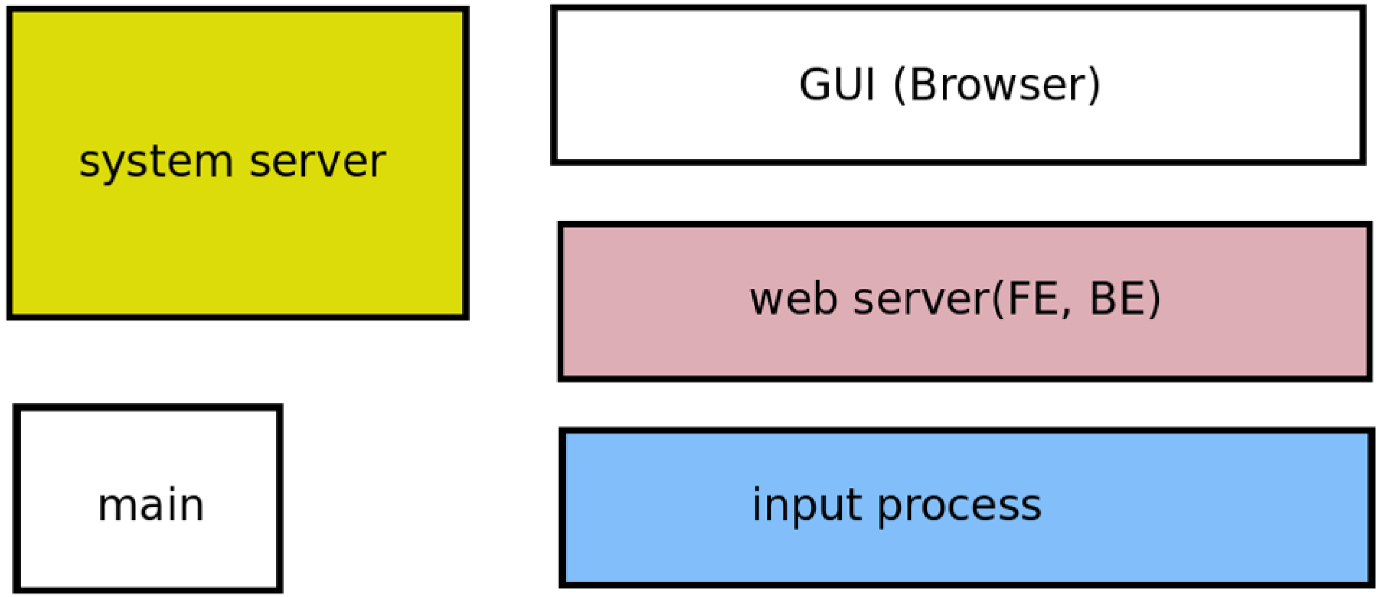
시그널

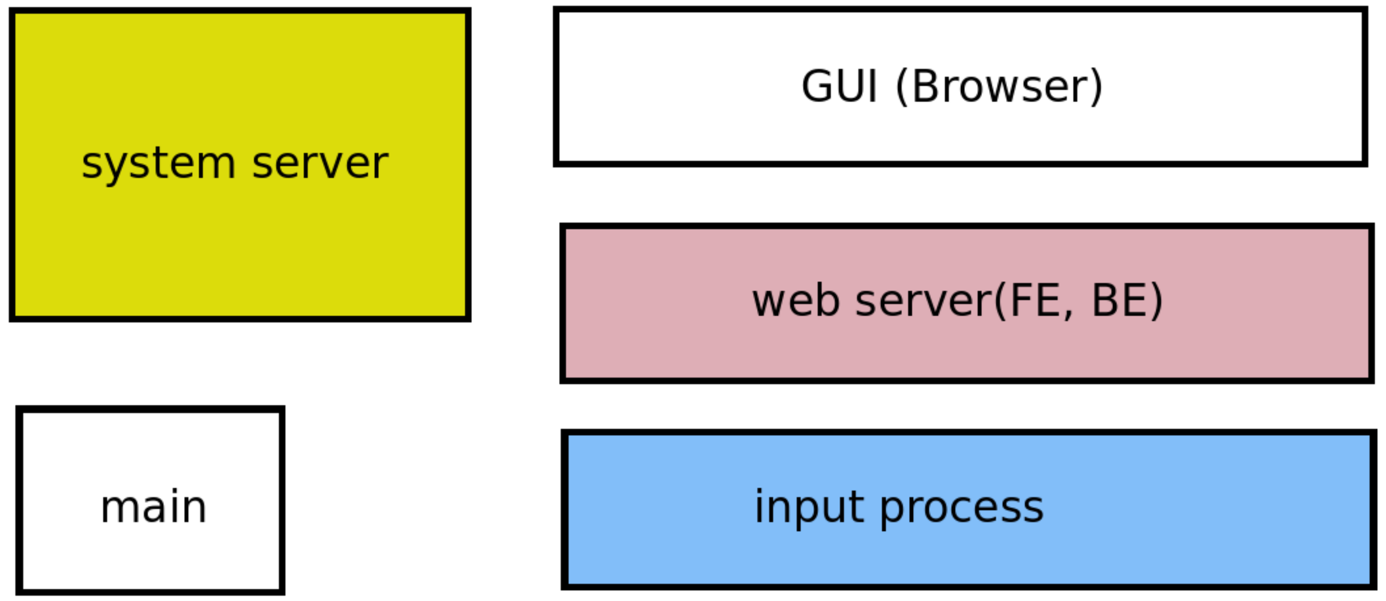
지난 토이 프로젝트 과제


system server
main
GUI (Browser)
web server(FE, BE)
input process

시그널

