11장 시그널

11.1 시그널

시그널

- 시그널은 예기치 않은 사건이 발생할 때 이를 알리는 소프트웨어 인터럽트이다.
- 시그널 발생 예
  - SIGFPE 부동소수점 오류
  - SIGPWR 정전
  - SIGALRM 알람시계 울림
  - SIGCHLD 자식 프로세스 종료
  - SIGINT 키보드로부터 종료 요청 (Ctrl-C)
  - SIGSTP 키보드로부터 정지 요청 (Ctrl-Z)

주요 시그널

시그널 이름	의미	기본 처리
SIGABRT	abort()에서 발생되는 종료 시그널	종료(코어 덤프)
SIGALRM	자명종 시계 alarm() 울림 때 발생하는 알람 시그널	종료
SIGCHLD	프로세스의 종료 혹은 정지를 부모에게 알리는 시그널	무시
SIGCONT	정지된 프로세스를 계속시키는 시그널	무시
SIGFPE	0으로 나누기와 같은 심각한 산술 오류	종료(코어 덤프)
SIGHUP	연결 끊김	종료
SIGILL	잘못된 하드웨어 명령어 수행	종료(코어 덤프)
SIGIO	비동기화 I/O 이벤트 알림	종료
SIGINT	터미널에서 Ctrl-C 할 때 발생하는 인터럽트 시그널	종료
SIGKILL	잡을 수 없는 프로세스 종료시키는 시그널	종료
SIGPIPE	파이프에 쓰려는데 리더가 없을 때	종료
SIGPIPE	끊어진 파이프	종료

주요 시그널

SIGPWR	전원고장	종료
SIGSEGV	유효하지 않은 메모리 참조	종료(코어 덤프)
SIGSTOP	프로세스 정지 시그널	정지
SIGTSTP	터미널에서 Ctrl-Z 할 때 발생하는 정지 시그널	정지
SIGSYS	유효하지 않은 시스템 호출	종료(코어 덤프)
SIGTERM	잡을 수 있는 프로세스 종료 시그널	종료
SIGTTIN	후면 프로세스가 제어 터미널을 읽기	정지
SIGTTOU	후면 프로세스가 제어 터미널에 쓰기	정지
SIGUSR1	사용자 정의 시그널	종료
SIGUSR2	사용자 정의 시그널	종료

시그널 생성

- 터미널에서 생성된 시그널
  - CTRL-C → SIGINT
  - CTRL-Z → SIGSTP
- 하드웨어 예외가 생성하는 시그널
  - 0으로 나누기 → SIGFPE
  - 유효하지 않는 메모리 참조 → SIGSEGV
- `kill()` 시스템 호출
  - 프로세스(그룹)에 시그널 보내는 시스템 호출
  - 프로세스의 소유자이거나 슈퍼유저이어야 한다.
- 소프트웨어 조건
  - SIGALRM: 알람 시계 울림
  - SIGPIPE: 끊어진 파이프
  - SIGCHLD: 자식 프로세스가 끝났을 때 부모에 전달되는 시그널

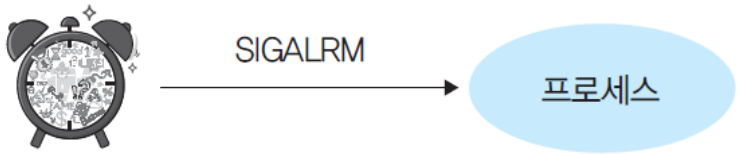
시그널 처리

- 기본 처리 동작
  - 프로세스를 종료시킨다(terminate)
  - 시그널 무시(ignore)
  - 프로세스 정지(suspend)
  - 프로세스 계속(resume)

alarm()

```
#include <unistd.h>
unsigned int alarm(unsigned int sec)
sec초 후에 프로세스에 SIGALRM 시그널이 발생되도록 설정한다.
```

- sec초 후에 프로세스에 SIGALRM 시그널이 발생한다.
  - 이 시그널을 받으면 "자명종 시계" 메시지를 출력하고 프로그램은 종료된다.



- 한 프로세스 당 오직 하나의 알람만 설정할 수 있다.
  - 이전에 설정된 알람이 있으면 취소되고 남은 시간(초)을 반환한다.
  - 이전에 설정된 알람이 없다면 0을 반환한다.

