System Programming

Exercise

Signal

시그널 (signal)

- 시그널은 프로세스에게 어떤 사건의 발생을 알릴 때 사용
- 시그널 이름은 파일 <signal.h>에 매크로로 정의되어 있음

```
user@user-VirtualBox:~$ 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+237) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+742) 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-857) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-362) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

- 각 시그널에는 기본 동작이 있음
  - exit, core dump, stop, ignore

프로세스 종료와 시그널 (1/2)

프로세스의 정상 종료
exit() 시스템 호출 사용
프로세스의 비정상 종료
abort() 시스템 호출 사용
이 시스템 호출은 자기 프로세스에게 SIGABRT 시그널을 보냄

이름	signo	설명	기본 동작
SIGABORT	6	비정상 종료 (abort)	종료+코어
SIGALRM	14	타이머 만료 (alarm)	종료
SIGBUS	7	하드웨어 고장	종료+코어
SIGCHLD	17	자식 상태 변경	무시
SIGCONT	18	정지된 프로세스 재개	재개 또는 무시
SIGFPE	8	산술 연산 예외처리	종료+코어
SIGHUP	1	제어 터미널 신호 끊김	종료
SIGILL	4	적법하지 않은 명령	종료+코어
SIGINT	2	터미널 가로채기 문자 (ctrl + c)	종료
SIGIO	29	비동기 입출력	종료 또는 무시
SIGKILL	9	Catch 또는 무시할 수 없는 종료	종료
SIGPIPE	13	판독자 없는 파이프에 쓰기	종료
SIGPROF	27	프로파일링 시간 경보 (setitimer)	종료
SIGPWR	30	파워 고장 또는 재시작	종료 또는 무시

- 시그널 개요
- 프로세스 종료
- 시그널 처리
- 시그널 집합
- sigaction
- sigprocmask
- kill
- raise/alarm
- pause
- sigsetjmp/ siglongjmp

프로세스 종료와 시그널 (2/2)

이름	signo	설명	기본 동작
SIGQUIT	3	터미널 중지 문자 (ctrl + \)	종료+코어
SIGSEGV	11	유효하지 않은 메모리 참조	종료+코어
SIGSTKFLT	16	보조 프로세스 스택 고장	종료
SIGSTOP	19	Catch 또는 무시할 수 없는 정지	프로세스 정지
SIGSYS	31	유효하지 않은 시스템 호출	종료+코어
SIGTERM	15	kill(1) 명령의 기본 종료	종료
SIGTRAP	5	하드웨어 고장	종료+코어
SIGTSTP	20	터미널 정지 문자 (ctrl + z)	프로세스 정지
SIGTTIN	21	제어 터미널에서 배경 읽기	프로세스 정지
SIGTTOU	22	배경에서 제어터미널에 쓰기	프로세스 정지
SIGURG	23	긴급 조건 (소켓)	무시
SIGUSR1	10	사용자 정의 신호	종료
SIGUSR2	12	사용자 정의 신호	종료
SIGVTALRM	26	가상 시간 경보 (setitimer)	종료
SIGWINCH	28	터미널 창 크기 변경	무시
SIGXCPU	24	CPU 한계 초과 (setrlimit)	종료(+코어)
SIGXFSZ	25	파일 크기 한계 초과 (setrlimit)	종료(+코어)

시그널 처리

- 프로세스가 시그널을 받았을 때의 행동은 다음 셋 중 하나
  - SIG_DFL: 기본 동작
  - SIG_IGN: 시그널을 통째로 무시
  - 사용자 지정 동작
- 시그널 집합(signal set)
  - 시그널을 다루는 시스템 콜의 주요 인수 중 하나
  - sigset_t 타입을 사용하여 시그널 집합을 정의

```
#include <signal.h>
// 초기화
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
// 조작
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
```

시그널 집합 만들기

```
sigset_t set1, set2;

//빈집합생성
sigemptyset(&set1);

//시그널추가
sigaddset(&set1, SIGINT);
sigaddset(&set1, SIGQUIT);