- 프로세스에게 이벤트가 발생했음을 알림.
  - 프로세스간 정보 전달용 알림.
  - OS가 알려주는 소프트웨어 인터럽트

토이 프로젝트 - 시그널 이용



시그널 핸들러

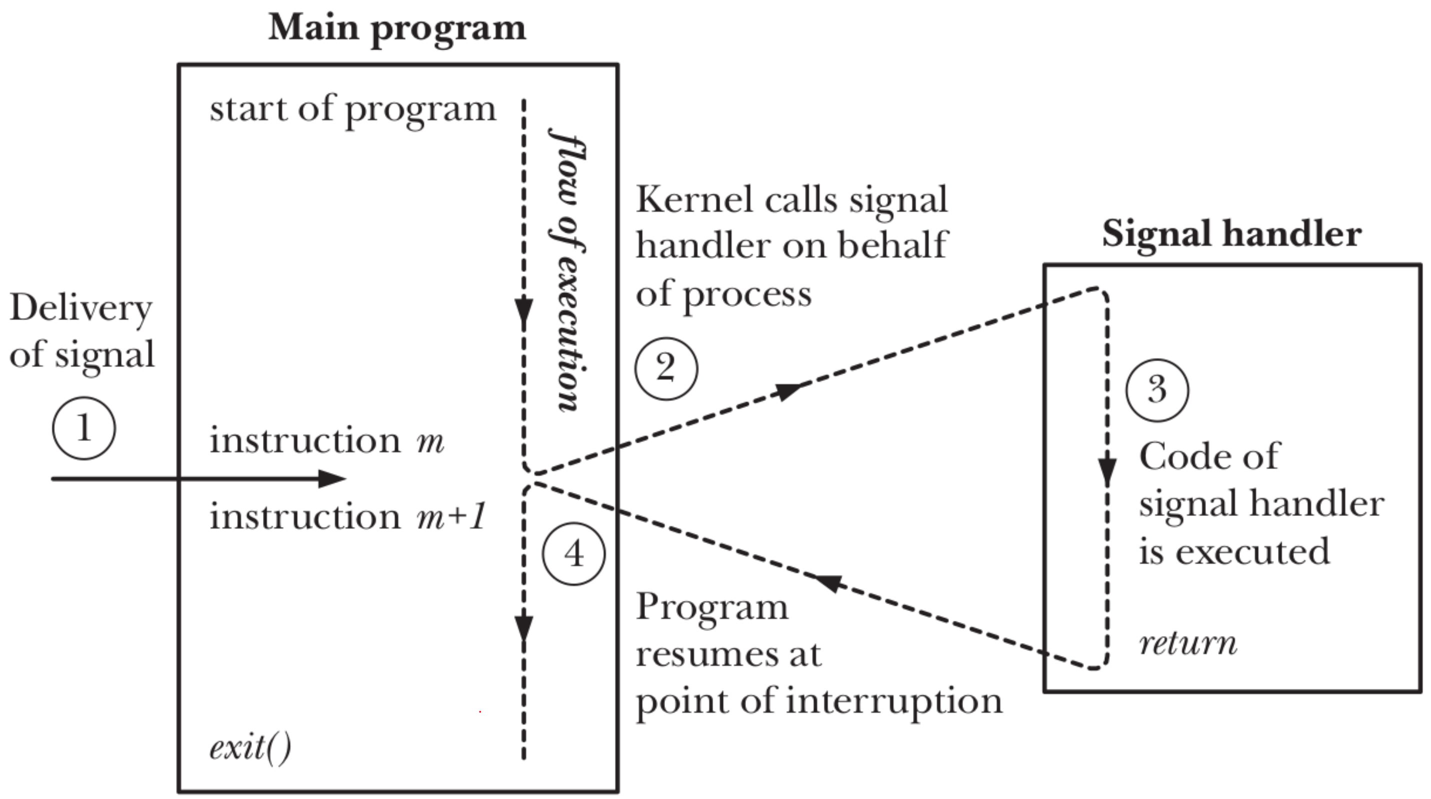
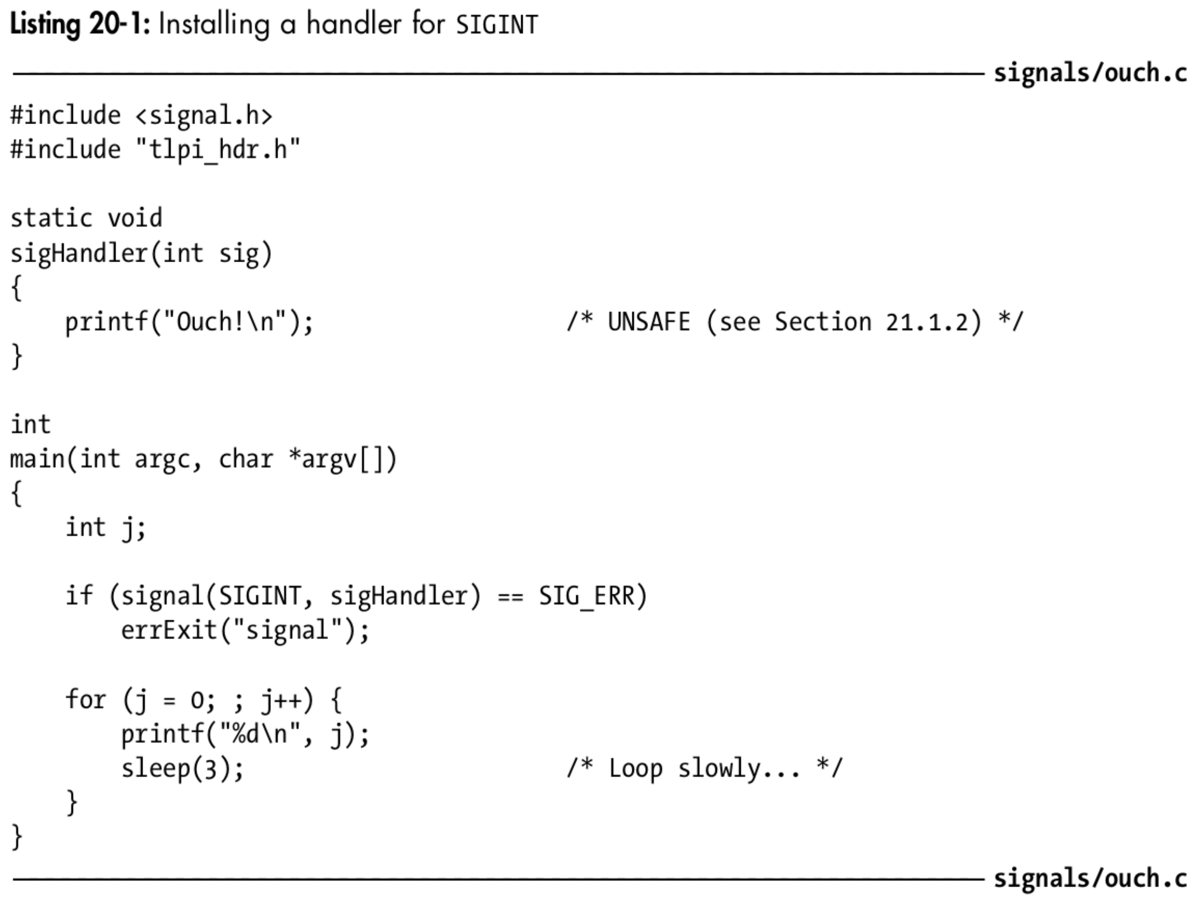

