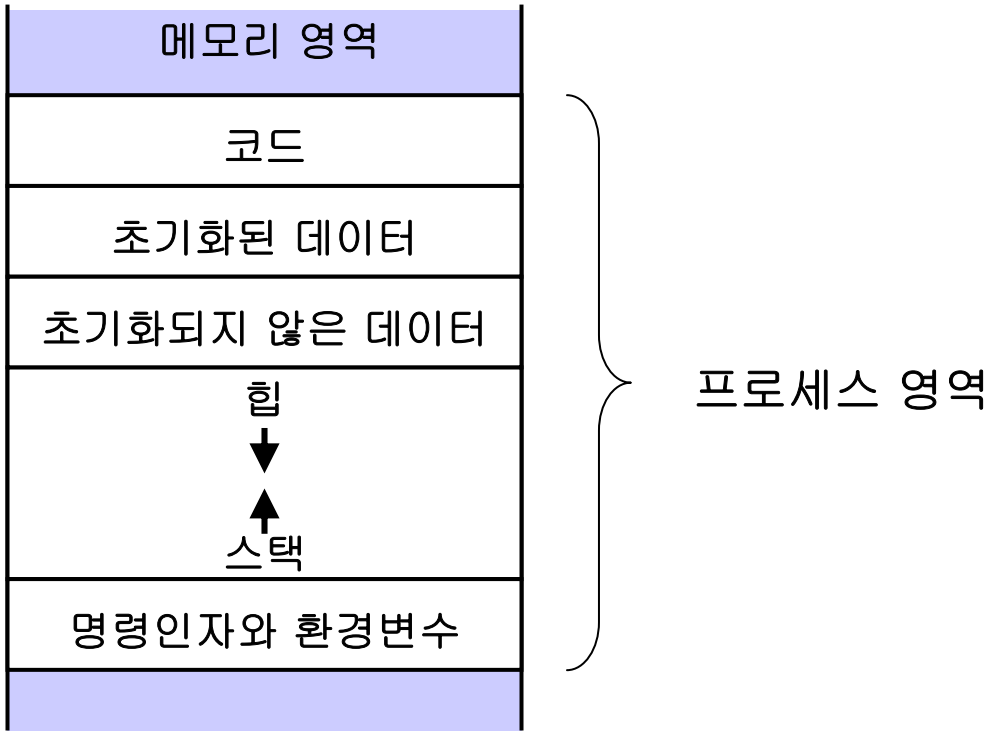
9장. 프로세스와 시그널

프로세스 제어

- 프로그램 – 디스크 상에 저장되어 있는 실행 가능한 파일
- 프로세스 – 수행중인 프로그램
  - 쉘 은 하나의 명령을 수행하기 위해 어떤 프로그램을 시작할 때마다 새로운 프로세스를 생성



프로세스 개념

- 프로세스 – 수행중인 프로그램
  - 쉘 은 하나의 명령을 수행하기 위해 어떤 프로그램을 시작할 때마다 새로운 프로세스를 생성
    - **Cat file1 file2** # 프로세스 생성
    - **Ls | wc –l** # 2 개의 프로세스 생성
  - **UNIX** 프로세스 환경은 디렉토리 트리처럼 계층적인 구조
    - 가장 최초의 프로세스는 **init** 이며 모든 시스템과 사용자 프로세스의 조상
  - 프로세스 시스템 호출들

이름	의 미
fork	호출 프로세스와 똑같은 새로운 프로세스를 생성
exec	한 프로세스의 기억공간을 새로운 프로그램으로 대체
wait	프로세스 동기화 제공. 연관된 다른 프로세스가 끝날 때까지 기다린다
exit	프로세스를 종료

프로세스 생성

fork 시스템 호출

기능

- 기본 프로세스 생성 함수
- 성공적으로 수행되면 호출 프로세스(부모 프로세스) 와 똑같은 새로운 프로세스(자식 프로세스) 생성

사용법

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

pid_t fork(void);
```

반환 값(return value)

- 정상 실행
  - 실행(부모) 프로세스는 : 생성(자식) 프로세스의 프로세스 ID 반환
  - 생성(자식) 프로세스는 : 0을 반환
- 이상 실행 : 음수 값을 반환

프로세스 식별번호(IDentifier)

프로세스 생성

▣ 프로세스 생성 함수 사용 예

◈ 프로세스 생성 예제 프로그램

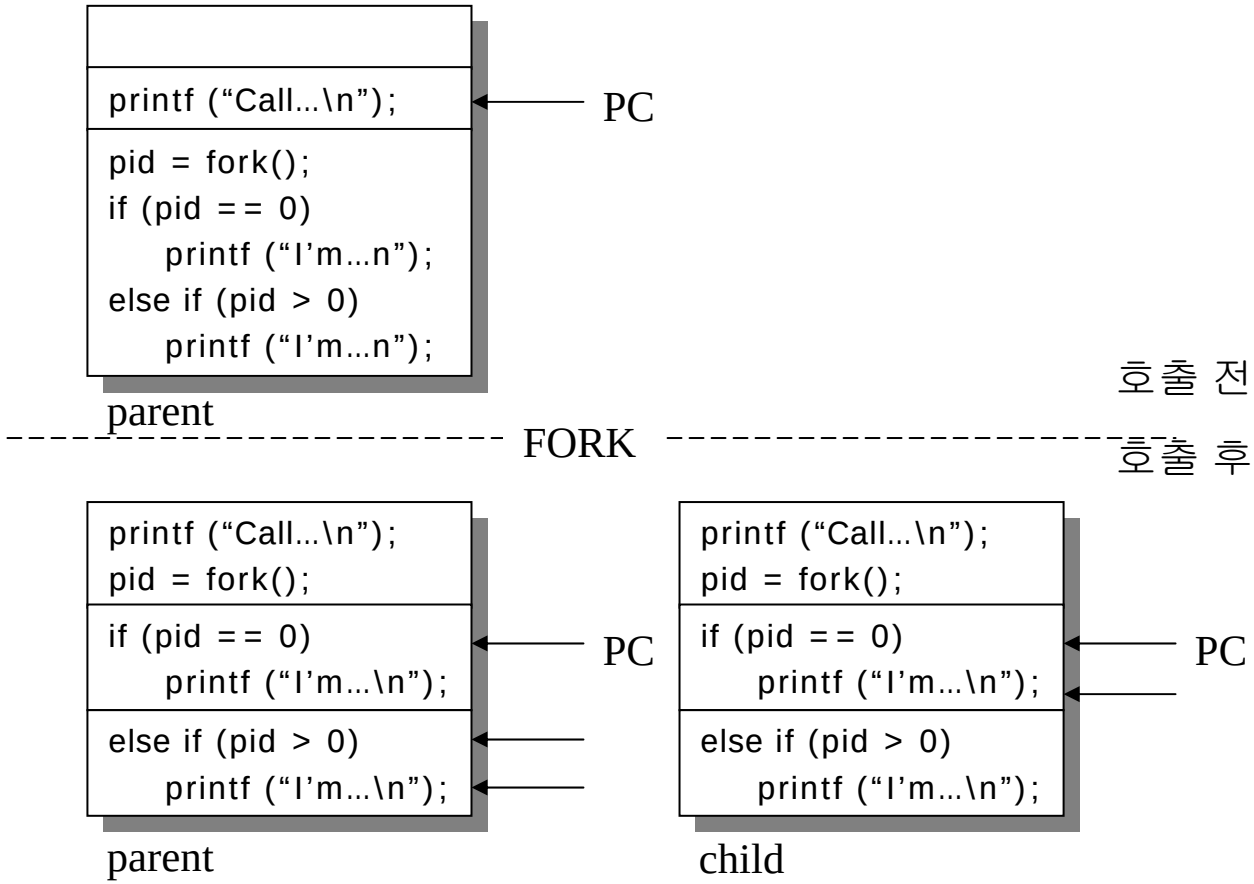
```
/* ex9_1.c */
/* fork example */
#include <sys/types.h>
#include <unistd.h>