- alarm(0)
  - 이전에 설정된 알람은 취소된다.

alarm.c

```
#include <stdio.h>
/* 알람 시그널을 보여주는 프로그램 */
int main( )
{
    int sec = 0;
    alarm(5);
    printf("무한 루프 ₩n");
    while (1) {
        sleep(1);
        printf("%d초 경과 ₩n", ++sec);
    }
    printf("실행되지 않음 ₩n");
}
```

```
$ alarm
무한 루프
1초 경과
2초 경과
3초 경과
4초 경과
자명종 시계
```

11.2 시그널 처리

시그널 처리기

- 시그널에 대한 처리함수 지정
  - signal() 시스템 호출
  - "이 시그널이 발생하면 이렇게 처리하라"
- signal() 시스템 호출

```
#include <signal.h>
signal(int signo, void (*func)(int)))
signo에 대한 처리 함수를 func으로 지정하고 기존의 처리함수를 리턴한다
```

  - 시그널 처리 함수 func
    - SIG_IGN : 시그널 무시
    - SIG_DFL : 기본 처리
    - 사용자 정의 함수 이름

예제: almhandler.c

```
#include <stdio.h>
#include <signal.h>
void alarmHandler();
/* 알람 시그널을 처리한다. */
int main( )
{
    int sec = 0;
    signal(SIGALRM, alarmHandler);
    alarm(5);     /* 알람 시간 설정 */
    printf("무한 루프 ₩n");
    while (1) {
        sleep(1);
        printf("%d초 경과 ₩n", ++sec);
    }
    printf("실행되지 않음 ₩n");
}
```

```
void alarmHandler(int signo)
{
    printf("일어나세요₩n");
    exit(0);
}
```

```
$almhandler
무한 루프
1초 경과
2초 경과
3초 경과
4초 경과
일어나세요
```

예제: sigint1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
void intHandler();
/* 인터럽트 시그널을 처리한다. */
int main( )
{
   signal(SIGINT,intHandler);
   while (1)
      pause();
   printf("실행되지 않음 ₩n");
}
```

```
void intHandler(int signo)
{
   printf("인터럽트 시그널 처리₩n");
   printf("시그널 번호: %d₩n", signo);
   exit(0);
}
```

```
$sigint1
^C인터럽트 시그널 처리
시그널 번호: 2
```

```
#include <signal.h>
pause()
   시그널을 받을 때까지 해당 프로세스를 잠들게 만든다.
```

sigaction() 함수

- signal()보다 정교하게 시그널 처리기를 등록하기 위한 함수
  - sigaction 구조체를 사용하여 정교한 시그널 처리 액션을 등록

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

signum 시그널(SIGKILL과 SIGSTOP 제외)이 수신되었을 때, 프로세스가 취할 액션을 변경하는 데 사용된다. 이 시그널에 대한 새로운 액션은 act가 되며, 기존의 액션은 oldact에 저장된다. 성공하면 0을 실패하면 –1를 반환한다.

```
struct sigaction {
        void (*sa_handler)(int);            // 시그널 처리기
        void (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t sa_mask;                   // 시그널을 처리하는 동안 차단할 시그널 집합
        int sa_flags;
}
```

sigint2.c

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

struct sigaction newact;
struct sigaction oldact;
void sigint_handler(int signo);

int main( void)
{
      newact.sa_handler = sigint_handler; // 시그널 처리기 지정
      sigfillset(&newact.sa_mask); // 모든 시그널을 차단하도록 마스크

    // SIGINT의 처리 액션을 새로 지정, oldact에 기존 처리 액션을 저장
    sigaction(SIGINT, &newact, &oldact);
```

sigint2.c

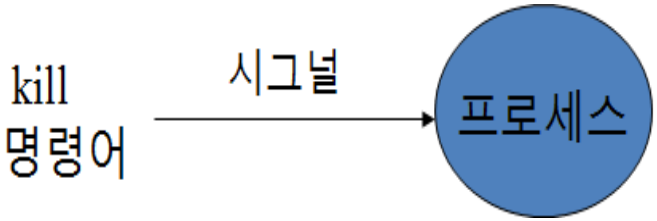
```
    while(1 ) {
        printf( "Ctrl-C를 눌러 보세요 !₩n");
        sleep(1);
    }
}

/* SIGINT 처리 함수 */
void sigint_handler(int signo)
{
    printf( "%d 번 시그널 처리!₩n", signo);
    printf( "또 누르면 종료됩니다.₩n");
    sigaction(SIGINT, &oldact, NULL); // 기존 처리 액션으로 변경
}
```

```
$ sigint2
Ctrl-C 를 눌러 보세요 !
^C2 번 시그널 처리!
또 누르면 종료됩니다.
Ctrl-C 를 눌러 보세요 !
^C
```

11.3 시그널 보내기

시그널 보내기: kill 명령어

- killl 명령어
  - 한 프로세스가 다른 프로세스를 제어하기 위해 특정 프로세스에 임의의 시그널을 강제적으로 보낸다.



- $ kill [-시그널] 프로세스ID

- $ kill –l // 시그널 리스트

  HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH POLL PWR SYS ...