Main program
start of program
flow of execution
Delivery of signal
1
instruction m
instruction m+1
exit()
Kernel calls signal handler on behalf of process
2
Signal handler
3
Code of signal handler is executed
return
4
Program resumes at point of interruption

Figure 20-1: Signal delivery and handler execution

OS -> 프로세스



Listing 20-1: Installing a handler for SIGINT

```
                                                                    signals/ouch.c
#include <signal.h>
#include "tlpi_hdr.h"

static void
sigHandler(int sig)
{
    printf("Ouch!\n");                  /* UNSAFE (see Section 21.1.2) */
}

int
main(int argc, char *argv[])
{
    int j;

    if (signal(SIGINT, sigHandler) == SIG_ERR)
        errExit("signal");

    for (j = 0; ; j++) {
        printf("%d\n", j);
        sleep(3);                       /* Loop slowly... */
    }
}
                                                                    signals/ouch.c
```

control-C (^C)를 통해 SIGINT 생성

프로세스 -> 프로세스

```
#include <signal.h>

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

Returns 0 on success, or –1 on error

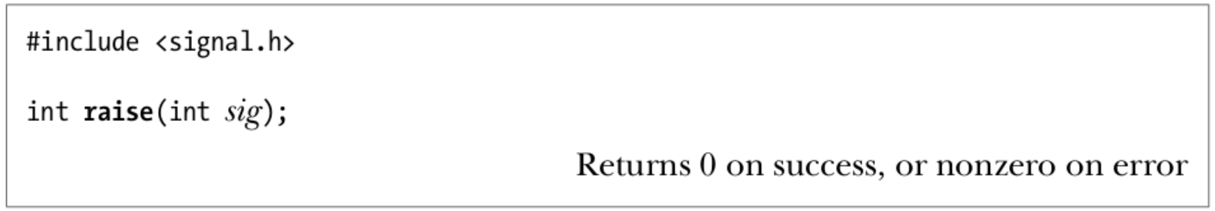
signals/t_kill.c

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

int
main(int argc, char *argv[])
{
    int s, sig;

    if (argc != 3 || strcmp(argv[1], "--help") == 0)
        usageErr("%s sig-num pid\n", argv[0]);

    sig = getInt(argv[2], 0, "sig-num");

    s = kill(getLong(argv[1], 0, "pid"), sig);

    if (sig != 0) {
        if (s == -1)
            errExit("kill");

    } else {                    /* Null signal: process existence check */
        if (s == 0) {
            printf("Process exists and we can send it a signal\n");
        } else {
            if (errno == EPERM)
                printf("Process exists, but we don't have "
                       "permission to send it a signal\n");
            else if (errno == ESRCH)
                printf("Process does not exist\n");
            else
                errExit("kill");
        }
    }

    exit(EXIT_SUCCESS);
}
```

signals/t_kill.c

시그널

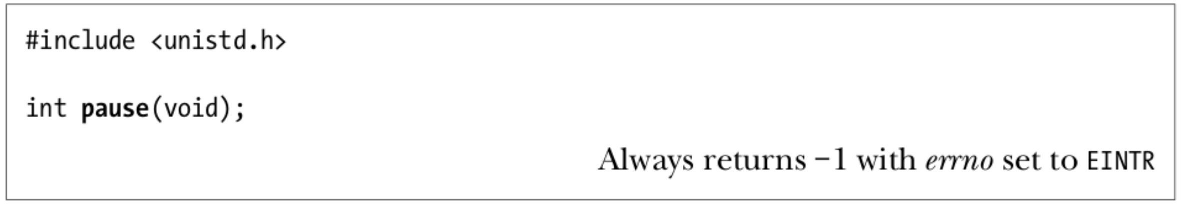
- 자신에게 보낼 때
  - raise
  - 가끔 필요



```
#include <signal.h>

int raise(int sig);
```

Returns 0 on success, or nonzero on error

- 시그널 대기
  - pause 사용



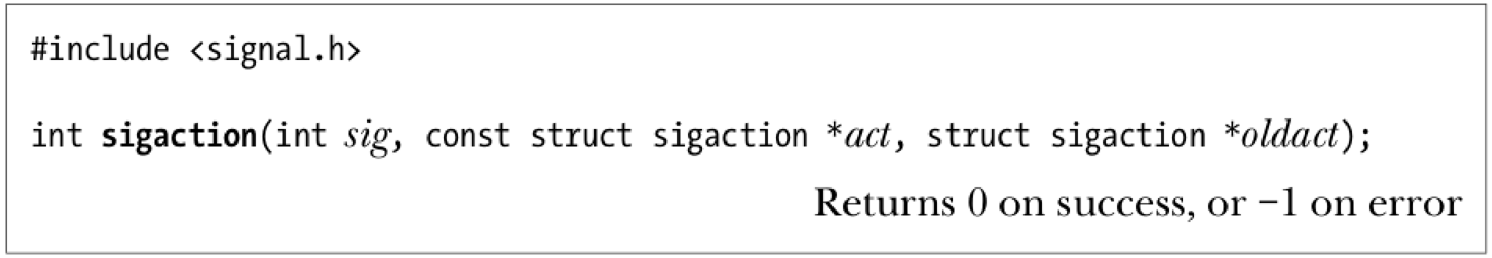
```
#include <unistd.h>

int pause(void);
```