main()
{
        pid_t pid;        /* 부모에서 프로세스 식별번호 저장 *
        printf("Calling fork ₩n");
        pid = fork();           /* 새로운 프로세스 생성 */
        if (pid == 0)
                printf("I'm the child process₩n");
        else if (pid > 0)
                printf("I'm the parent process₩n");
        else
                printf("fork failed₩n");
}
```

프로세스 생성

▣ 프로세스 생성 함수 사용 예

◈ 프로세스 생성 예제 프로그램

➢ **fork** 호출 후 두 프로세스는 바로 다음 문장부터 수행 계속



프로세스 생성

▣ 프로세스 생성 함수 사용 예

◈ 프로세스 생성 예제 프로그램 실행 결과

```
[cprog2@seps5 ch9]$ gcc ex9_1.c
[cprog2@seps5 ch9]$ ./a.out
Calling fork
I'm the child process
I'm the parent process
[cprog2@seps5 ch9]$
```

프로세스 종료

exit 시스템 호출

기능

- 프로세스 종료

사용법

```
#include <stdlib.h>

void exit (int status);
```

- **status :** 종료 상태 를 나타냄
- 반환값: 정상적 종료: **0**, 그렇지 않을 경우: **0** 이 아닌 값

모든 개방된 파일 기술자들을 닫는다.

프로세스 종료

▣ 프로세스 종료 함수 사용 예

◈ 프로세스 종료 예제 프로그램

```
/* ex9_2.c */
/* exit example */
#include <stdlib.h>

main()
{
        int exit_status;

        printf("enter exit status: ");
        scanf("%d", &exit_status);
        exit(exit_status);
}
```

◈ 프로세스 종료 프로그램 실행 결과

➢ **echo $?** 명령으로 종료 상태 확인

```
[cprog2@seps5 ch9]$ cprog2@seps5 ch9]$ gcc ex9_2.c
[cprog2@seps5 ch9]$ ./a.out
enter exit status: 0
[cprog2@seps5 ch9]$ echo $?
0
[cprog2@seps5 ch9]$ ./a.out
enter exit status: 2
[cprog2@seps5 ch9]$ echo $?
2
[cprog2@seps5 ch9]$
```

프로세스 종료

atexit 함수

기능

- 실행 프로세스 종료 시 호출되는 루틴 설정

사용법

```
#include <stdlib.h>
void atexit (void (*func) (void));
```

- 프로그래머가 종료 루틴을 정의

프로세스 종료

▣ 프로세스 종료 동작 함수 사용 예

◈ 프로세스 종료 동작 예제 프로그램

```
/* ex9_3.c */
/* atexit example */
#include <stdlib.h>
void exitfunc1(void);
void exitfunc2(void);

main()
{
        atexit(exitfunc1);
        atexit(exitfunc2);
        printf("This is main function.₩n");
}

void exitfunc1(void)
{
        printf("This is exit function 1.₩n");
}
```

```
void exitfunc2(void)
{
        printf("This is exit function 2.₩n");
}
```

◈ 프로그램 실행 결과

```
[cprog2@seps5 ch9]$ gcc ex9_3.c
[cprog2@seps5 ch9]$ ./a.out
This is main function.
This is exit function 2.
This is exit function 1.
[cprog2@seps5 ch9]$
```

프로세스 동기화

wait 시스템 호출

기능

- 자식 프로세스들 중 하나가 수행을 마치기를 기다린다

사용법

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

pid_t wait(int *status);
```

- **status : wait**가 복귀될 때 유용한 상태 정보 **(**퇴장 상태**)**
- 때로 부모 프로세스가 **fork** 로 자식을 만든 직후에 호출

```
cpid = fork();
if (cpid == 0) {          /* 자식 : 무언가 일을 수행한다 ... */
} else {                  /* 부모 : 자식을 기다린다 ... */
        cpid = wait (&status);
}
```

프로세스 동기화

▣ 프로세스 동기화 함수 사용 예

◈ 프로세스 동기화 예제 프로그램

```
/* /* ex9_4.c */
/* wait example */
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
        pid_t pid;
        int status, exit_status;

        if ((pid = fork()) < 0)
                perror ("fork failed");
        if (pid == 0) {
                sleep(4); /* 수행을 4초 동안 중단 */
                exit(5);
        }
        /* 부모 코드: 자식이 퇴장(exit)할 때까지 대기 */
        if ((pid = wait(&status)) == -1) {
                perror("wait failed");
                exit(2);
        }
        if (WIFEXITED(status)) {
                exit_status = WEXITSTATUS(status);
                  printf("Exit status from %d was %d₩n",
        pid, exit_status);
        }
        exit(0);
}
```

◈ 프로그램 실행 결과

```
[cprog2@seps5 ch9]$ gcc ex9_4.c
[cprog2@seps5 ch9]$  ./a.out
Exit status from 27368 was 5
[cprog2@seps5 ch9]$
```

프로세스 동기화

waitpid 시스템 호출

기능

- 특정 자식 프로세스가 끝나기를 기다림

사용법

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

pid_t waitpid(pid_t pid, int *status, int options);
```

- **options : 옵션**
  - WNOHANG ; 자식 프로세스가 수행중일 때, 상황을 감시하면서 루프를 돌 수 있게 한다

프로세스 동기화

waitpid 함수 사용 예

```
/* status2 – waitpid를 사용하여 자식의 퇴장 상태
    를 어떻게 알아내는지 보여준다 */
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
    pid_t pid;
    int status, exit_status;

    if ((pid = fork()) < 0)
        fatal ("fork failed");
    if (pid == 0) /* 자식 */
    {
        sleep (4); /* 수행을 4초 동안 중단 */
        exit (5);
    }
```