//완전히차있는집합생성
sigfillset(&set2);

//시그널제거
sigdelset(&set2, SIGCHLD);
```

sigaction()

```
#include <signal.h>

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

- 특정 시그널을 받았을 때 취할 동작을 설정하거나 변경한다.
  - signo
    행동을 지정하려는 시그널 식별자(혹은 시그널 번호)로서,
    SIGSTOP과 SIGKILL을 제외한 모든 시그널 지정 가능
  - act
    signo를 받았을 때 취할 행동을 지정
  - oact
    현재 설정되어 있는 행동을 돌려 받음

sigaction 구조체

```
struct sigaction{
  void (*sa_handler)(int);        // 취해질 행동
  sigset_t sa_mask;               // signal을 처리하는 동안 추가의 signal 봉쇄
  int sa_flags;                   // signal 형태에 영향을 미칠 플래그들
  void (*sa_sigaction) (int, siginfo_t*, void *); // signal 핸들러에 대한 포인터
};
```

- sa_handler
  - signo번호를 가지는 시그널이 발생했을 때 실행될 함수를 설정
  - 함수 외에도 SIG_DFL과 SIG_IGN을 지정할 수 있음
    - SIG_DFL: signal에 대한 기본 행동으로 설정
    - SIG_IGN: signal을 무시하도록 설정
- sa_mask
  - sa_handler에 등록된 시그널 핸들러 함수가 실행되는 동안 블록되어야 하는 시그널 집합

sigaction 구조체 (1/2)

- sa_flags: 시그널 처리 동작을 관리하는 플래그
  - **SA_INTERRUPT:** 이 시그널에 의해 가로채인 시스템 콜이 자동으로 재시작 되지 않음
  - **SA_NOCLDSTOP:** signo가 SIGCHLD인 경우, 자식 프로세스가 정지되었을 때 그에 대한 시그널을 전달하지 않음
  - **SA_NOCLDWAIT:** signo가 SIGCHLD인 경우, 자식 프로세스가 종료되었을 때 시스템이 좀비 프로세스를 만들지 못하게 함.
    이후 wait 류 함수를 호출하면 모든 자식이 종료될 때까지 부모 프로세스가 차단되며, erro를 ECHILD로 설정하고 -1을 반환함
  - **SA_NODEFER:** 시그널에 대한 처리 함수가 실행되는 동안 동일 시그널을 자동으로 차단하지 않음. (sa_mask와 별개)
  - **SA_RESETHAND:** 시그널 처리함수에 진입할 때, 해당 시그널의 처리 방법을 SIG_DFL로 재설정하고 SA_SIGINFO 플래그를 지움
  - **SA_RESTART**: 이 시그널에 의해 가로채인 시스템콜이 자동으로 재시작 됨

sigaction 구조체 (2/2)

- sa_flags: 시그널 처리 시 프로세스의 동작을 관리하는 플래그
  - **SA_SIGINFO**: sa_handler 대신 sa_sigaction을 이용하여 시그널 핸들러를 설정함. 이때 siginfo_t는 시그널에 대한 추가 정보를 포함함

```
struct siginfo {
  int           si_signo; /* 시그널 번호 */
  int           si_errno; /* 0이 아니면 errno.h에 정의된 errno 값 */
  int           si_code;  /* 시그널 종류에 따른 추가 정보 */
  pid_t         si_pid;   /* 시그널을 보낸 프로세스의 ID */
  uid_t         si_uid;   /* 시그널을 보낸 프로세스의 real user ID */
  void          *si_addr; /* 오류를 일으킨  주소 */
  int           si_status;/* 종료 상태 또는 시그널 번호 */
  union sigval  si_value; /* 응용 프로그램 고유 값 */
  /* … 그 외의 필드들도 존재할 수 있음 … */
};
```

- 시그널 개요
- 프로세스 종료
- 시그널 처리
- 시그널 집합
- sigaction
- sigprocmask
- kill
- raise/alarm
- pause
- sigsetjmp/ siglongjmp