Always returns –1 with *errno* set to EINTR

시그널 타입

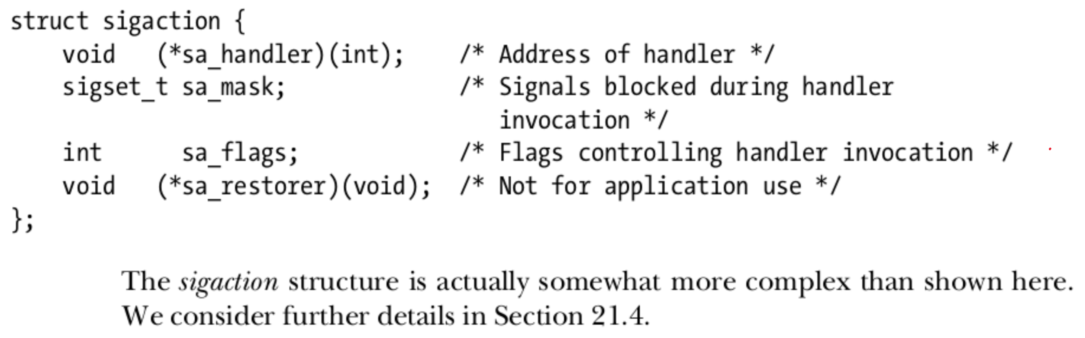
Name	Signal number	Description	SUSv3	Default
SIGABRT	6	Abort process	•	core
SIGALRM	14	Real-time timer expired	•	term
SIGBUS	7 (SAMP=10)	Memory access error	•	core
SIGCHLD	17 (SA=20, MP=18)	Child terminated or stopped	•	ignore
SIGCONT	18 (SA=19, M=25, P=26)	Continue if stopped	•	cont
SIGEMT	undef (SAMP=7)	Hardware fault		term
SIGFPE	8	Arithmetic exception	•	core
SIGHUP	1	Hangup	•	term
SIGILL	4	Illegal instruction	•	core
SIGINT	2	Terminal interrupt	•	term
SIGIO / SIGPOLL	29 (SA=23, MP=22)	I/O possible	•	term
SIGKILL	9	Sure kill	•	term
SIGPIPE	13	Broken pipe	•	term
SIGPROF	27 (M=29, P=21)	Profiling timer expired	•	term
SIGPWR	30 (SA=29, MP=19)	Power about to fail		term
SIGQUIT	3	Terminal quit	•	core
SIGSEGV	11	Invalid memory reference	•	core
SIGSTKFLT	16 (SAM=undef, P=36)	Stack fault on coprocessor		term
SIGSTOP	19 (SA=17, M=23, P=24)	Sure stop	•	stop
SIGSYS	31 (SAMP=12)	Invalid system call	•	core
SIGTERM	15	Terminate process	•	term
SIGTRAP	5	Trace/breakpoint trap	•	core
SIGTSTP	20 (SA=18, M=24, P=25)	Terminal stop	•	stop
SIGTTIN	21 (M=26, P=27)	Terminal read from BG	•	stop
SIGTTOU	22 (M=27, P=28)	Terminal write from BG	•	stop
SIGURG	23 (SA=16, M=21, P=29)	Urgent data on socket	•	ignore
SIGUSR1	10 (SA=30, MP=16)	User-defined signal 1	•	term
SIGUSR2	12 (SA=31, MP=17)	User-defined signal 2	•	term
SIGVTALRM	26 (M=28, P=20)	Virtual timer expired	•	term
SIGWINCH	28 (M=20, P=23)	Terminal window size change		ignore
SIGXCPU	24 (M=30, P=33)	CPU time limit exceeded	•	core
SIGXFSZ	25 (M=31, P=34)	File size limit exceeded	•	core

시그널 속성 변경



```
#include <signal.h>

int sigaction(int sig, const struct sigaction *act, struct sigaction *oldact);
```

Returns 0 on success, or –1 on error



```
struct sigaction {
    void   (*sa_handler)(int);    /* Address of handler */
    sigset_t sa_mask;             /* Signals blocked during handler
                                     invocation */
    int      sa_flags;            /* Flags controlling handler invocation */
    void   (*sa_restorer)(void);  /* Not for application use */
};
```

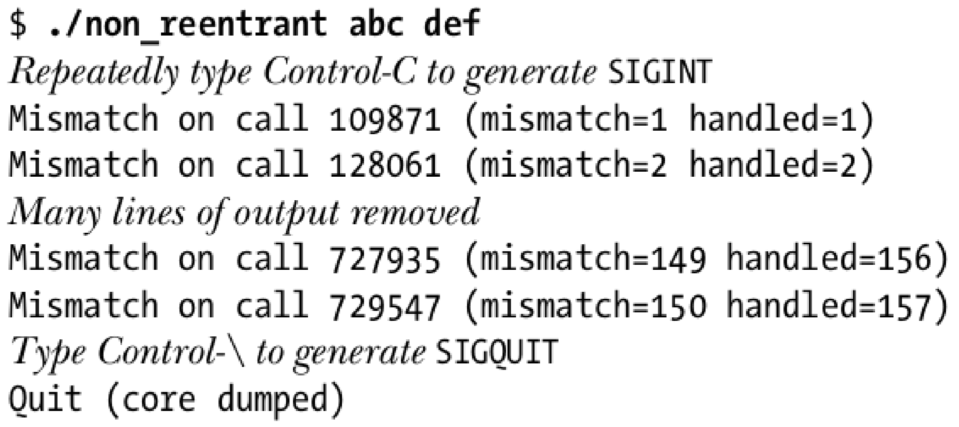
The *sigaction* structure is actually somewhat more complex than shown here. We consider further details in Section 21.4.

시그널 특징

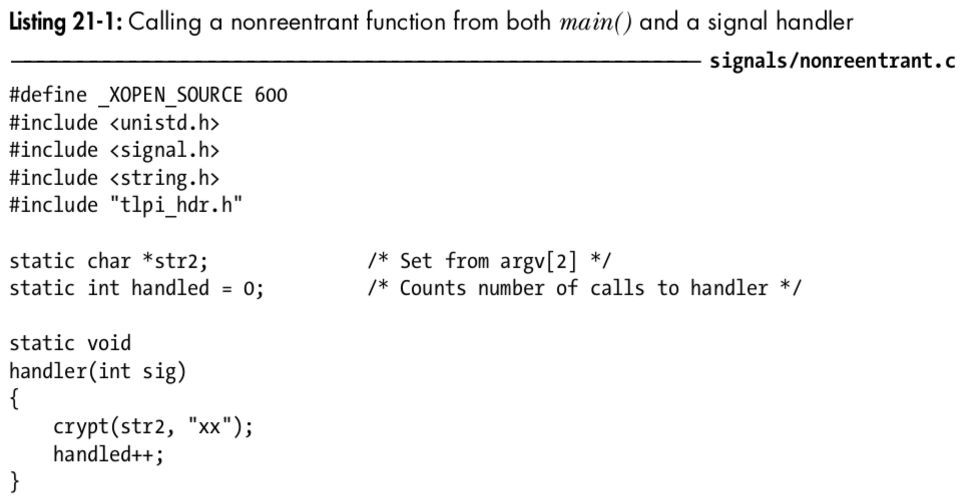
- 시그널은 큐에 들어가지 않는다.
  - 누적되어도 여러 번 호출 X

- 전역 변수 조심히 사용해야 함
  - 재진입 가능한 함수 사용.