시그널 보내기: kill()

- kill() 시스템 호출
  - 특정 프로세스 pid에 원하는 임의의 시그널 signo를 보낸다.
  - 보내는 프로세스의 소유자가 프로세스 pid의 소유자와 같거나 혹은 보내는 프로세스의 소유자가 슈퍼유저이어야 한다.

```
#include <sys/types.h>
#include <signal.h>
int kill(int pid, int signo);
```

프로세스 pid에 시그널 signo를 보낸다. 성공하면 0 실패하면 -1를 리턴한다.

kill()

- *pid* > 0 :
  - signal to the process whose process ID is pid
- *pid* == 0 :
  - signal to the processes whose process group ID equals that of sender
- *pid* < 0 :
  - signal to the processes whose process group ID equals abs. of pid
- *pid* == -1 :
  - POSIX.1 leaves this condition unspecified (used as a broadcast signal in SVR4, 4.3+BSD)

예제: 제한 시간 명령어 실행

- tlimit.c 프로그램
  - 명령줄 인수로 받은 명령어를 제한 시간 내에 실행
  - execute3.c 프로그램을 알람 시그널을 이용하여 확장

- 프로그램 설명
  - 자식 프로세스가 명령어를 실행하는 동안 정해진 시간이 초과되면 SIGALRM 시그널이 발생
  - SIGALRM 시그널에 대한 처리함수 alarmHandler()에서 자식 프로세스를 강제 종료
  - kill(pid, SIGINT) 호출을 통해 자식 프로세스에 SIGINT 시그널을 보내어 강제 종료
  - 만약 SIGALRM 시그널이 발생하기 전에 자식 프로세스가 종료하면 이 프로그램은 정상적으로 끝남

tlimit.c

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
int pid;
void alarmHandler();
/* 명령줄 인수로 받은 명령어 실행에 제한
   시간을 둔다. */
int main(int argc, char *argv[])
{
   int child, status, limit;

   signal(SIGALRM, alarmHandler);
   sscanf(argv[1], "%d", &limit);
   alarm(limit);
   pid = fork( );
   if (pid == 0) {
      execvp(argv[2], &argv[2]);
      fprintf(stderr, "%s:실행 불가₩n", argv[1]);
   } else {
      child = wait(&status);
      printf("[%d] 자식 프로세스 %d 종료 ₩n",
            getpid(), pid);
   }
}
void alarmHandler()
{
   printf("[알람] 자식 프로세스 %d 시간 초과₩n", pid);
   kill(pid, SIGINT);
}
```

tlimit.c 실행

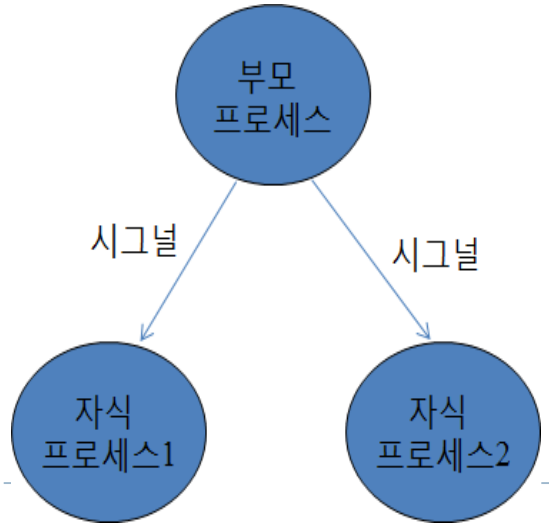
```
$ tlimit 5 sleep 6
[알람] 자식 프로세스 14012 시간 초과
[14011] 자식 프로세스 14012 종료
```

시그널을 이용한 프로세스 제어

- 시그널을 이용하여 다른 프로세스를 제어할 수 있다.

SIGCONT	프로세스 재개
SIGSTOP	프로세스 정지
SIGKILL	프로세스 종료
SIGTSTP	Ctrl-Z에서 발생
SIGCHLD	자식 프로세스 정지 혹은 종료 시 부모 프로세스에 전달

- 예제: 시그널을 이용한 자식 프로세스 제어



control.c

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

/* 시그널을 이용하여 자식 프로세스들을 제어한다. */
int main( )
{
   int pid1, pid2, count1=0, count2=0;

   pid1 = fork( );
   if (pid1 == 0) {
      while (1) {
         sleep(1);
         printf("자식 [1] 실행: %d₩n, ++count1");
      }
   }
```