```
    /* 부모 코드 : 자식이 퇴장했는지 확인한다. 퇴장
    하지 않았으면, 1초 동안 잠든 후 다시 검사한다.
    */
    while (waitpid (pid, &status, WNOHANG) == 0) {
        printf ("still waiting... ₩n");
        sleep (1);
    }
    if (WIFEXITED(status))
    {
        exit_status = WEXITSTATUS (status);
        printf ("Exit status from %d was %d₩n",
    pid, exit_status);
    }
    exit (0);
}
```

좀비 프로세스

- 좀비(**zombi**) 와 너무 이른 퇴장
  - 부모 프로세스가 **wait** 를 수행하지 않고 있는 상태에서 자식이 퇴장할 때
    - 퇴장 프로세스는 좀비 프로세스가 됨
    - 부모 프로세스가 **wait** 를 수행할 때 삭제됨
  - 하나 이상의 자식 프로세스가 수행 중인 상태에서 부모가 퇴장할 때
    - 자식 프로세스는 **init** 프로세스(리눅스 최초 수행 프로세스)에게 맡겨짐

프로그램 수행

exec 계열 함수

기능

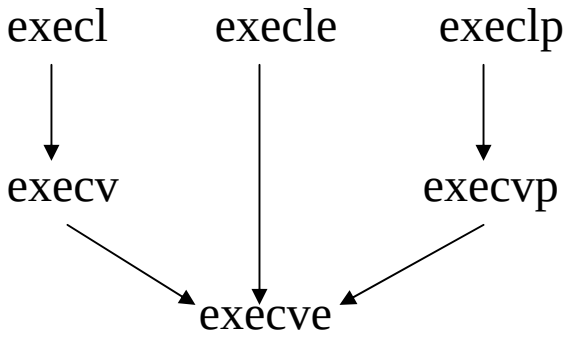
- 새로운 프로그램의 수행 시작

사용법

```
#include <unistd.h>

/* execl 호출 계열은 NULL 끝나는 리스트가 인수로 주어져야 한다 */
/* execl은 수행가능 파일에 대한 유효한 경로 이름이 주어져야 한다 */
int execl(const char *path, const char *arg0, ..., const char *argn, (char *) 0);
int execle(const char *path, const char *arg0, ..., const char *argn,
        (char *)0, char *const envp[]);
/* execlp은 단지 수행가능 파일의 파일 이름을 필요로 한다 */
int execlp(const char *file, const char *arg0, ..., const char *argn, (char *) 0);

/* execv 호출 계열은 반드시 인수의 배열이 주어져야 한다 */
int execv(const char *path, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
/* execvp는 단지 수행가능 파일의 파일 이름을 필요로 한다 */
int execvp(const char *file, char *const argv[]);
```

exec 계열

exec 계보



동작

- 호출 프로세스의 기억 공간에 새로운 프로그램을 적재
- 정상적으로 수행하는 경우 실행 프로세스로 복귀하지 않음
  - **fork** 와는 다름

exec 함수 사용 예

- 디렉토리를 나열하는 ls 예제 프로그램

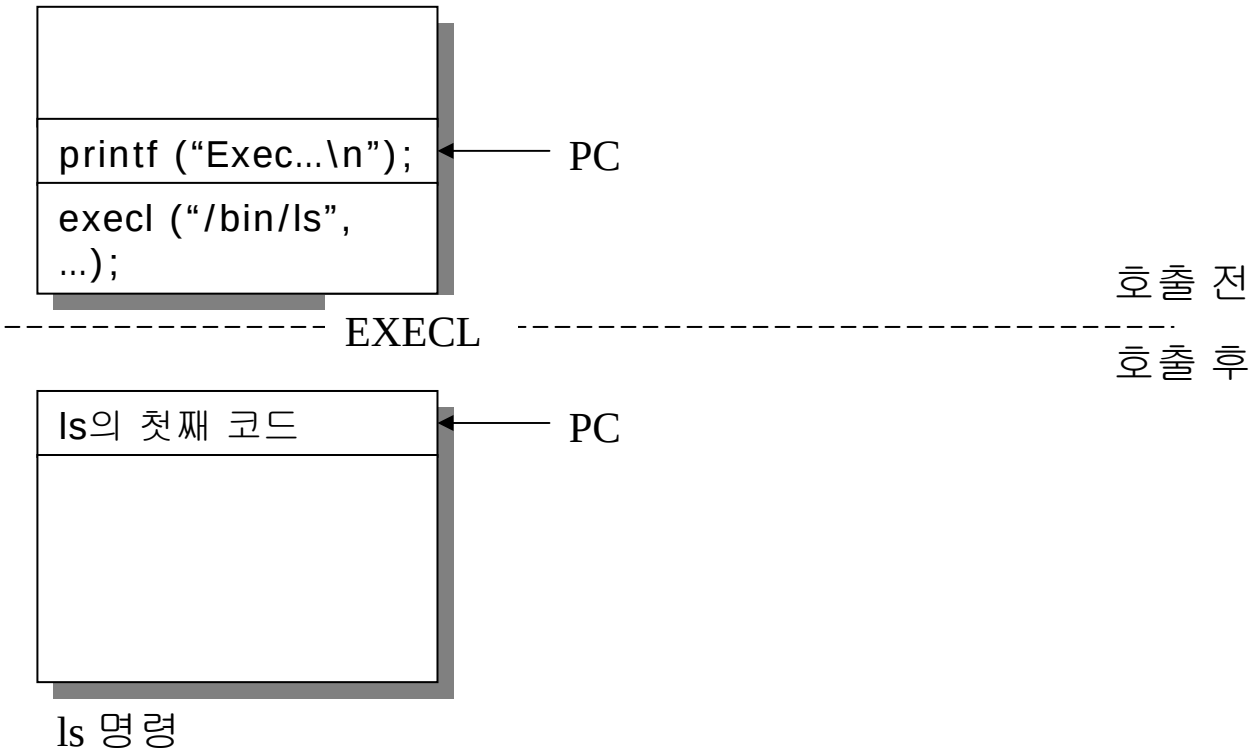
```
/* ex9_5.c */
/* execl 로 ls 실행 */
#include <unistd.h>

main()
{
        printf("Executing execl.₩n");

        execl("/bin/ls", "ls", "-l", (char *) 0);
        /* 만약 execl이 복귀하면, 호출은 실패. */
        perror("execl failed to run ls");
        exit(1);
}
```

- 프로그램 실행 결과

```
[cprog2@seps5 ch9]$ gcc ex9_5.c
[cprog2@seps5 ch9]$ ./a.out
executing execl.
합계 16
-rwxrwxr-x    1 cprog2   cprog2       9937  2월 16 17:19 a.out
-rw-rw-r--    1 cprog2   cprog2        243  2월 16 17:19 ex9_5.c
[cprog2@seps5 ch9]$
```

exec 함수 사용 예

- 디렉토리를 나열하는 프로그램 동작 과정



exec 함수 사용 예

execv, execlp, execvp 예제 프로그램

