

8. Homework

(Do not use `^z` when experimenting with signals. `^z` will stop the process, and the process cannot receive further signals.)

1) Try to kill `ex1` and `ex2` in Section 1 with `^c`. What happens? You cannot kill `ex2` with `^c`, just leave it and go to homework 1-1).

[ex1]

```
kyumin@DESKTOP-NUDFAPK ~  
$ ex1 | ps -ef  
  UID      PID     PPID  TTY          TIME COMMAND  
  kyumin   1687      1666  pty0        15:48:06 /cygdrive/c/Users/kyumin/AppData/Roam  
ing/SPB_Data/ex1  
  kyumin   1688      1666  pty0        15:48:06 /usr/bin/ps  
  kyumin   1666      1665  pty0        15:44:07 /usr/bin/bash  
  kyumin   1665         1  ?          15:44:07 /usr/bin/mintty  
  
kyumin@DESKTOP-NUDFAPK ~  
$ ps -ef  
  UID      PID     PPID  TTY          TIME COMMAND  
  kyumin   1666      1665  pty0        15:44:07 /usr/bin/bash  
  kyumin   1665         1  ?          15:44:07 /usr/bin/mintty  
  kyumin   1689      1666  pty0        15:48:13 /usr/bin/ps
```

프로세스가 종료된다.

[ex2]

```
kyumin@DESKTOP-NUDFAPK ~  
$ ex2  
I am ok  
|
```

프로세스가 종료되지 않고, `foo` 함수가 실행되므로 메시지가 출력된다. 이후 `for`문을 계속 돈다.

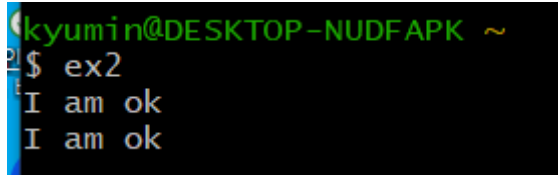
1-1) Open another putty window and find the pid of ex2 by

```
$ ps -ef | grep 12345
```

where 12345 is your student ID. Suppose ex2's pid is 334455. Send SIGINT (signal number 2) to ex2 with

```
$ kill -2 334455
```

What happens? Explain the result.



```
kyumin@DESKTOP-NUDFAPK ~  
$ ex2  
I am ok  
I am ok
```

Ctrl + c와 동일하게 foo 함수가 실행되므로 메시지가 출력된다.

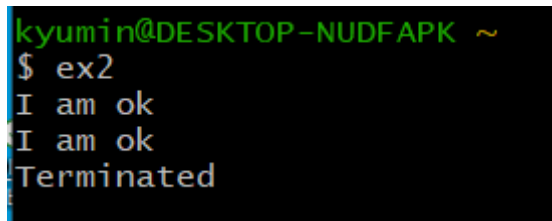
2) You can kill ex2 as follows:

open another terminal

find the pid of ex2, e.g. 1234

kill 1234 (or “kill -15 1234”)

Why SIGINT (signal 2) cannot kill ex2 while SIGTERM (signal 15) can?



```
kyumin@DESKTOP-NUDFAPK ~  
$ ex2  
I am ok  
I am ok  
Terminated
```

시그널 15번은 pid를 kill하는 명령어이므로 프로세스가 종료되었다.

시그널 2번은 핸들러가 있다면 대비해둔 내용을 실행하고, 없다면 프로그램을 종료하는 것이다. Ex1의 경우 대비가 없었으므로 종료된 것이고, ex2의 경우 foo를 실행하기로 대비가 되어있었으므로 메시지가 출력된 것이다. 그러나 foo를 실행한 후 다시 for문으로 돌아간다.

2-1) Change ex2.c as follows.

```
#include <stdio.h>
#include <signal.h>
void foo(int signum){
    printf("I am ok\n");
}
void main(){
    signal(2, foo);    // prepare for signal number 2. same as signal(SIGINT, foo)
    signal(15, foo);
    for(;;);
}
```

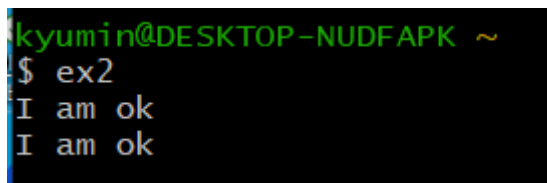
Now compile and run ex2 and try to kill it with SIGINT and SIGTERM in another window.