재 진입 불가 예제

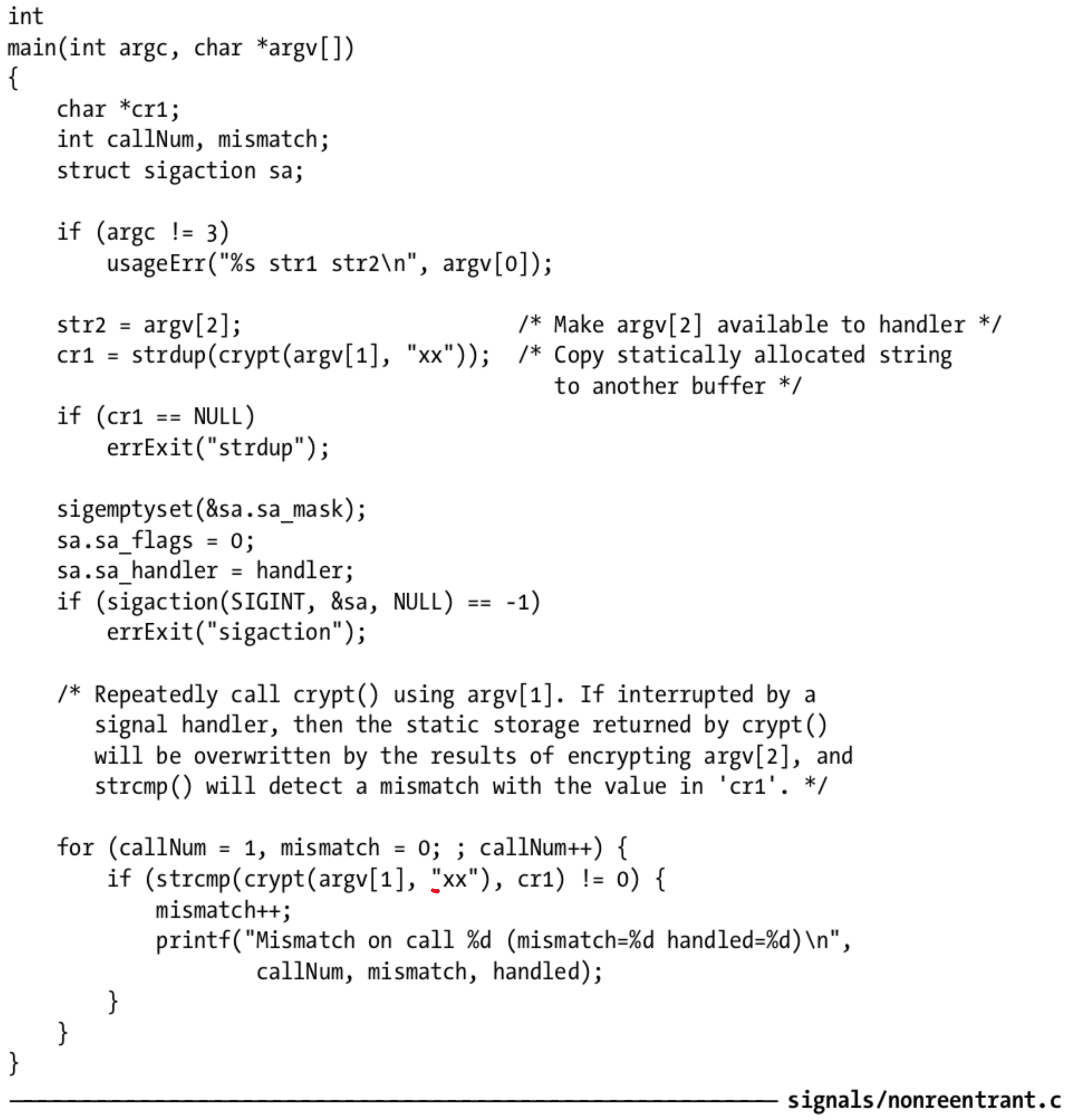


```
$ ./non_reentrant abc def
Repeatedly type Control-C to generate SIGINT
Mismatch on call 109871 (mismatch=1 handled=1)
Mismatch on call 128061 (mismatch=2 handled=2)
Many lines of output removed
Mismatch on call 727935 (mismatch=149 handled=156)
Mismatch on call 729547 (mismatch=150 handled=157)
Type Control-\ to generate SIGQUIT
Quit (core dumped)
```



Listing 21-1: Calling a nonreentrant function from both *main()* and a signal handler

```
——————————————————————————————————————————————— signals/nonreentrant.c
#define _XOPEN_SOURCE 600
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include "tlpi_hdr.h"

static char *str2;              /* Set from argv[2] */
static int handled = 0;         /* Counts number of calls to handler */

static void
handler(int sig)
{
    crypt(str2, "xx");
    handled++;
}
```

