

# UNIX 시스템 프로그래밍

» 7장. 시그널

# signal

- ▶ signal :
  - software interrupt
  - kernel → process or process → process
  - 자료 전송보다는 비정상적인 상황을 알릴 때 사용
  - 예: program 수행 중 Ctrl-C (interrupt key)
    - kernel이 문자 감지, 해당 session에 있는 모든 process에게 "SIGINT"라는 signal을 보냄;
    - 모든 process는 종료! 그러나, shell process는 무시!

# signal (2)

- ▶ signal은 <signal.h>에 정의 (page 292, 표7-7 참조)
- ▶ signal의 기본 처리
  - 종료 (signal에 의한 정상 종료)
  - 코어덤프 후 종료 (signal에 의한 비정상 종료)
    - core file (종료 직전의 memory의 상태) 생성 후 종료
  - 중지
  - 무시

# child process의 종료 상태 확인

- ▶ child process의 종료 상태 확인 :

```
pid=wait(&status);
```

```
if (WIFEXITED(status))
```

“정상종료”

```
    printf("%d\n", WEXITSTATUS(status));
```

```
if (WIFSIGNALED(status))
```

“signal을 받고 종료”

```
    printf("%d\n", WTERMSIG(status));
```

# signal 보내기

## ▶ 사용법 :

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

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

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

- pid : signal을 받게될 process 지정;
- sig : 보낼 signal 지정;

# signal 보내기 (2)

- ▶ signal을 받을 process 또는 process group 지정
  - $pid > 0$  : 해당 id의 process에게 signal 전달
  - $pid = 0$  : sender와 같은 process group에 속하는 모든 process에게 signal 전달. sender 자신 포함;
  - $pid = -1$  : uid가 sender의 euid와 같은 모든 process에게 signal 전달. sender 자신 포함;
  - $pid < 0$  &  $pid \neq -1$  : process의 group id가 pid의 절대값과 같은 모든 process에게 signal 전달
- ▶ 다른 사용자의 process에게 signal을 보내면 -1 return;

# signal 보내기 (3)

- ▶ 사용법 :

  - `#include <signal.h>`

  - `int raise(int sig);`

  - 호출 process에게 sig를 보낸다.

# signal handling

- ▶ signal Handling :
  - default action (프로세스 종료)
  - 무시
  - 정의된 action;



# signal handling (2)

- ▶ sigaction 지정 : signal 수신 시 원하는 행동을 취할 수 있도록 한다.
  - 예외) SIGSTOP(process의 일시 중단), SIGKILL(process의 종료)의 경우는 별도의 action을 취할 수 없다.
- ▶ 지정 방법:

```
#include <signal.h>
int sigaction(int signo, const struct sigaction *act,
struct sigaction *oact);
```

# signal handling (3)

## ▶ sigaction의 구조 :

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

# sigaction의 구조

- ▶ `void (*sa_handler) (int);`
  - `signo`를 수신하면 취할 행동 지정;
    - `SIG_DFL` (default 행동, 즉 종료 선택);
    - `SIG_IGN` (무시);
    - 정의된 함수 (`signal`을 받으면 함수로 제어 이동; 함수 실행 후, `signal`을 받기 직전의 처리 문장으로 `return`);

# sigaction의 구조 (2)

- ▶ `sigset_t sa_mask;`
  - 여기 정의된 signal들은, `sa_handler`에 의해 지정된 함수가 수행되는 동안 blocking된다.
- ▶ `int sa_flags;`
  - `SA_RESETHAND` : handler로부터 복귀 시 signal action을 `SIG_DFL`로 재설정;
  - `SA_SIGINFO` : `sa_handler` 대신 `sa_sigaction` 사용

# signal 사용 예

```
#include <signal.h>
main(){
    static struct sigaction act;
    void catchint(int);

    act.sa_handler=catchint;
    sigaction(SIGINT, &act, NULL);

    printf("sleep call1\n");
    sleep(1);
    printf("sleep call2\n");
    sleep(1);
    printf("exiting\n");
    exit(0);
}
void catchint(int signo) {
    printf("\n CATCHINT: signo=%d\n", signo);
}
```