```
/* ex9_6.c */
/* execv 로 ls 실행 */
#include <unistd.h>

main()
{
        char * const argv[] = {"ls", "-l", (char *) 0};

        printf("executing execv.₩n");
        execv("/bin/ls", argv);
        /* 만약 execv 이 복귀하면, 호출은 실패. */
        perror("execv failed to run ls");
        exit(1);
}
```

- **execlp, execvp** 는 첫째 인수가 경로명이 아니라 파일 이름 – 경로는 환경변수 **PATH** 로 찾음

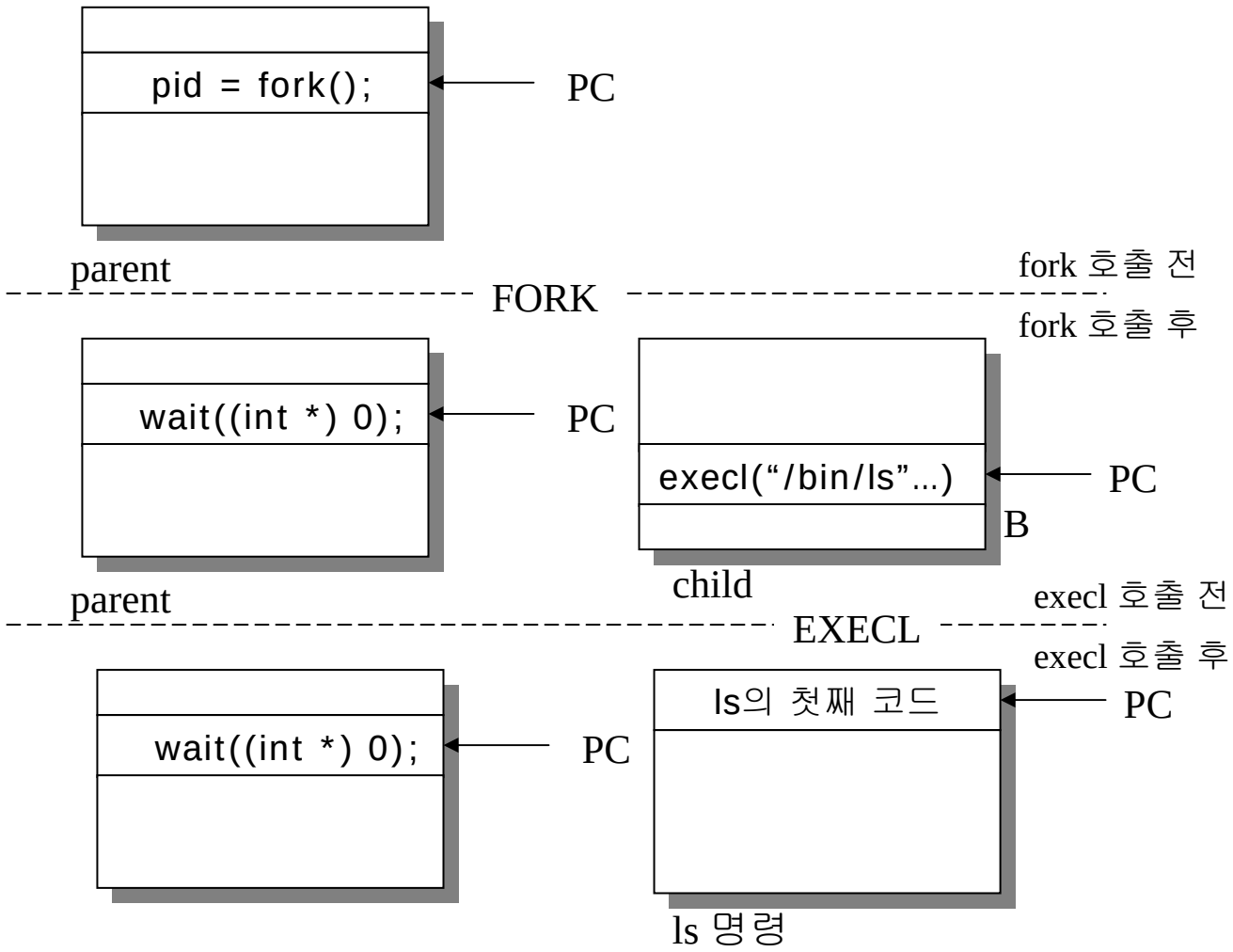
간단한 쉘 만들기

- **Fork 와 Exec 를 병행하여 사용한 프로그램**

```
/* ex9_7.c */
/* fork 와 execl을 함께 사용 */
#include <sys/types.h>
#include <unistd.h>

main()
{
        pid_t pid;

        pid = fork();
        if (pid == 0) {
                /* 자식 프로세스가 execl 호출 */
                execl("/bin/ls", "ls", "-l", (char *) 0);
                perror("execl failed");
        } else if (pid > 0) {
                /* 자식이 끝날 때까지 수행을 일시 중단하기 위해 wait 호출 */
                wait((int *) 0);
                printf("ls completed₩n");
                exit(0);
        } else
                perror("fork failed");
}
```

간단한 쉘 만들기

- **Fork 와 Exec 를 병행 사용 프로그램 동작**



간단한 쉘 만들기

▣ Fork 와 Exec 를 병행하여 사용한 프로그램

◈ Fork 와 Exec 를 병행하여 사용한 프로그램 실행 결과

```
[cprog2@seps5 ch9]$ gcc ex9_7.c
[cprog2@seps5 ch9]$ ./a.out
합계 16
-rwxrwxr-x    1 cprog2   cprog2       9937  2월 16 17:19 a.out
-rw-rw-r--    1 cprog2   cprog2        243  2월 16 17:19 ex9_7.c
ls completed
[cprog2@seps5 ch9]$
```

간단한 쉘 만들기

▣ 간단한 쉘 구현 프로그램

◈ 기능

- ➢ 간단한 명령 처리기

◈ 특징

- ➢ 이전에 학습한 내용을 발전
- ➢ 쉘과 표준 **UNIX** 명령과 유틸리티들에 대한 이해

◈ 쉘 프로그램의 전체적인 수행 과정

```
while(1) { /* 무한 루프 */
        사용자로부터 명령어 입력을 받아들인다.
        명령어 인자를 구분한다.
        자식 프로세스를 생성한다.
        if (자식 프로세스) 명령어를 실행한다.
        else if (부모 프로세스)    자식 프로세스를 기다린다.
}
```

간단한 쉘 만들기

▣ 간단한 쉘 프로그램