Assume ex2 has pid 334455.

```
$ kill -2 334455
```

```
$ kill -15 334455
```

Explain the result.

A terminal window with a black background and green text. The prompt is 'kyumin@DESKTOP-NUDFAPK ~'. The user enters '\$ ex2'. The program outputs 'I am ok' twice, on two separate lines. The terminal is currently showing the first 'I am ok' output.

```
kyumin@DESKTOP-NUDFAPK ~
$ ex2
I am ok
I am ok
```

Kill -2 1717 / kill -15 1717 / kill 1717을 해봤지만 모두 동일하게 메시지만 출력되고 종료 되지 않는다. 시그널 2번은 이미 대비를 해줬으므로 foo가 실행되는 것이다. 시그널 15번 또한 대비해줬으므로 foo가 실행되는 것이다. kill -15 1717 과 kill 1717은 동일한 명령어이다. 그러므로 동일한 결과가 나온다.

3) You can also kill ex2 by closing the terminal where ex2 is running. Confirm this by first finding out the pid of ex2 with "ps -ef | grep 1234" (assuming your student id is 1234) in another terminal, close the terminal where ex2 is currently running, and find this pid again with "ps -ef | grep 1234" in the second window. Explain why ex2 dies in this case.

```
kyumin@DESKTOP-NUDFAPK ~  
$ ps -ef  
  UID      PID  PPID  TTY      STIME COMMAND  
  kyumin    1693    1692  pty1    15:48:44 /usr/bin/bash  
  kyumin    1717    1666  pty0    16:09:44 /cygdrive/c/Users/kyumin/AppData/Roam  
ing/SPB_Data/ex2  
  kyumin    1666    1665  pty0    15:44:07 /usr/bin/bash  
  kyumin    1721    1693  pty1    16:17:54 /usr/bin/ps  
  kyumin    1692         1  ?      15:48:44 /usr/bin/mintty  
  kyumin    1665         1  ?      15:44:07 /usr/bin/mintty  
  
kyumin@DESKTOP-NUDFAPK ~  
$ ps -ef  
  UID      PID  PPID  TTY      STIME COMMAND  
  kyumin    1693    1692  pty1    15:48:44 /usr/bin/bash  
  kyumin    1722    1693  pty1    16:18:00 /usr/bin/ps  
  kyumin    1692         1  ?      15:48:44 /usr/bin/mintty
```

Ex2를 실행 중이던 터미널을 닫으니 ex2 프로세스가 사라졌다.

이는 부모 프로세스(1666)가 종료되면 좀비 상태인 자식 프로세스(1717)는 SIGHUP신호를 받는다. 그러면 좀비상태가 되고, 부모가 없어졌으므로 init 프로세스(1번 pid)로 입양된다. 이 때 1번 프로세스는 계속 wait을 보내므로 좀비를 없애는 역할을 한다. 그래서 자식 프로세스가 사라진다.

4) How can you prevent ex2 from being killed even when you close the terminal?

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

void foo(int signum){
    printf("I am ok\n");
}

void main(){
    signal(2, foo);
    signal(15, foo);
    signal(1, foo);
    for(;;);
}
```

SIGHUP 신호에 대해 대비를 한다면 프로세스가 종료되지 않도록 막을 수 있다. 시그널 1 번에 대한 대비로 foo가 실행되도록 했다.

```
kyumin@DESKTOP-NUDFAPK ~
$ ps -ef
  UID      PID  PPID  TTY      STIME COMMAND
  kyumin    1693   1692  pty1    15:48:44 /usr/bin/bash
  kyumin    1738   1693  pty1    18:53:55 /usr/bin/ps
  kyumin    1735     1  pty0    18:53:35 /cygdrive/c/Users/kyumin/AppData/Roam
ing/SPB_Data/ex2
  kyumin    1692     1  ?       15:48:44 /usr/bin/mintty
```

터미널을 종료하고 다른 터미널로 실행 중인 프로세스를 확인해보니 ex2가 여전히 살아있는 것을 볼 수 있다.

5) If you were successful in problem 4 in preventing ex2 from being killed, how can you still kill ex2?

