그널을 식별
- SIGSTOP, SIGKILL을 제외한 모든 시그널 이름 지정가능

act : signo에 대해 지정하고 싶은 행동

oact: 현재 설정 값을 채운다.

sigaction 구조체

```
struct sigaction{
    void (*sa_handler)(int);    //취해질 행동
    sigset_t sa_mask;          //시그널을 처리하는 동안 추가의 시그널봉쇄
    int sa_flags;               //시그널 형태에 영향을 미칠 플래그들
    void (*sa_sigaction) (int, siginfo_t*, void *);
                                // 시그널 핸들러에 대한포인터
};
```

sa_handler

- signo번호를 가지는 시그널이 발생했을 때 실행될 함수를 설치한다.
- 함수외에도 SIG_DFL과 SIG_IGN을 지정할 수 있다.
- 전자는 시그널에 대한 기본행동을 후자는 시그널을 무시하기 위해서 사용

sa_mask

- sa_handler에 등록된 시그널 핸들러 함수가 실행되는 동안 블락되어야 하는 시그널의 마스크를 제공한다.

sigaction 구조체 (cont'd)

- `sa_flags`

- 시그널 처리 프로세스의 행위를 수정하는 일련의 플래그들을 명시한다.
다음 중 하나 이상의 것들에 의해서 만들어진다.
- `SA_NOCLDSTOP`
만약 `signum`이 `SIGCHLD`라면, 자식프로세스가 `SIGSTOP`, `SIGTSTP`, `SIGTTIN`, `SIGTTOU`등을 받아서 중단되었을 때 이를 통지 받을 수 없게 된다.
- `SA_ONESHOT`, `SA_RESETHAND`
일단 시그널 처리기가 호출되면, 기본 상태에 대한 시그널 액션을 재 저장한다.
이는 `signal(2)` 호출에 대한 기본행위이다.
- `SA_RESTART`
일부 시스템 호출들이 시그널을 통해 재 시작할 수 있도록 함으로서 BSD 시그널과 호환 되도록 한다.
- `SA_NOMASK`, `SA_NODEFER`
시그널 이 자체 시그널 처리기로부터 수신 받지 않도록 한다.
- `SA_SIGINFO`
시그널 처리기가 하나가 아닌 3개의 인자를 취할 경우 `sa_handler` 대신 `sa_sigaction`의 `siginfo_t`를 이용할 수 있다.
`siginfo_t`는 시그널에 대한 추가의 정보를 포함하고있다.

Example – sigaction (SIGINT 처리)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <signal.h>
5
6 void catchint(int signum) {
7     printf("\nCATCHINT: signum = %d\n", signum);
8     printf("CATCHINT: returning\n\n");
9 }
10
11 int main()
12 {
13     static struct sigaction act;
14
15     act.sa_handler = catchint;
16
17     sigfillset(&(act.sa_mask));
18     sigaction(SIGINT, &act, NULL);
19
20     printf("sleep call #1\n");
21     sleep(1);
22     printf("sleep call #2\n");
23     sleep(1);
24     printf("sleep call #3\n");
25     sleep(1);
26     printf("sleep call #4\n");
27     sleep(1);
28     printf("Exiting\n");
29     exit(0);
30 }

```

```

[TA3@localhost lab9]$ ./sigaction
sleep call #1
sleep call #2
^C
CATCHINT: signum = 2
CATCHINT: returning

sleep call #3
sleep call #4
Exiting
[TA3@localhost lab9]$

```

Example - Restoring a previous action

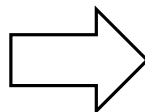
```
#include <stdio.h>
#include <signal.h>

struct sigaction act_new;
struct sigaction act_old;

void sigint_handler(int signo)
{
    printf("Press the Ctrl-C key.\n");
    printf("Press again to exit.\n");
    sigaction(SIGINT, &act_old, &act_new);
}

int main( void)
{
    act_new.sa_handler = sigint_handler;
    sigemptyset(&act_new.sa_mask);

    sigaction(SIGINT, &act_new, &act_old);
    while(1)
    {
        printf("=== action ===\n");
        sleep( 1);
    }
}
```



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

struct sigaction act_new;
struct sigaction act_old;

void sigint_handler(int signo)
{
    printf("Press the Ctrl-C key.\n");
    printf("Press again to exit.\n");
    sigaction(SIGINT, &act_old, &act_new);
}

int main( void)
{
    act_new.sa_handler = sigint_handler;
    sigemptyset(&act_new.sa_mask);
    sigaction(SIGINT, &act_new, &act_old);
    while(1)
    {
        printf("=== action ===\n");
        sleep( 1);
    }
}
```

Example – Signal block

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

void sigint_handler(int signo)
{
    int i;

    for (i = 3; 0 < i; i--){
        printf("%d seconds left.\n", i);
        sleep(1);
    }
}

int main(void)
{
    struct sigaction act;

    act.sa_handler = sigint_handler;
    sigfillset( &act.sa_mask);

    sigaction(SIGINT, &act, NULL);
    while(1)
    {
        printf("=== action ===\n");
        sleep( 1);
    }
}
```

```
[2017107775@mango signal]$ ./signalblock
=== action ===
=== action ===
=== action ===
^C3 seconds left.
2 seconds left.
^Z1 seconds left.
[2]+  Stopped                  ./signalblock
```

Example - Graceful Exit

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <signal.h>
4
5 void g_exit(int s)
6 {
7     unlink("tempfile");
8     fprintf(stderr, "Interrupted - exiting\n");
9     exit(1);
10 }
11
12 int main(void)
13 {
14     /*
15     ...
16     */
17
18     static struct sigaction act;
19     act.sa_handler = g_exit;
20     sigaction(SIGINT, &act, NULL);
21
22     /*
23     ...
24     */
25
26 }
```

Example - Signal

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <signal.h>
4 #include <sys/types.h>
5
6 static void sig_usr(int signum)
7 {
8     if (signum == SIGUSR1)
9         printf("received SIGUSR1\n");
10    else if (signum == SIGUSR2)
11        printf("received SIGUSR2\n");
12    else {
13        fprintf(stderr, "received signal %d\n", signum);
14        fflush(stderr);
15        abort();
16    }
17    return;
18 }
19
20 int main(void)
21 {
22     if (signal(SIGUSR1, sig_usr) == SIG_ERR) {
23         perror("can't catch SIGUSR1");
24         exit(1);
25     }
26     if (signal(SIGUSR2, sig_usr) == SIG_ERR) {
27         perror("can't catch SIGUSR2");
28         exit(1);
29     }
30
31     for (;;)
32         pause();
33 }

```

```

[TA3@localhost lab9]$ ps
  PID TTY          TIME CMD
 3170 pts/2    00:00:00 bash
 3411 pts/2    00:00:00 ps
[TA3@localhost lab9]$ ./signal &
[1] 3412
[TA3@localhost lab9]$ ps
  PID TTY          TIME CMD
 3170 pts/2    00:00:00 bash
 3412 pts/2    00:00:00 signal
 3413 pts/2    00:00:00 ps
[TA3@localhost lab9]$ kill -USR1 3412
received SIGUSR1
[TA3@localhost lab9]$ kill -USR2 3412
received SIGUSR2
[TA3@localhost lab9]$ kill 3412
[1]+  종료됨                  ./signal
[TA3@localhost lab9]$

```

Q & A

- Thank you :)