```
/* ex9_8.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
        char buf[256];
        char *argv[50];
        int narg;
        pid_t pid;

        while (1) {
                printf("shell> ");
                gets(buf);
                narg = getargs(buf, argv);

                pid = fork();
                if (pid == 0)
                        execvp(argv[0], argv);
                else if (pid > 0)
                        wait((int *) 0);
                else
                        perror("fork failed");
        }
}

int getargs(char *cmd, char **argv)
{
        int narg = 0;

        while (*cmd) {
                if (*cmd == ' ' || *cmd == '₩t')
                        *cmd++ = '₩0';
                else {
                        argv[narg++] = cmd++;
                          while (*cmd != '₩0' && *cmd != ' '
    && *cmd != '₩t')
                                cmd++;
                }
        }
        argv[narg] = NULL;
        return narg;
}
```

간단한 쉘 만들기

▣ 간단한 쉘 구현 프로그램

◈ 간단한 쉘 구현 프로그램 실행 결과

```
[cprog2@seps5 ch9]$ gcc ex9_8.c
/tmp/ccmg9tcy.o(.text+0x2e): In function `main':
: the `gets' function is dangerous and should not be used.
[cprog2@seps5 ch9]$ ./a.out
shell> ls
a.out    ex9_2.c  ex9_4.c  ex9_6.c  ex9_8.c
ex9_1.c  ex9_3.c  ex9_5.c  ex9_7.c
shell> date
금  2월 20 10:50:43 KST 2004
shell> pwd
/home/cprog2/c2/ch9
shell>
[cprog2@seps5 ch9]$
```

상속된 자료와 파일 기술자

- **fork** 수행 시 파일과 자료
  - 부모 프로세스로부터 자식 프로세스로 자료 복제
  - **File**
    - 부모 프로세스가 개방한 file 은 자식 프로세스에서도 개방
    - 파일 기술자만 복제
    - 파일 포인터는 공유 – 시스템에 의해 관리
      - 파일 포인터를 전진시키면 어느 프로세스에서든 전진됨

fork 시 파일 처리 예

```
/* proc_file – fork시 파일이 어떻게 취급되는지 보인다. */
/* fork는 "data"가 최소한 20문자 이상임을 가정한다. */
#include <unistd.h>
#include <fcntl.h>

main()
{
    int fd;
    pid_t pid;          /* 프로세스 식별번호 */
    char buf[10];       /* 파일 자료를 저장할 버퍼 */

    if ((fd = open ("data", O_RDONLY)) == -1)
        fatal ("open failed");

    read (fd, buf, 10); /* 파일 포인터를 전진시킨다 */

    printpos ("Before fork", fd);

    /* 이제 두 개의 프로세스를 생성한다 */
    switch ((pid = fork()) {
    case –1:                    /* 오류 */
        fatal ("folk failed");
        break;
    case 0:             /* 자식 */
        printpos ("Child before read", fd);
        read (fd, buf, 10);
        printpos ("Child after read", fd);
        break;
    default:
        wait ((int *) 0);
        printpos ("Parent after wait", fd);
    }
}

/* 파일 내에서 위치를 출력한다 */
int printpos (const char *string, int filedes)
{
    off_t pos;
    if ((pos = lseek (filedes, 0, SEEK_CUR)) == -1)
        fatal ("lseek failed");
    printf ("%s:%ld\n", string, pos);
}
```

exec 와 개방된 파일

- 보통 **open** 된 파일 기술자들도 **exec** 에 의해 변형된 프로세스에 전달됨
- **fcntl** 시스템 호출을 이용하여 **close-on-exec** 플래그 설정
  - **On** 일 때 **(**기본 상태는 **off), exec** 가 호출될 때 파일이 **close** 됨

```
#include <fcntl.h>

int fd;
fd = open (“file”, O_RDONLY);

/* close-on-exec 플래그를 on으로 설정 */
fcntl (fd, F_SETFD, 0);
```

프로세스 속성

▣ 프로세스 식별번호

◈ 각 프로세스는 고유의 식별번호를 가짐

- ➢ 한 시점에서 유일한 프로세스 번호
- ➢ 프로세스 **0 :** 스케줄러, 즉 스와퍼 **(swapper)** 프로세스
- ➢ 프로세스 **1 : init** 초기화 프로세스

```
pid = getpid();    /* 프로세스 식별번호 획득 */

ppid = getppid(); /* 부모 프로세스의 식별번호 획득 */
```

getpid 예

▣ 프로세스 식별 함수 사용 예

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

static int num = 0;
static char namebuf[20];
static char prefix[] = “/tmp/tmp”;

char *gentemp(void)
{
    int length;
    pid_t pid;

    pid = getpid();        /* 프로세스 식별번호를 얻는다 */
    /* 표준 문자열처리 루틴들 */
    strcpy (namebuf, prefix);
    length = strlen(namebuf);

    /* 파일 이름에 프로세스 식별번호(pid)를 추가한다 */
    itoa (pid, &namebuf[length]);

    strcat (namebuf, “.”);
    length = strlen (namebuf);
    do {
            /* 접미 번호를 추가한다 */
            itoa (num++, &namebuf[length]);
    } while (access(namebuf, F_OK) != -1) ;
    return (namebuf);
}

/* itoa – 정수를 문자열로 변환한다 */
int itoa(int i, char *string)
{
    int power, j;

    j = i;
    for (power = 1; j >= 10; j /= 10)
            power *= 10;

    for ( ; power > 0; power /= 10)
    {
            *string++ = ‘0’ + I/power;
            i %= power;
    }

    *string = ‘₩0’;
}
```

시그널

- 시그널
  - 다른 프로세스에게 이벤트 발생을 알리는 소프트웨어 인터럽트
  - 시그널 발생 예
    - 프로그램 실행 시 **Ctrl-C**(인터럽트 키) 를 눌러 강제 종료시킬 때
    - 백그라운드 작업을 종료시킬 때, **kill** 명령 사용

시그널

▣ 시그널의 종류 (30여가지)

이름	의 미
SIGHUP	hangup. 단말기 연결이 끊어졌을 때 그 단말기에 연결된 모든 프로세스에 보냄. 이것을 받으면 종료.
SIGINT	interrupt. 사용자가 인터럽트키를 칠 때 단말기와 연결된 모든 프로세스에 보냄. 수행중인 프로그램을 중지시키는 일반적인 방법
SIGQUIT	quit. SIGINT 와 마찬가지로 사용자가 단말기에서 종료(quit)키를 칠 때 보냄. 종료키의 일반적인 값은 ASCII FS 또는 CTRL-₩
SIGILL	illegal Instruction. 비정상적인 명령 수행 시 보냄
SIGTRAP	trace trap. ptrace 시스템과 함께 sdb 또는 adb 등의 디버거에 의해 사용되는 특별한 시그널.
SIGABRT	abort. 현재 프로세스가 abort 함수를 호출할 때 보냄. 비정상적인 종료(abnormal termination) 가 됨. 이것을 받으면 코어 덤프하고 종료.
SIGFPE	floating-point exception. 오버플로우나 언더풋로우 같은 부동 소숫점 오류가 발생했을 때 보냄
SIGKILL	kill. 프로세스로부터 다른 프로세스를 종료시키기 위해 보냄.
SIGUSR1 SIGUSR2	SIGTERM 과 마찬가지로 이것들은 커널에 의해 사용되는 것이 아니라 사용자가 원하는 목적을 위하여 사용 가능.
SIGSEGV	segmentation violation. 프로세스가 유효하지 않은 메모리 주소에 접근할 때 보냄. 이것을 받으면 비정상적 종료.