# signal 사용 예 (2)

- ▶ SIGINT를 무시 :

```
act.sa_handler=SIG_IGN;  
sigaction(SIGINT, &act, NULL);
```

- ▶ SIGINT시 종료:

```
act.sa_handler=SIG_DFL;  
sigaction(SIGINT, &act, NULL);
```

# signal 사용 예 (3)

- ▶ 여러개의 signal을 무시하려면:  
act.sa\_handler=SIG\_IGN;  
sigaction(SIGINT, &act, NULL);  
sigaction(SIGQUIT, &act, NULL);
- ▶ 한 process에서 무시되는 signal은 exec()후에도 계속 무시된다.

# signal 집합 지정

- ▶ signal 집합 지정 :
  - sigemptyset → sigaddset, 또는
  - sigfillset → sigdelset
- ▶ 사용 방법 :

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);

int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);

int sigismember(sigset_t *set, int signo);
```



# signal 집합 지정 (2)

▶ 예:

```
sigset_t mask1, mask2;
```

```
sigemptyset(&mask1);
```

```
sigaddset(&mask1, SIGINT);
```

```
sigaddset(&mask1, SIGQUIT);
```

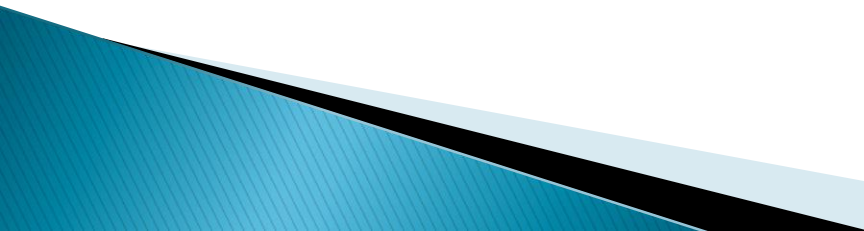
```
sigfillset(&mask2);
```

```
sigdelset(&mask2, SIGCHLD);
```

# sa\_sigaction()에 의한 signal handling

```
int main(void){
    static struct sigaction act;
    act.sa_flags=SA_SIGINFO;
    act.sa_sigaction=handler;
    sigaction(SIGUSR1, &act, NULL);
    ...
}

void handler(int signo, siginfo_t *sf, ucontext_t *uc){
    psiginfo(sf, "...:");
    printf("%d\n", sf->si_code);    <page 313, 표7-8 참조>
}
```



# 이전의 설정 복원하기

## ▶ 이전의 설정 복원하기 :

```
sigaction(SIGTERM, NULL, &oact); /* 과거 설정 저장 */
```

```
act.sa_handler=SIG_IGN;
```

```
sigaction(SIGTERM, &act, NULL);
```

```
// do anything;
```

```
sigaction(SIGTERM, &oact, NULL);
```

# alarm signal 설정

## ▶ timer 사용 :

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

```
unsigned int alarm(unsigned int secs);
```

- secs : 초 단위의 시간; 시간 종료 후 SIGALRM을 보낸다;
- alarm은 exec 후에도 계속 작동; but fork 후에는 자식 process에 대한 alarm은 작동하지 않는다.
- alarm(0); ---> alarm 끄기;
- alarm은 누적되지 않는다. 2번 사용되면, 두 번째 alarm이 대체;
- 두 번째 alarm의 return 값이 첫 alarm의 잔여 시간;

# signal blocking

## ▶ 사용법 :

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

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

- how := SIG\_SETMASK : set에 있는 signal들을 지금부터 봉쇄;
- oset은 봉쇄된 signal들의 현재 mask; 관심 없으면 NULL로 지정;
- how := SIG\_UNBLOCK : 봉쇄 제거;

# pause 시스템 호출

## ▶ 사용법 :

```
#include <unistd.h>
```

```
int pause(void);
```

- signal 도착까지 실행을 일시 중단 (CPU 사용 없이);
- signal이 포착되면; 처리 routine 수행 & -1 return;