Example – catch SIGINT (1/2)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

int count;

// 시그널 핸들러
void catch_sigint(int signum) {
  printf("\n(count=%d) CTRL-C pressed!\n",count);
  return;
}

int main(int argc, char *argv[]) {
  struct sigaction act;
  sigset_t masksets;
  int i;
  char buf[10];
  sigfillset(&masksets);
```

Example – catch SIGINT (2/2)

```
  // 시그널 핸들러 설치
  act.sa_handler = catch_sigint;

  // 시그널 핸들러가 실행되는 동안 모든 시그널을 블록함
  act.sa_mask = masksets;
  act.sa_flags = 0;
  sigaction(SIGINT, &act, NULL);

  for(count=0; count<3 ; count++ )
    read(0, buf, sizeof(buf));
  return 0;
}
```

```
^C
(count=0) CTRL-C pressed!
^C
(count=1) CTRL-C pressed!
^C
(count=2) CTRL-C pressed!
```

시그널 동작 복구

```
#include <signal.h>

int main() {
          static struct sigaction act, oact;
          sigaction(SIGINT, NULL, &oact);  /* save old sigaction*/
          act.sa_handler=SIG_IGN;
          sigaction(SIGINT, &act, NULL);   /* set new sigaction*/
          ...
          sigaction(SIGINT, &oact, NULL);  /* restore old sigaction*/
          return 0;
}
```

- 시그널 개요
- 프로세스 종료
- 시그널 처리
- 시그널 집합
- sigaction
- sigprocmask
- kill
- raise/alarm
- pause
- sigsetjmp/ siglongjmp

Example – Catch ball (1/3)

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

int ntimes = 0;

int main() {
    pid_t pid, ppid;
    void p_action (int), c_action (int);
    static struct sigaction pact, cact;

    /* 부모를 위해 SIGUSR1 행동을 지정한다. */
    pact.sa_handler = p_action;
    sigaction (SIGUSR1, &pact, NULL);

    switch (pid = fork()){
        case -1: /* 오류 */
        perror ("synchro");
        exit (1);
```

Example – Catch ball (2/3)

```
case 0: /* 자식 */
/* 자식을 위해 행동을 지정 */
cact.sa_handler = c_action;
sigaction (SIGUSR1, &cact, NULL);
/* 부모의 프로세스 식별번호를 얻음. */
ppid = getppid();

for (;;) {
    sleep (1);
    kill (ppid, SIGUSR1);
    pause();
}
/* 결코 퇴장(exit) 않음. */
```

- 시그널 개요
- 프로세스 종료
- 시그널 처리
- 시그널 집합
- sigaction
- sigprocmask
- kill
- raise/alarm
- pause
- sigsetjmp/ siglongjmp

Example – Catch ball (3/3)

```
        default: /* 부모 */
        for(;;) {
            pause();
            sleep (1);
            kill (pid, SIGUSR1);
        }
        /* 결코 퇴장(exit) 않음 */
    }
}

void p_action (int sig) {
    printf ("Parent caught signal #%d\n", ++ntimes);
}

void c_action (int sig) {
    printf ("Child caught signal #%d\n", ++ntimes);
}
```

```
Parent caught signal #1
Child caught signal #1
Parent caught signal #2
Child caught signal #2
Parent caught signal #3
Child caught signal #3
Parent caught signal #4
Child caught signal #4
^\끝내기 (코어 덤프됨)
```

- 시그널 개요
- 프로세스 종료
- 시그널 처리
- 시그널 집합
- sigaction
- sigprocmask
- kill
- raise/alarm
- pause
- sigsetjmp/ siglongjmp

Example – sigint&quit (1/2)

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void errExit(const char *str){
    perror(str);
    exit(EXIT_FAILURE);
}

static void sigHandler(int sig) {
    static int count = 0;
    if (sig == SIGINT) {
        count++;
        printf("Caught SIGINT (%d)\n", count);
        return;
    }

    printf("Caught SIGQUIT!\n");
    exit(EXIT_SUCCESS);
}
```

Example – sigint&quit (2/2)