```
kyumin@DESKTOP-NUDFAPK ~  
$ ps -ef  
    UID      PID     PPID  TTY          STIME COMMAND  
    kyumin    1693     1692 pty1      15:48:44 /usr/bin/bash  
    kyumin    1750         1 pty0      10:00:50 /cygdrive/c/User  
ing/SPB_Data/ex2  
    kyumin    1752     1693 pty1      10:00:57 /usr/bin/ps  
    kyumin    1692         1 ?         15:48:44 /usr/bin/mintty  
  
kyumin@DESKTOP-NUDFAPK ~  
$ kill -9 1750  
  
kyumin@DESKTOP-NUDFAPK ~  
$ ps -ef  
    UID      PID     PPID  TTY          STIME COMMAND  
    kyumin    1693     1692 pty1      15:48:44 /usr/bin/bash  
    kyumin    1753     1693 pty1      10:01:10 /usr/bin/ps  
    kyumin    1692         1 ?         15:48:44 /usr/bin/mintty
```

시그널 9번을 보내면 된다. 9번은 대비를 하더라도 프로세스가 무조건 종료되기 때문이다. 또한 9번이 아니더라도 대비하지 않은 시그널 번호를 보내도 된다. 대비하지 않은 시그널을 받으면 프로세스는 종료되기 때문이다.

6) Write a program (ex3.c) that forks two children as below. All three processes (the parent and the two children) run infinite loops. Run this program and find the pids of the three processes in another window. Kill the parent ("kill -15 parent-pid") and observe what happens to the children.

```

.....
x = fork();
if (x==0) for(;;); // first child
else{
    y=fork();
    if (y==0) for(;;); // second child
}
for(;;); // parent
.....

```

The terminal screenshot shows the execution of the program and the effect of killing the parent process. It is divided into three sections:

Section 1: Initial state

```

kyumin@DESKTOP-NUDFAPK ~
$ ps -ef

```

| UID | PID | PPID | TTY | STIME | COMMAND |
|--------|------|------|------|----------|---------------------------------------|
| kyumin | 1693 | 1692 | pty1 | 15:48:44 | /usr/bin/bash |
| kyumin | 1768 | 1764 | pty0 | 10:39:16 | /cygdrive/c/Users/kyumin/AppData/Roam |
| kyumin | 1770 | 1768 | pty0 | 10:39:17 | /cygdrive/c/Users/kyumin/AppData/Roam |
| kyumin | 1771 | 1693 | pty1 | 10:39:20 | /usr/bin/ps |
| kyumin | 1769 | 1768 | pty0 | 10:39:16 | /cygdrive/c/Users/kyumin/AppData/Roam |
| kyumin | 1763 | 1 | ? | 10:39:11 | /usr/bin/mintty |
| kyumin | 1692 | 1 | ? | 15:48:44 | /usr/bin/mintty |
| kyumin | 1764 | 1763 | pty0 | 10:39:12 | /usr/bin/bash |

Section 2: Killing the parent

```

kyumin@DESKTOP-NUDFAPK ~
$ kill -15 1768

```

Section 3: State after killing the parent