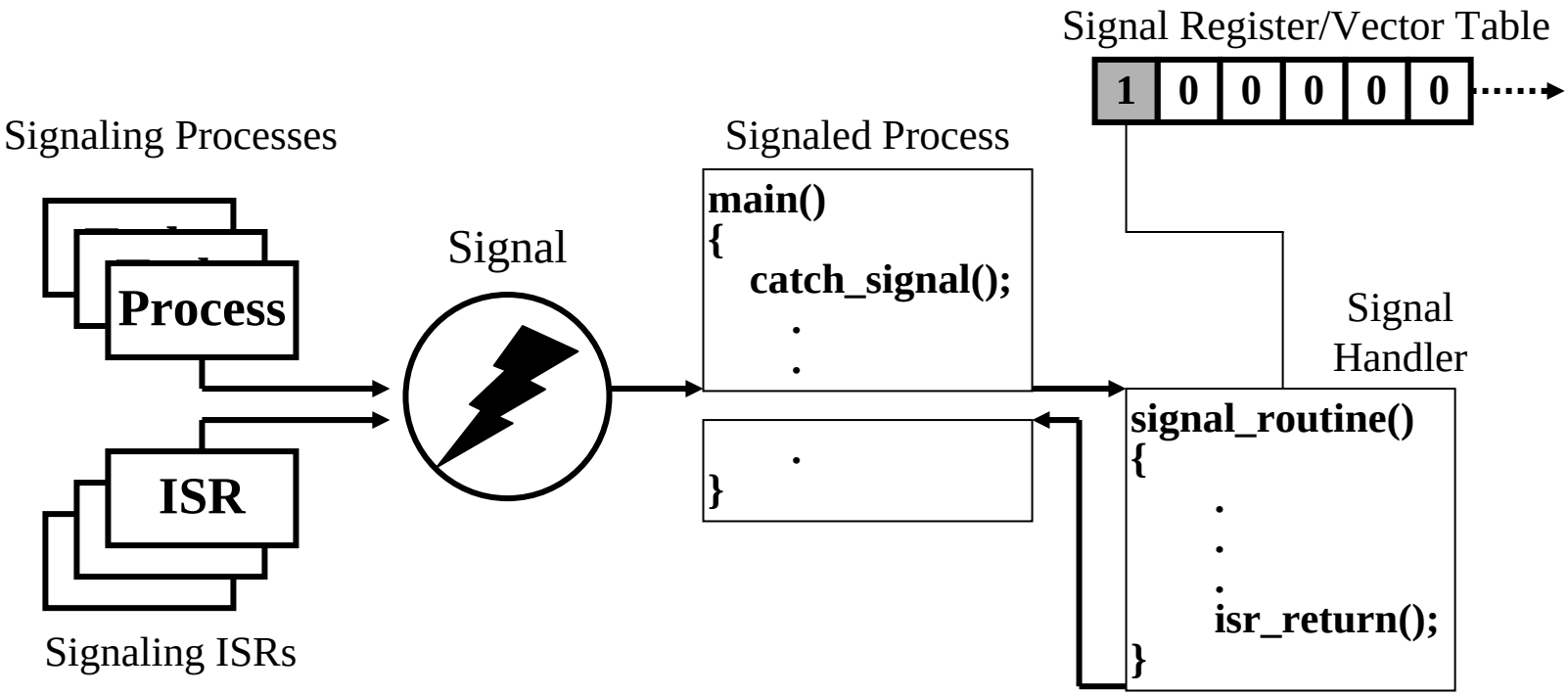
시그널

▣ 시그널의 종류 계속

이름	의 미
SIGPIPE	write on a pipe with no-one to read it. 종료한 파이프나 소켓에 쓸 때 보냄. (파이프는 또다른 프로세스간 통신 방식)
SIGALRM	alarm clock. 타이머가 만료되었을 때 프로세스에 보냄. alarm 함수로 이루어짐.
SIGTERM	software termination. 프로세스를 종료시키기 위해 사용자에 의해 사용.
SIGSTKFLT	stack fault. 스택 오류일 때 보냄.
SIGCHLD	child status has changed. 자식 프로세스가 종료하거나 중단될 때 부모 프로세스에 보냄. 이 시그널을 받으면 무시.
SIGCONT	continue. 이 시그널을 받으면 중단된 프로세스의 경우 계속 실행하고, 실행중일 때는 무시.
SIGSTOP	stop, unblockable. 프로세스를 중단시키기 위해 보냄.
SIGTSTP	keyboard stop. 사용자가 일시 중지 키 (Ctrl-Z)을 칠 때 보냄. SIGSTOP와 비슷
SIGTTIN	백그라운드 프로세스가 단말기로부터 읽기를 시도할 때 보냄.
SIGTTOU	백그라운드 프로세스가 단말기로 쓰기를 시도할 때 보냄.
SIGURG	프로세스에게 네트워크 연결에 긴급하거나 대역을 벗어난 자료가 수신되었음을 알림.
SIGSYS	bad arguments to a system call. 이것은 잘못된 시스템 호출을 보냈을 때 보냄.

시그널

- 시그널 동작
  - 1. 기본적으로 설정한 동작 수행
  - 2. 시그널을 무시하고 프로그램 수행 계속
  - 3. 미리 정해진 일을 수행



시그널 집합

- 시그널 집합 - 시그널을 원소로 하는 집합
- 관련 시스템 호출
  - 기능
    - 시그널 집합 관련 기능 수행
    - 각각의 시그널을 따로 처리하는 것보다 편리
  - 사용법

```
#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);
```

    - **sigemptyset** – 비어있는 시그널 집합 생성
    - **sigfillset** – 모든 시그널을 포함하는 집합 생성
    - **sigaddset** – 특정 시그널 번호를 집합에 추가
    - **sigdelset** – 특정 시그널 번호를 집합에서 제거

시그널 처리

sigaction 시스템 호출

기능

- 프로세스가 특정 시그널에 대해 다음 세 가지 행동 중 하나를 지정
  - 프로세스는 종료하고 코어 덤프
  - 시그널을 무시
  - 시그널 핸들러에 지정된 함수를 수행

사용법

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

- **signo: signal number**
- **act : sigaction 구조체**
- **oact:** 이전 설정값

시그널 처리

▣ sigaction 시스템 호출

◈ sigaction 구조체

```
struct sigaction {
    void (*) (int) sa_handler(int); /* 함수, SIG_DFL 또는 SIG_IGN */
    void (*sa_sigaction)(int, siginfo_t *, void *); /* 시그널 핸들러 포인터 */
    sigset_t sa_mask; /* sa_handler에서 봉쇄할 시그널 */
    int sa_flags; /* 시그널 동작 변경자 */
};
```