```c
int main(void) {
    if (signal(SIGINT, sigHandler) == SIG_ERR)
        errExit("signal");
    if (signal(SIGQUIT, sigHandler) == SIG_ERR)
        errExit("signal");
    for (;;) /* Loop forever, waiting for signals */
        pause(); /* Block until a signal is caught */
    return 0;
}
```

```
^CCaught SIGINT (1)
^CCaught SIGINT (2)
^CCaught SIGINT (3)
^CCaught SIGINT (4)
^\Caught SIGQUIT!
```

sigprocmask

```
#include <signal.h>

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

- 블록 할 시그널 집합을 변경하기 위해 사용한다.
- 호출 결과는 how 값에 따라서 달라진다.
  - SIG_BLOCK
    set에 설정된 시그널 집합을 블록 할 시그널 집합에 추가한다.
  - SIG_UNBLOCK
    set에 설정된 시그널 집합을 블록 할 시그널 집합에서 삭제한다.
  - SIG_SETMASK
    set에 설정된 시그널 집합만 블록한다.
- oset이 NULL이 아니면, 현재 시그널 마스크를 oset에 저장한다.

kill

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

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

- pid> 0: pid를 가진 프로세스에게 시그널을 보낸다.
- pid== 0: 시그널을 보내는 프로세스와 같은 프로세스 그룹에 속하는 모든 프로세스에게 시그널을 보낸다.
- pid< -1: 프로세스의 group ID가 pid의 절대값과 같은 모든 프로세스에게 시그널을 보낸다.
- pid== -1
  - non superuser: 시그널을 보내는 프로세스의 Effective user ID와 동일한 real user ID를 갖는 모든 프로세스에 시그널을 보낸다.
  - superuser: 특수한 시스템 프로세스를 제외한 모든 프로세스에게 시그널을 보낸다. (Effective user ID가 root인 경우도 포함)

raise/alarm

```
#include <signal.h>
int raise (int sig);
unsigned int alarm (unsigned int secs);
```

- raise
  - 현재 프로세스에게 sig에 해당하는 시그널을 보낸다 .
- alarm
  - alarm은 secs초 후에 프로세스에 SIGALRM을 보낸다.
  - alarm(0) 을 호출하면 알람이 꺼진다.
  - return
    - 이전에 설정한 알람이 있는 경우, 알람이 시그널을 전달할 때까지 남은 시간을 초 단위 숫자로 반환
    - 이전에 설정한 알람이 없을 경우, 0을 반환

pause

```
#include <unistd.h>
int pause (void);
```

- 시그널을 받을 때까지 호출한 프로세스를 중지시킨다.

```
#include <unistd.h>
#include <signal.h>
void sig_handler(int signo) {
        printf("SIGINT\n");
}
int main() {
        printf("hello world!\n");
        signal(SIGINT, (void *)sig_handler);
        pause();
        printf("Interrupted\n");
}
```

- 시그널 개요
- 프로세스 종료
- 시그널 처리
- 시그널 집합
- sigaction
- sigprocmask
- kill
- raise/alarm
- pause
- sigsetjmp/
  siglongjmp

Example – myalarm

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

void myalarm() {
  printf("ding dong dang\n");
}

int main() {

  int i=0;

  printf("alarm setting\n");
  // SIGALRM 이 발생하면 myalarm() 함수를 실행한다.
  signal(SIGALRM, myalarm);

  // 알람을 1초로 설정한다.
  alarm(1);
  while(i<5) {
    printf("ok\n");
    // 신호를 기다린다.
    pause();
    // alarm 을 2초로 설정한다.
    alarm(2);
    i++;
  }
}
```

```
alarm setting
ok
ding dong dang
ok
ding dong dang
ok
ding dong dang
ok
ding dong dang
ok
ding dong dang
```

sigsetjmp / siglongjmp