```

kyumin@DESKTOP-NUDFAPK ~
$ ps -ef

```

| UID | PID | PPID | TTY | STIME | COMMAND |
|--------|------|------|------|----------|---------------------------------------|
| kyumin | 1693 | 1692 | pty1 | 15:48:44 | /usr/bin/bash |
| kyumin | 1772 | 1693 | pty1 | 10:40:09 | /usr/bin/ps |
| kyumin | 1770 | 1 | pty0 | 10:39:17 | /cygdrive/c/Users/kyumin/AppData/Roam |
| kyumin | 1769 | 1 | pty0 | 10:39:16 | /cygdrive/c/Users/kyumin/AppData/Roam |
| kyumin | 1763 | 1 | ? | 10:39:11 | /usr/bin/mintty |
| kyumin | 1692 | 1 | ? | 15:48:44 | /usr/bin/mintty |
| kyumin | 1764 | 1763 | pty0 | 10:39:12 | /usr/bin/bash |

Ex3를 실행하면 부모 프로세스1개, 자식 프로세스 2개가 생긴다. 부모 프로세스를 종료하면 자식 프로세스는 따로 시그널을 받지 않으므로 종료되지 않고, 부모를 잃었으므로 1번 프로세스로 입양된다.

참고로 터미널 종료 시 발생했던 SIGHUP은 터미널을 종료할 때만 발생하는 시그널이고, 부모가 죽어서 발생하는 시그널이 아니다.

7) Same as 6), but this time kill the shell of the terminal where ex3 is running with "kill -9 114455", where 114455 is the pid of the shell. What happens to the three processes (the parent and the two children)? Why the three children die and how can you prevent the three processes from being killed?

```
kyumin@DESKTOP-NUDFAPK ~  
$ ps -ef  
  UID      PID     PPID  TTY          STIME COMMAND  
  kyumin    1693     1692  pty1      15:48:44 /usr/bin/bash  
  kyumin    1808     1806  pty0      10:45:45 /cygdrive/c/Users/kyumin/AppData/Roam  
ing/SPB_Data/ex3  
  kyumin    1802     1801  pty0      10:45:42 /usr/bin/bash  
  kyumin    1807     1806  pty0      10:45:45 /cygdrive/c/Users/kyumin/AppData/Roam  
ing/SPB_Data/ex3  
  kyumin    1809     1693  pty1      10:45:47 /usr/bin/ps  
  kyumin    1806     1802  pty0      10:45:45 /cygdrive/c/Users/kyumin/AppData/Roam  
ing/SPB_Data/ex3  
  kyumin    1801         1  ?         10:45:42 /usr/bin/mintty  
  kyumin    1692         1  ?         15:48:44 /usr/bin/mintty  
  
kyumin@DESKTOP-NUDFAPK ~  
$ kill -9 1801  
  
kyumin@DESKTOP-NUDFAPK ~  
$ ps -ef  
  UID      PID     PPID  TTY          STIME COMMAND  
  kyumin    1693     1692  pty1      15:48:44 /usr/bin/bash  
  kyumin    1810     1693  pty1      10:45:57 /usr/bin/ps  
  kyumin    1692         1  ?         15:48:44 /usr/bin/mintty  
  
kyumin@DESKTOP-NUDFAPK ~
```

터미널의 pid인 1801을 kill하니 ex3에서 생성된 3개의 프로세스가 모두 종료된 것을 볼 수 있다.(bash를 삭제하면 프로세스가 종료되지 않는다. Mintty를 삭제해야한다.) 이는 터미널과의 연결이 끊기면 SIGHUP 시그널이 발생하므로 3개의 프로세스는 바다가 삭제되고 좀비 상태가 된다. 그리고 부모가 없어졌으므로 1번 PID로 입양이 되는데 3개의 프로세스 모두 좀비 상태이고, 1번 PID는 계속 wait을 부르니 3개의 프로세스는 모두 제거된다. 그러므로 3개의 프로세스를 죽지 않게하기 위해선 SIGHUP에 대한 대비를 하면 된다.

8) Same as 6), but this time kill one of the children instead of the parent. What happens to the dead child? Can you remove this dead child (zombie) with signal 9? How can you remove this dead child?

```
kyumin@DESKTOP-NUDFAPK ~  
$ kill 1817  
  
kyumin@DESKTOP-NUDFAPK ~  
$ ps -ef  
UID      PID     PPID  TTY          STIME   COMMAND  
kyumin   1693    1692  ptty1       15:48:44 /usr/bin/bash  
kyumin   1820    1693  ptty1       11:01:02 /usr/bin/ps  
kyumin   1816    1812  ptty0       11:00:35 /cygdrive/c/Users/kyumin/AppData/Roam  
ing/SPB_Data/ex3  
kyumin   1812    1811  ptty0       11:00:33 /usr/bin/bash  
kyumin   1818    1816  ptty0       11:00:35 /cygdrive/c/Users/kyumin/AppData/Roam  
ing/SPB_Data/ex3  
kyumin   1811      1  ?           11:00:33 /usr/bin/mintty  
kyumin   1692      1  ?           15:48:44 /usr/bin/mintty
```

자식을 kill하면 그냥 kill된다. 자식은 좀비상태가 되지 않는다.

9) In problem 8), how can you prevent the generation of a zombie child? Modify your code in 6) to prevent the generation of zombie child. Remember the parent cannot wait in "wait()" since the parent should run its infinite loop too. The parent should prepare a handler for "SIGCHLD" signal and call "wait()" inside this handler.

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