- **sa_handler: signal number**
- **Sa_sigaction: sigaction 구조체**
- **Sa_mask: 이전 설정값**
- **Sa_flags:**
  - SA_NOCLDSTOP
  - SA_RESETHAND
  - SA_RESTART
  - SA_NODEFER
  - SA_SIGINFO

시그널 처리

▣ 시그널 처리함수 사용 예

◈ 시그널 핸들러가 인터럽트 시그널을 받는 프로그램

```
/* ex9_9.c */
/* sigaction example */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
void handler(int signo);

main()
{
        struct sigaction act;
        int i = 0;
        act.sa_handler = handler;
        sigfillset(&(act.sa_mask));
        sigaction(SIGINT, &act, NULL);
        printf("SIGINT on₩n");
        while(1) {
                sleep(1);
                printf("sleep for %d sec(s).₩n", ++i);
        }
}
```

```
void handler(int signo)
{
        printf("handler: signo=%d₩n", signo);
}
```

시그널 처리

▣ 시그널 처리함수 사용 예

◈ 시그널 핸들러가 인터럽트 시그널을 받는 프로그램 실행 결과

```
[cprog2@seps5 signal]$ gcc ex9_9.c
[cprog2@seps5 signal]$ ./a.out
SIGINT on
sleep for 1 sec(s).
sleep for 2 sec(s).
sleep for 3 sec(s).
handler: signo=2    <---- Ctrl-C 입력
sleep for 4 sec(s).
sleep for 5 sec(s).<---- Ctrl-C 입력
[cprog2@seps5 signal]$
```

시그널 처리

▣ 시그널 처리함수 사용 예

◈ 시그널을 무시하는 프로그램

```
/* ex9_10.c */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

main()
{
      struct sigaction act;
        int i = 0;
        act.sa_handler = SIG_IGN;
        sigfillset(&(act.sa_mask));
        sigaction(SIGINT, &act, NULL);
        printf("SIGINT off₩n");
        while(1) {
                sleep(1);
                printf("sleep for %d sec(s).₩n", ++i);
        }
}
```

시그널 처리

▣ 시그널 처리함수 사용 예

◈ 시그널을 무시하는 프로그램 실행 결과

```
[cprog2@seps5 signal]$ gcc ex9_10.c
[cprog2@seps5 signal]$ ./a.out
SIGINT off
sleep for 1 sec(s).
sleep for 2 sec(s). <---- Ctrl-C 입력
sleep for 3 sec(s).
sleep for 4 sec(s). <---- Ctrl-C 입력
sleep for 5 sec(s).
sleep for 6 sec(s). <---- Ctrl-₩ 입력
[cprog2@seps5 signal]$
```

시그널 전송

kill 시스템 호출

기능

- 다른 프로세스에게 특정 시그널 전송

사용법

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

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

- **sig:** 전송할 시그널
- **pid:** 시그널 **sig** 를 받을 프로세스 **ID**
  - 양수: 해당 프로세스 식별번호를 가진 프로세스
  - 0: kill 을 호출한 프로세스와 그룹에 속하는 모든 프로세스에게 전송
  - -1: 1번 프로세스를 제외한 모든 프로세스에게 전송, 프로세스 유효사용자 ID가 수퍼유저가 아니면, 유효사용자 ID가 같은모든 프로세스에게 전송
  - 음수: 프로세스 그룹 식별번호가 pid 의 절대값과 같은 모든 프로세스에게 전송

시그널 전송

pause 시스템 호출

기능

- 시그널이 도착할 때까지 프로세스 실행을 중단하고 대기

사용법

```
#include <unistd.h>

int pause(void);
```

시그널 전송

▣ 시그널 전송 함수 사용 예

◈ 시그널을 전송하고 대기하는 프로그램

```
/* ex9_11.c */
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
int i = 0;
void p_handler (int), c_handler (int);

main()
{
        pid_t pid, ppid;
        struct sigaction act;

        pid = fork();
        if (pid == 0) {
                act.sa_handler = c_handler;
                sigaction (SIGUSR1, &act, NULL);
                ppid = getppid(); /* get parent's process id. */
                while (1) {
                        sleep (1);
                        kill (ppid, SIGUSR1);
                        pause();
                }
        } else if (pid > 0) {
                act.sa_handler = p_handler;
                sigaction(SIGUSR1, &act, NULL);
                while (1) {
                        pause();
                        sleep (1);
                        kill (pid, SIGUSR1);
                }
        } else
                perror ("Error");
}

void p_handler(int signo)
{
        printf("Parent handler: call %d times.₩n", ++i);
}

void c_handler(int signo)
{
        printf("Child handler: call %d times.₩n", ++i);
}
```

시그널 전송

▣ 시그널 전송 함수 사용 예

◈ 시그널을 전송하고 대기하는 프로그램 실행 결과

```
[cprog2@seps5 signal]$ gcc ex9_11.c
[cprog2@seps5 signal]$ ./a.out
Parent handler: call 1 times.
Child handler: call 1 times.
Parent handler: call 2 times.
Child handler: call 2 times.

[cprog2@seps5 signal]$
```

시그널 전송

raise 시스템 호출

기능

- 현재 프로세스에게 시그널 전송

사용법

```
#include <signal.h>

int raise(int sig);
```

- **sig:** 전송할 시그널

시그널 전송

▣ 시그널 전송 함수 사용 예

◈ 자신에게 시그널을 전송하는 프로그램

```
/* ex9_12.c */
/* raise example */
#include <signal.h>

main()
{
        int a, b;

        a = 10;
        b = 0;
        if (b == 0) /* preempt divide by zero error */
                raise(SIGFPE);
        a = a / b;
}
```

◈ 프로그램 실행 결과

```
[cprog2@seps5 signal]$ gcc ex9_12.c
[cprog2@seps5 signal]$ ./a.out
부동 소수점 예외
[cprog2@seps5 signal]$
```

시그널 전송

alarm 시스템 호출

기능

- 지정된 초 후에 현재 프로세스에게 **SIGALRM** 시그널을 전달

사용법

```
#include <unistd.h>

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

- **secs: SIGALRM** 시그널을 전송하기 전까지 기다리는 시간(초)

실행한 후 즉시 복귀하여 정상적인 수행 계속

alarm 취소

- **alarm(0)** 호출

시그널 전송

▣ 시그널 전송 함수 사용 예

◈ alarm 함수 사용 예제 프로그램

```
/* ex9_13.c */
/* alarm example */
#include <stdio.h>
#include <signal.h>
void alarm_handler(int);
int alarm_flag = 0;

main()
{
        struct sigaction act;

        act.sa_handler = alarm_handler;
        sigaction(SIGALRM, &act, NULL);

        alarm (5);        /* Turn alarm on. */
        pause(); /* pause */
        if (alarm_flag)
                printf("Passed a 5 secs.₩n");
}
```