```
#include <setjmp.h>
int sigsetjmp(sigjmp_buf env, int savemask);
int siglongjmp(sigjmp_buf env, int val);
```

- sigsetjmp
  - savemask의 값이 0이 아닐 경우, 시그널 마스크 정보를 env에 저장
- siglongjmp
  - env에서 시그널 마스크를 복원하고 val을 반환

```
void goback(void) {
      siglongjmp(position, 1);
}
int main() {
      ...
      if(sigsetjmp(position, 1)==0){       /* saved position */
               act.sa_handler=goback;
               sigaction(SIGINT,&act,NULL);
               …
      }
}
```

Example – [sig]setjmp (1/4)

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <setjmp.h>
#include <signal.h>

#define USE_SIGSETJMP
sig_atomic_t canJump = 0;

#ifdef USE_SIGSETJMP
    sigjmp_buf senv;
#else
    jmp_buf env;
#endif

void errExit(const char* str){
    perror(str);
    exit(EXIT_FAILURE);
}
```

Example – [sig]setjmp (2/4)

```
void printSigset(FILE *of, const char *prefix, const sigset_t *sigset) {
    int sig, cnt;
    cnt = 0;
    for (sig = 1; sig < NSIG; sig++) {
        if (sigismember(sigset, sig)) {
            cnt++;
            fprintf(of, "%s%d (%s)\n", prefix, sig, strsignal(sig));
        }
    }

    if (cnt == 0)
    fprintf(of, "%s<empty signal set>\n", prefix);
}

int printSigMask(FILE *of, const char *msg) {
    sigset_t currMask;

    if (msg != NULL)
        fprintf(of, "%s", msg);

    if (sigprocmask(SIG_BLOCK, NULL, &currMask) == -1)
        return -1;

    printSigset(of, "\t\t", &currMask);
    return 0;
}
```

- 시그널 개요
- 프로세스 종료
- 시그널 처리
- 시그널 집합
- sigaction
- sigprocmask
- kill
- raise/alarm
- pause
- sigsetjmp/ siglongjmp

Example – [sig]setjmp (3/4)

```
static void handler(int sig) {

    printf("Received signal %d (%s), signal mask is:\n", sig, strsignal(sig));
    printSigMask(stdout, NULL);

    if (!canJump) {
        printf("'env' buffer not yet set, doing a simple return\n");
        return;
    }

#ifdef USE_SIGSETJMP
    siglongjmp(senv, 1);
#else
    longjmp(env, 1);
#endif
}
```

- 시그널 개요
- 프로세스 종료
- 시그널 처리
- 시그널 집합
- sigaction
- sigprocmask
- kill
- raise/alarm
- pause
- sigsetjmp/ siglongjmp

Example – [sig]setjmp (4/4)

```
int main(int argc, char *argv[]) {
    struct sigaction sa;

    printSigMask(stdout, "Signal mask at startup:\n");

    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sa.sa_handler = handler;
    if (sigaction(SIGINT, &sa, NULL) == -1)
        errExit("sigaction");

#ifdef USE_SIGSETJMP
    printf("Calling sigsetjmp()\n");
    if (sigsetjmp(senv, 1) == 0)
#else
    printf("Calling setjmp()\n");
    if (setjmp(env) == 0)
#endif
        canJump = 1; /* Executed after [sig]setjmp() */

    else /* Executed after [sig]longjmp() */
        printSigMask(stdout, "After jump from handler, signal mask is:\n" );

    for (;;) /* Wait for signals until killed */
        pause();
    return 0;
}
```

실습문제

- 기본 문제 #1: hello_again
  - 프로그램이 ctrl + c로 종료되지 않도록 만드시오.
  - 힌트: sigprocmask를 사용할 것

- 기본 문제 #2
  - SIGALRM을 받으면 1초 간격으로 "wake up\n"을 출력하는 프로그램을 작성하시오.
  - SIGALRM 시그널의 핸들러가 실행되는 동안 SIGINT를 제외한 모든 시그널을 블록하시오.

THANK YOU

Presented by Hasoo Eun