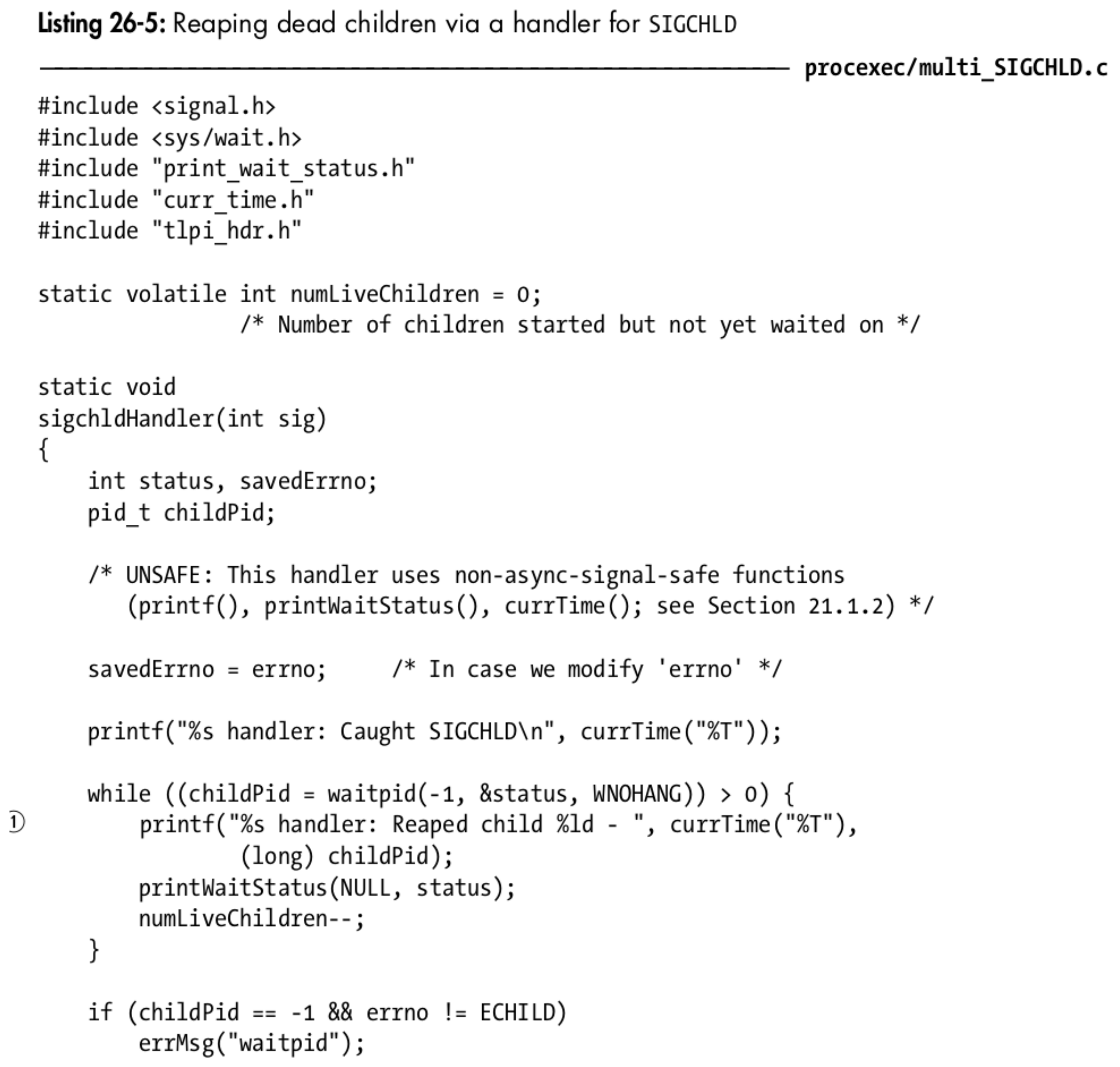


```
int
main(int argc, char *argv[])
{
    char *cr1;
    int callNum, mismatch;
    struct sigaction sa;

    if (argc != 3)
        usageErr("%s str1 str2\n", argv[0]);

    str2 = argv[2];                         /* Make argv[2] available to handler */
    cr1 = strdup(crypt(argv[1], "xx"));     /* Copy statically allocated string
                                               to another buffer */
    if (cr1 == NULL)
        errExit("strdup");

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

    /* Repeatedly call crypt() using argv[1]. If interrupted by a
       signal handler, then the static storage returned by crypt()
       will be overwritten by the results of encrypting argv[2], and
       strcmp() will detect a mismatch with the value in 'cr1'. */

    for (callNum = 1, mismatch = 0; ; callNum++) {
        if (strcmp(crypt(argv[1], "xx"), cr1) != 0) {
            mismatch++;
            printf("Mismatch on call %d (mismatch=%d handled=%d)\n",
                    callNum, mismatch, handled);
        }
    }
}
——————————————————————————————————————————————— signals/nonreentrant.c
```

The SIGCHLD Signal

- 자식의 시그널을 부모가 받을 수 있음
- 언제 사용?
  - 예외 처리
  - 시스템의 연속성을 위해서 자식 프로세스를 다시 살려야 함.