```
int status;
```

```
void foo(int signum)
{
    wait(&status);
}
```

```
void main(){
    int x, y;
    x=fork();
    if (x==0) for(;;); // first child
    else{
        y=fork();
        if (y==0) for(;;); // second child
    }

    signal(17, foo);

    for(;;); // parent
}
```

시그널 17번을 받을 때 foo 함수를 실행하고, foo 함수 내부에는 wait을 호출하도록 만들었다. 이전 문제에서 자식이 좀비 상태가 되지 않고 바로 종료되어 위 코드가 정상적으로 작동하는지는 모르겠다.

10) Modify your FTP server program such that it doesn't generate zombie children.

9번과 동일하게 17번 시그널에 대비하면 된다.

11) Write a program that uses kill() to send a signal to a process.

enter pid and signal number

1234 15

enter pid and signal number

1122 2

.....

Use above program to kill ex2.

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

void main(){
    int pidNum, sigNum;

    for(int i = 0; i < 10; i++){
        printf("enter pid and signal number\n");
        scanf("%d", &pidNum);
        scanf("%d", &sigNum);
        kill(pidNum, sigNum);
    }
}
```

Kill 명령어를 사용해서 입력받은 pid로 signal을 보낸다.

12) Modify your shell such that it can handle & symbol. In a normal shell, the & symbol means "give me prompt without waiting".

ex) Suppose "inf.c" is as follows:

```
int main(){
    for(;;){
        .....
    }
}
```

If you run "inf" without &, the shell will wait until "inf" finishes.

\$./inf

-

--- shell wait here

But if you run it with &, the shell runs "inf" but is ready for next command.

```
$ ./inf&
```

```
$ -- "inf" is running, but you can issue next command
```

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

void main(int argc, char *argv[]){
    int x;
    if (argc > 1 && strcmp(argv[1], "$") == 0) {
        x = fork();
        if(x == 0) kill(getppid(), 15);
    }
    for(;;);
}
```

Inf 실행 시 프로세스는 하나만 존재하며 무한 루프가 돈다. 만약 `inf $`을 실행한다면 자식 프로세스가 생기고, 이 때 부모 프로세스를 킬하면 자식 프로세스는 init프로세스(1번 PID)로 입양이 된다. 즉 해당 터미널은 자식이 없는 상태가 되므로 입력을 할 수 있는 상태가 되고, init 프로세스로 입양된 프로세스는 무한 루프를 돌게된다.

(&이 아닌 \$을 사용한 이유는 &을 사용할 경우 프로세스하나가 자동으로 종료되버리는 문제가 있기 때문에 다른 문자를 사용했다. 문제가 발생하는 정확한 이유는 모르겠다. 기본적으로 셸에서 &을 사용하면 앞의 내용은 백그라운드로 실행하고, 뒤 명령어를 바로 실행하는데 뒤 명령어 없어서 생긴 문제인지..)

```
kyumin@DESKTOP-NUDFAPK ~
$ inf $
Terminated

kyumin@DESKTOP-NUDFAPK ~
$
$ ps -ef
```

| UID | PID | PPID | TTY | STIME | COMMAND |
|--------|------|------|------|----------|---------------------------------------|
| kyumin | 1836 | 1835 | pty0 | 16:10:11 | /usr/bin/bash |
| kyumin | 1924 | 1718 | pty1 | 22:23:49 | /usr/bin/ps |
| kyumin | 1922 | 1 | pty0 | 16:29:06 | /cygdrive/c/Users/kyumin/AppData/Roam |
| kyumin | 1718 | 1717 | pty1 | 15:34:32 | /usr/bin/bash |
| kyumin | 1835 | 1 | ? | 16:10:11 | /usr/bin/mintty |
| kyumin | 1717 | 1 | ? | 15:34:32 | /usr/bin/mintty |

Inf 프로세스는 하나만 존재하며 그 프로세스의 부모 PID는 1번이다. 즉 부모 프로세스는 kill되었고, 자식이 1번 프로세스로 입양된 것이다.

13) Make a daemon which wakes up every 5 seconds and prints hello in a pre-specified file.

[daemon.c]