```
void alarm_handler(int sig)
{
        printf("Received a alarm signal.₩n");
        alarm_flag = 1;
}
```

◈ 프로그램 실행 결과

```
[cprog2@seps5 signal]$ gcc ex9_13.c
[cprog2@seps5 signal]$ ./a.out
Received a alarm signal.
Passed a 5 secs.
[cprog2@seps5 signal]$
```

시그널 차단

sigprocmask 시스템 호출

기능

- 특정 시그널을 차단하도록 허용

사용법

```
#include <signal.h>

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

- **how: sigprocmask** 가 어떻게 동작할 것인지를 결정
  - SIG_BLOCK: 차단하는 시그널은 현재 설정된 것들과 set 에 포함된 것
  - SIG_UNBLOCK: set 에 포함된 시그널들이 차단되지 않도록 설정
  - SIG_SETMASK: set 에 포함된 시그널들이 차단되도록 설정
- **set:** 선택할 시그널들의 집합
- **oset:** 이전에 설정된 시그널들의 마스크 값

시그널 차단

▣ 시그널 차단 함수 사용 예

◈ 시그널을 차단하고 해제하는 프로그램

```
/* ex9_14.c */
/* sigprocmask example */
#include <stdio.h>
#include <signal.h>

main()
{
        sigset_t set1, set2;

        /* fill set1 */
        sigfillset(&set1);
        /* set up set2 with just SIGINT */
        sigemptyset(&set2);
        sigaddset(&set2, SIGINT);
        printf("Critical region start.₩n");
            sigprocmask(SIG_BLOCK, &set1, NULL); /*
block */
        sleep(5);
        printf("Less critical region start.₩n");
         sigprocmask(SIG_UNBLOCK, &set2, NULL); /*
unblock SIGINT */
        sleep(5);
        printf("Non critical region start.₩n");
         sigprocmask(SIG_UNBLOCK, &set1, NULL); /*
unblock all */
        sleep(5);
}
```

시그널 차단

▣ 시그널 차단 함수 사용 예

◈ 시그널을 차단하고 해제하는 프로그램 실행 결과

```
[cprog2@seps5 signal]$ gcc ex9_14.c
[cprog2@seps5 signal]$./a.out
Critical region start.          <---- Ctrl-C 입력
Less critical region start.

[cprog2@seps5 signal]$./a.out
Critical region start.          <---- Ctrl-₩ 입력
Less critical region start.
Non critical region start.
종료
[cprog2@seps5 signal]$
```

시그널 차단

- **SIGUSR1과 SIGUSR2 중 SIGUSR2를 항상 먼저 받도록 하는 프로그램**

```
/* ex9_15.c */
/* sigprocmask example */
#include <stdio.h>
#include <signal.h>

void handler1(int);
void handler2(int);
sigset_t set;

int main()
{
        struct sigaction act;

        /* Set up signal set with just SIGUSR1. */
        sigemptyset(&set);
        sigaddset(&set, SIGUSR1);

        /* Trap SIGUSR1. */
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = handler1;
        sigaction(SIGUSR1, &act, NULL);

        /* Trap SIGUSR2 - just change handler in action. */
        act.sa_handler = handler2;
        sigaction(SIGUSR2, &act, NULL);

        /* Block SIGUSR1 */
        sigprocmask(SIG_BLOCK, &set, NULL);

    /* We will get a USR2 first because USR1 is blocked. */
        while(1) {
                pause();
                sigprocmask(SIG_UNBLOCK, &set, NULL);
                                /* Unblock USR1 here. */
        }
}
void handler1(int sig)
{
        printf("Got a SIGUSR1 signal₩n");
}
void handler2(int sig)
{
        printf("Got a SIGUSR2 signal₩n");
}
```

시그널 차단

- **SIGUSR1과 SIGUSR2 중 SIGUSR2를 항상 먼저 받도록 하는 프로그램**
  - ◈ 프로그램 실행 결과

```
[cprog2@seps5 signal]$ ./a.out

[cprog2@seps5 signal]$ ./a.out &
[1] 19688
[cprog2@seps5 signal]$ kill -SIGUSR1 19688
[cprog2@seps5 signal]$ kill -SIGUSR1 19688
[cprog2@seps5 signal]$ kill -SIGUSR2 19688
Got a SIGUSR2 signal
Got a SIGUSR1 signal
[cprog2@seps5 signal]$ kill -SIGUSR2 19688
Got a SIGUSR2 signal
[cprog2@seps5 signal]$
```

시그널로부터 이전 상태 복귀

sigsetjmp 와 siglongjmp 시스템 호출

기능

- **sigsetjmp** 함수는 현재 프로그램 상태를 저장
- **siglongjmp** 함수는 저장된 위치로 복귀

사용법

```
#include <setjmp.h>

int sigsetjmp(sigjmp_buf env, int savesigs);
void siglongjmp(sigjmp_buf env, int val);
```

- **env:** 프로그램 위치를 저장할 **sigsetjmp_buf** 형의 객체
- **val:** 복귀한 뒤 **sigsetjmp** 에 의해 반환할 값

sigsetjmp 함수는 현재의 스택과 시그널 마스크 저장

- **siglongjmp** 가 호출되어 저장한 위치로 복귀할 때 이들을 복구

sigsetjmp 함수의 반환값

- 직접 반환되면 **0** 을 반환
- **Siglongjmp** 에 의해 반환되면 **0** 이 아닌 값**(val)** 반환

시그널로부터 이전 상태 복귀

▣ 시그널 복귀 함수 사용 예

◈ 피보나치 수열(a[i]=a[i-1]+a[i-2]) 출력 프로그램

```
/* ex9_16.c */
/* example use of sigsetjmp and siglongjmp */
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>

void start(int signo);
sigjmp_buf jbuf;

main()
{
        struct sigaction act;
        int cur_i, past_i, tmp_i;

        /* 현재 위치를 저장 */
        if (sigsetjmp(jbuf, 1) == 0) {
                act.sa_handler = start;
                sigaction(SIGINT, &act, NULL);
        }
        cur_i = past_i = 1;
        while (1) {
                printf("%d₩n", cur_i);
                tmp_i = cur_i;
                cur_i += past_i;
                past_i = tmp_i;
                sleep(1);
        }
}

void start(int signo)
{
        fprintf(stderr, "Interrupted₩n");
        siglongjmp(jbuf, 1); /* 저장된 곳으로 복귀 */
}
```

시그널로부터 이전 상태 복귀

▣ 시그널 복귀 함수 사용 예

◈ 피보나치 수열(a[i]=a[i-1]+a[i-2]) 출력 프로그램 실행 결과

```
[cprog2@seps5 signal]$ gcc ex9_16.c
[cprog2@seps5 signal]$ ./a.out
1
2
3     <---- Ctrl-C 입력
Interrupted
1
2
3
5
8
13    <---- Ctrl-₩ 입력
종료
[cprog2@seps5 signal]$
```