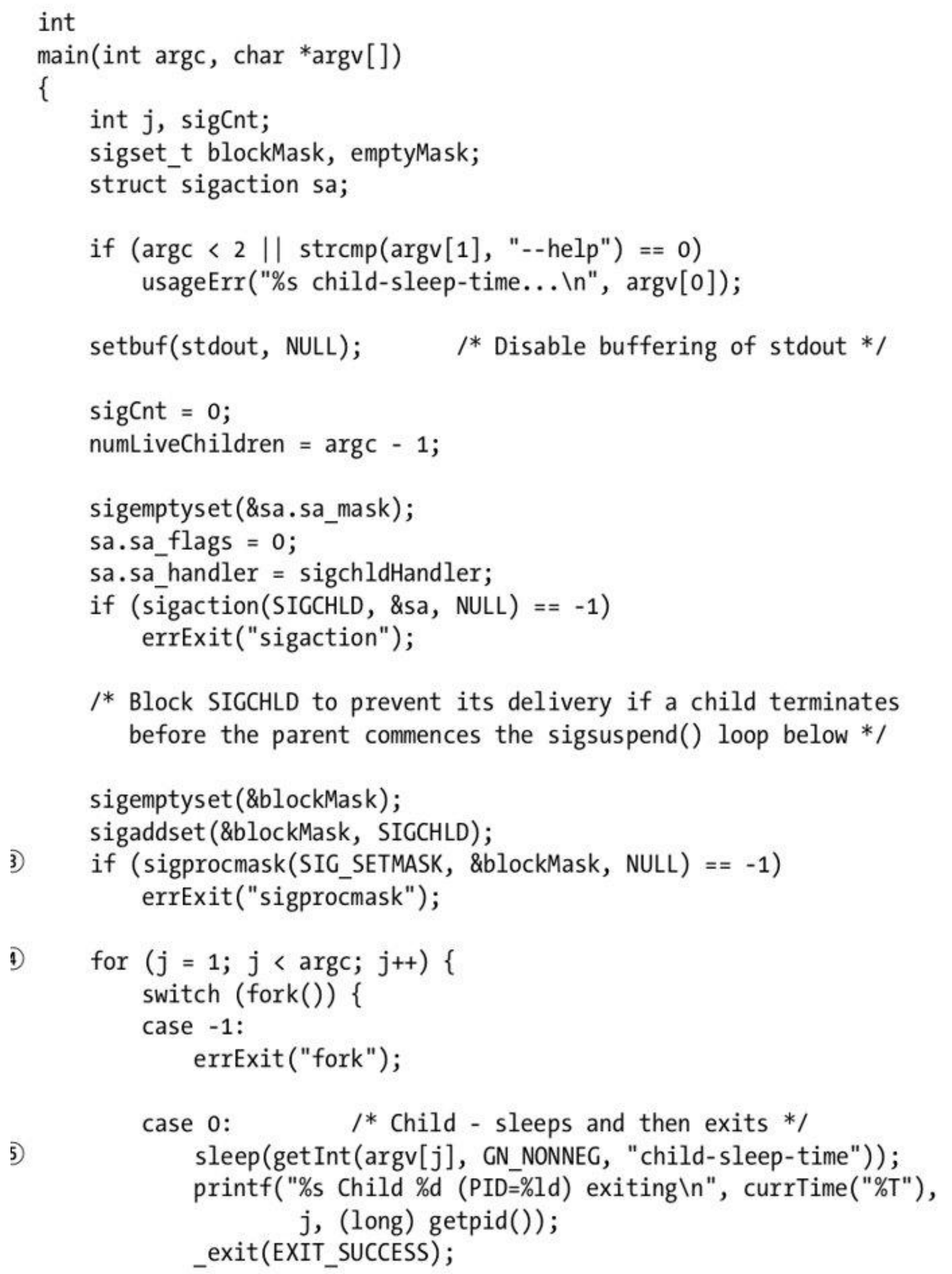
The SIGCHLD Signal



Listing 26-5: Reaping dead children via a handler for SIGCHLD

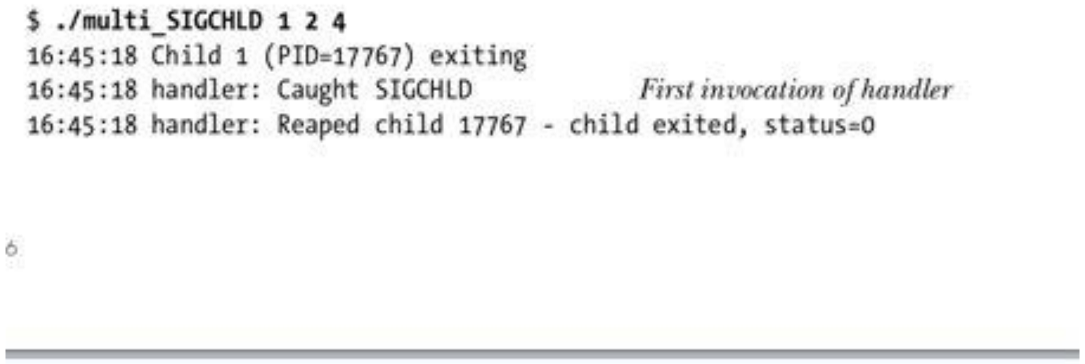
procexec/multi_SIGCHLD.c

```
#include <signal.h>
#include <sys/wait.h>
#include "print_wait_status.h"
#include "curr_time.h"
#include "tlpi_hdr.h"

static volatile int numLiveChildren = 0;
                /* Number of children started but not yet waited on */

static void
sigchldHandler(int sig)
{
    int status, savedErrno;
    pid_t childPid;

    /* UNSAFE: This handler uses non-async-signal-safe functions
       (printf(), printWaitStatus(), currTime(); see Section 21.1.2) */

    savedErrno = errno;     /* In case we modify 'errno' */

    printf("%s handler: Caught SIGCHLD\n", currTime("%T"));

    while ((childPid = waitpid(-1, &status, WNOHANG)) > 0) {
①       printf("%s handler: Reaped child %ld - ", currTime("%T"),
                (long) childPid);
        printWaitStatus(NULL, status);
        numLiveChildren--;
    }

    if (childPid == -1 && errno != ECHILD)
        errMsg("waitpid");
```



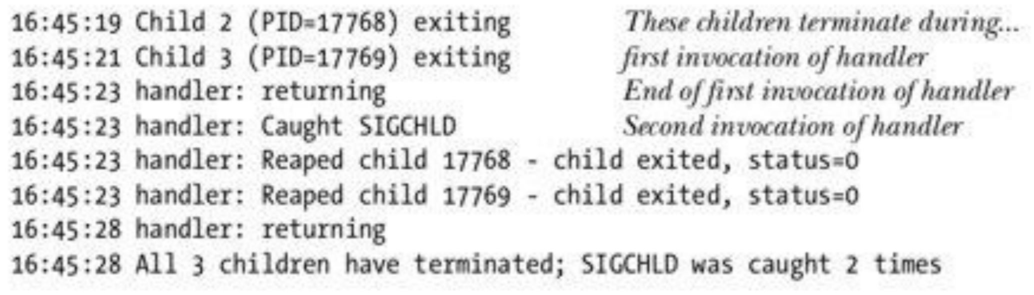
```
int
main(int argc, char *argv[])
{
    int j, sigCnt;
    sigset_t blockMask, emptyMask;
    struct sigaction sa;

    if (argc < 2 || strcmp(argv[1], "--help") == 0)
        usageErr("%s child-sleep-time...\n", argv[0]);

    setbuf(stdout, NULL);       /* Disable buffering of stdout */

    sigCnt = 0;
    numLiveChildren = argc - 1;

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

    /* Block SIGCHLD to prevent its delivery if a child terminates
       before the parent commences the sigsuspend() loop below */

    sigemptyset(&blockMask);
    sigaddset(&blockMask, SIGCHLD);
③   if (sigprocmask(SIG_SETMASK, &blockMask, NULL) == -1)
        errExit("sigprocmask");

④   for (j = 1; j < argc; j++) {
        switch (fork()) {
        case -1:
            errExit("fork");

        case 0:         /* Child - sleeps and then exits */
⑤           sleep(getInt(argv[j], GN_NONNEG, "child-sleep-time"));
            printf("%s Child %d (PID=%ld) exiting\n", currTime("%T"),
                    j, (long) getpid());
            _exit(EXIT_SUCCESS);
```

The SIGCHLD Signal



```
$ ./multi_SIGCHLD 1 2 4
16:45:18 Child 1 (PID=17767) exiting
16:45:18 handler: Caught SIGCHLD             First invocation of handler
16:45:18 handler: Reaped child 17767 - child exited, status=0
```

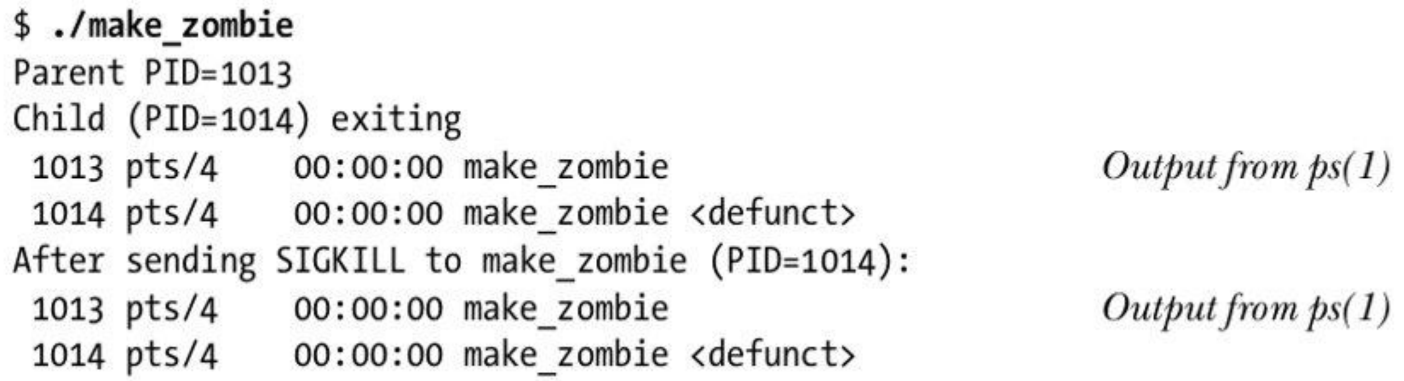
6



```
16:45:19 Child 2 (PID=17768) exiting         These children terminate during...
16:45:21 Child 3 (PID=17769) exiting         first invocation of handler
16:45:23 handler: returning                  End of first invocation of handler
16:45:23 handler: Caught SIGCHLD             Second invocation of handler
16:45:23 handler: Reaped child 17768 - child exited, status=0
16:45:23 handler: Reaped child 17769 - child exited, status=0
16:45:28 handler: returning
16:45:28 All 3 children have terminated; SIGCHLD was caught 2 times
```