control.c 실행

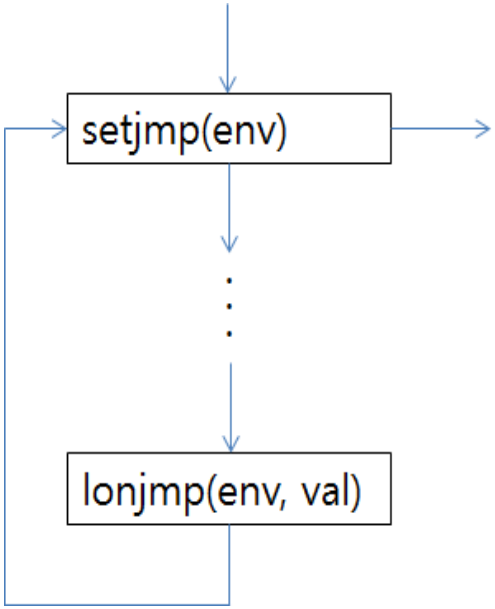
```
    pid2 = fork( );
    if (pid2 == 0) {
        while (1) {
            sleep(1);
            printf( " 자식 [2] 실행: %d₩n", ++count2);
        }
    }
    sleep(2);
    kill(pid1, SIGSTOP);
    sleep(2);
    kill(pid1, SIGCONT);
    sleep(2);
    kill(pid2, SIGSTOP);
    sleep(2);
    kill(pid2, SIGCONT);
    sleep(2);
    kill(pid1, SIGKILL);
    kill(pid2, SIGKILL);
}
```

```
$ control
자식[1] 실행: 1
자식[2] 실행: 1
자식[2] 실행: 2
자식[2] 실행: 3
자식[1] 실행: 2
자식[2] 실행: 4
자식[1] 실행: 3
자식[2] 실행: 5
자식[1] 실행: 4
자식[1] 실행: 5
자식[2] 실행: 6
자식[1] 실행: 6
자식[2] 실행: 7
자식[1] 실행: 7
```

11.4 시그널과 비지역 점프

비지역 점프 :setjmp/longjmp

- goto 문
  - 함수 내에서 지역 점프(local jump)
- 비지역 점프(longjmp)
  - 호출자 함수 내의 임의의 위치로 비지역 점프
  - 오류/예외 처리, 시그널 처리 등에 유용함
- int setjmp(jmp_buf env)
  - longjmp 전에 호출되어야 함
  - longjmp 할 곳을 지정함.
  - 한 번 호출되고 여러 번 반환함.
- void longjmp(jmp_buf env, int val)
  - setjmp 후에 호출됨
  - setjmp에 의해 설정된 지점으로 비지역 점프
  - 한 번 호출되고 반환하지 않음.



setjmp/longjmp

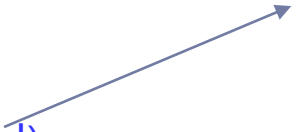
#include <setjmp.h>

int setjmp(jmp_buf env);

비지역 점프를 위해 스택 내용 등을 env에 저장한다.

(1) setjmp()는 처음 반환할 때 0을 반환하고

(2) longjmp()에 의해 두 번째 반환할 때는 0이 아닌 val 값을 반환한다.



void longjmp(jmp_buf env, int val);

env에 저장된 상태를 복구하여 스택 내용 등이 저장된 곳으로 비지역 점프한다.

구체적으로 상응하는 setjmp() 함수가 val 값을 반환하고 실행이 계속된다.

jump1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <setjmp.h>

void p1(), p2();
jmp_buf env;

int main()
{
    if (setjmp(env) != 0) {
        printf("오류로 인한 복귀 및 처리₩n");
        exit(0);
    }
   else printf("처음 통과₩n");

   p1();
}
```

jump1.c

```
void p1()
{
    p2();
}

void p2()
{
    int error;

    error = 1;
    if (error) {
        printf("오류 ₩n");
        longjmp(env, 1);
    }
}
```

```
$ jump1
처음 통과
오류
오류로 인한 복귀 및 처리
```

jump2.c

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

void p1();
void intHandler();
jmp_buf env;

int main()
{
    signal(SIGINT, intHandler);
    if (setjmp(env) != 0) {
        printf("인터립트로 인해 복귀₩n");
        exit(0);
    }
    else printf("처음 통과₩n");
    p1();
}
```

jump2.c

```
void p1()
{
    while (1) {
        printf("루프₩n");
        sleep(1);
    }
}

void intHandler()
{

    printf("인터럽트₩n");
    longjmp(env, 1);
}
```

```
$ jump2
처음 통과
루프
루프
루프
^C인터럽트
 인터립트로 인해 복귀
```

핵심 개념

- 시그널은 예기치 않은 사건이 발생할 때 이를 알리는 소프트웨어 인터럽트이다.
- signal() 시스템 호출은 특정 시그널에 대한 처리 함수를 지정한다.
- kill 명령어나 kill() 시스템 호출을 이용하여 특정 프로세스에 원하는 시그널을 보낼 수 있다.
- longjmp() 함수는 setjmp() 함수에 의해 설정된 지점으로 비지역 점프를 한다.