고아와 좀비 프로세스

- 자식 프로세스는 종료하면 좀비 상태로 변경됨
- 부모 -> wait 함수 호출 안하면?
  - 좀비로 남음



```
$ ./make_zombie
Parent PID=1013
Child (PID=1014) exiting
 1013 pts/4    00:00:00 make_zombie                    Output from ps(1)
 1014 pts/4    00:00:00 make_zombie <defunct>
After sending SIGKILL to make_zombie (PID=1014):
 1013 pts/4    00:00:00 make_zombie                    Output from ps(1)
 1014 pts/4    00:00:00 make_zombie <defunct>
```

고아와 좀비 프로세스

Listing 26-4: Creating a zombie child process

```
——————————————————————————————————————————————————— procexec/make_zombie.c
#include <signal.h>
#include <libgen.h>             /* For basename() declaration */
#include "tlpi_hdr.h"

#define CMD_SIZE 200

int
main(int argc, char *argv[])
{
    char cmd[CMD_SIZE];
    pid_t childPid;

    setbuf(stdout, NULL);       /* Disable buffering of stdout */

    printf("Parent PID=%ld\n", (long) getpid());

    switch (childPid = fork()) {
    case -1:
        errExit("fork");

    case 0:     /* Child: immediately exits to become zombie */
        printf("Child (PID=%ld) exiting\n", (long) getpid());
        _exit(EXIT_SUCCESS);

    default:    /* Parent */
        sleep(3);               /* Give child a chance to start and exit */
        snprintf(cmd, CMD_SIZE, "ps | grep %s", basename(argv[0]));
        cmd[CMD_SIZE - 1] = '\0';       /* Ensure string is null-terminated */
        system(cmd);            /* View zombie child */

        /* Now send the "sure kill" signal to the zombie */

        if (kill(childPid, SIGKILL) == -1)
            errMsg("kill");
        sleep(3);               /* Give child a chance to react to signal */
        printf("After sending SIGKILL to zombie (PID=%ld):\n", (long) childPid);
        system(cmd);            /* View zombie child again */

        exit(EXIT_SUCCESS);
    }
}
——————————————————————————————————————————————————— procexec/make_zombie.c
```

시그널

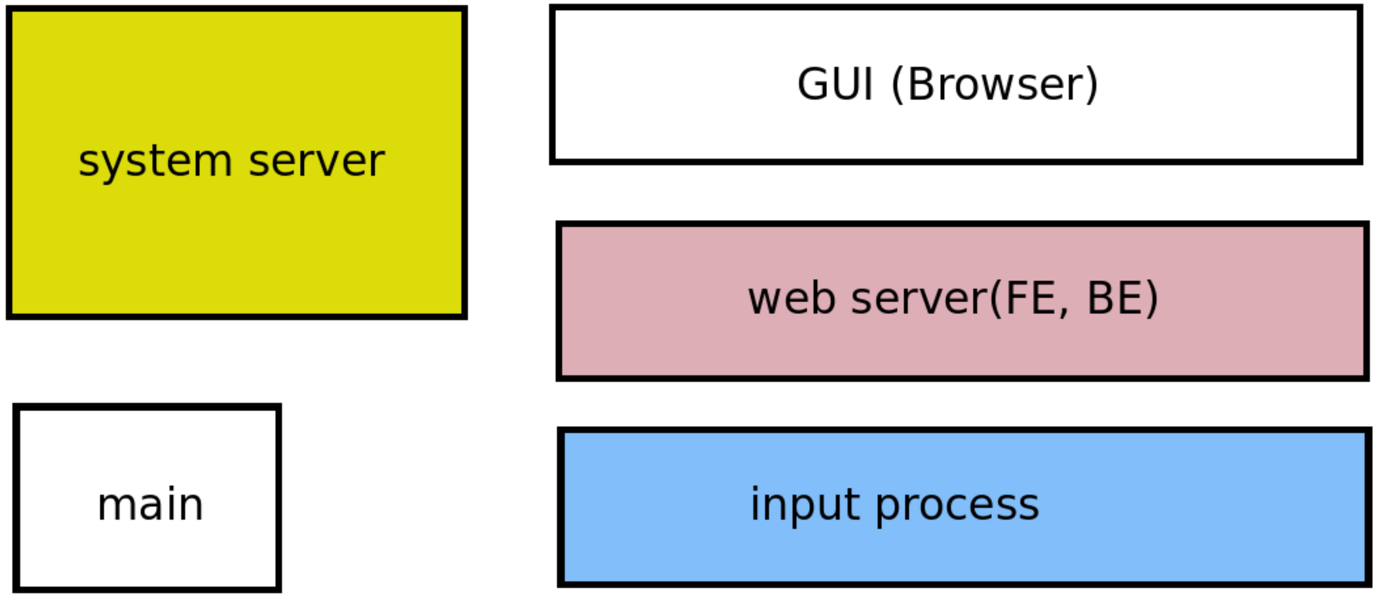
- Real-world 시그널
  - Exception(crash) handler로 중요
    - 운영중인 시스템에 오류 발생 시 디버깅 용도로 굉장히 중요함.
  - 프로그램 오류 발생 시 예외 처리
    - Call stack 저장
    - 예) 안드로이드: https://source.android.com/docs/core/tests/debug/native-crash?hl=ko
  - 타이머 시그널
    - 타이머 인터럽트로 활용.
    - delayed_timeout
  - USER 시그널
    - 다목적 (동적인 로그 출력)

실습 코드 분석

- vscode debugger로 디버깅
- tlpi-dist/signals/ouch.c
  - 코드 분석 및 실행
- tlpi-dist/signals/t_kill.c
  - 코드 분석 및 실행
- tlpi-dist/procexec/make_zombie.c
  - 코드 분석 및 실행
- tlpi-dist/procexec/multi_SIGCHLD.c
  - 코드 분석 및 실행

토이 프로젝트 - 시그널

- 시그널 구현 실습
- Input process seg falut 처리 핸들러 구현
- main process 자식 프로세스 시그널 출력


system server
GUI (Browser)
web server(FE, BE)
main
input process