```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

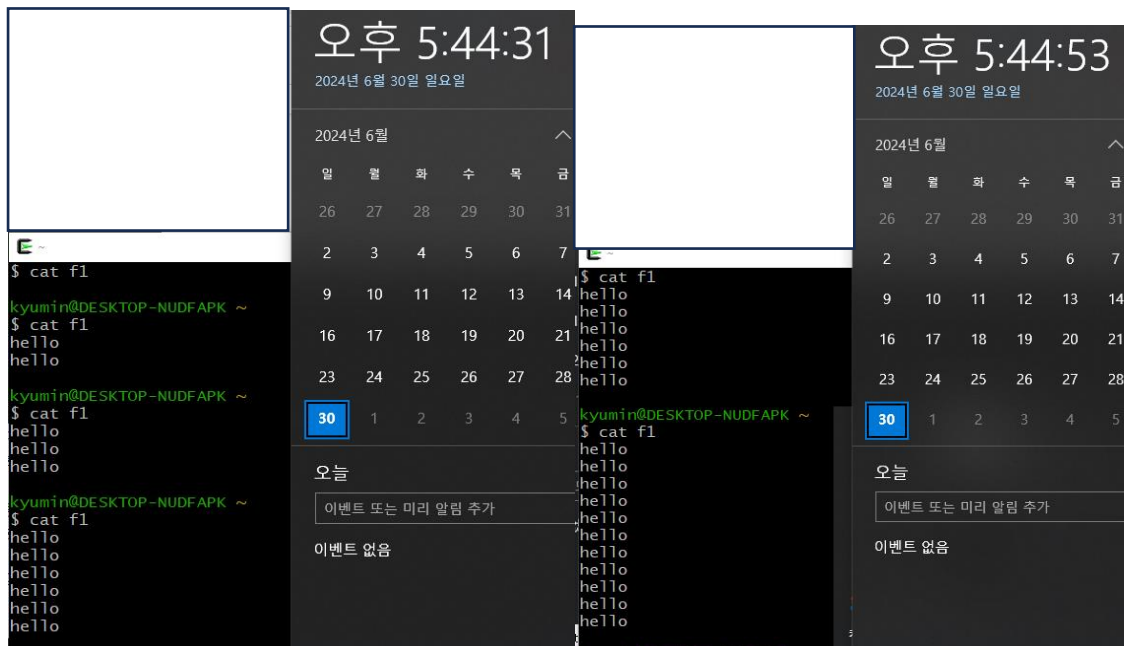
void main(){
    int x, fd;

    //파일 오픈
    x = open("f1", O_RDWR, 00777); // open
    if (x == -1){
        perror("error in open");
    }

    //프로세스 복제 후 부모 프로세스 제거
    fd = fork();
    if(fd == 0){
        kill(getppid(), 15);
    }

    //메시지 쓰기
    for(;;){
        write(x, "hello\n", 6);
        sleep(5);
    }
}
```

이전 문제와 동일하게 부모 프로세스를 kill하여 1번 PID로 입양되도록 만들었다. 이후 f1 파일에 hello 메시지를 출력한다. Sleep 함수를 사용해서 5초마다 반복되도록 만들었다.



F1에는 hello 메시지가 6개였으나 22초가 지난 후 다시 확인해보면 11개가 되었다. 5초마다 hello 메시지가 출력되고 있음을 알 수 있다.

14) Make a program which re-runs automatically when it is killed.

```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int x;

void foo(int signum){
    close(x);
    //현재 프로그램을 다시 실행
    char *str[2];
    str[0] = "/cygdrive/c/Users/kyumin/AppData/Roaming/SPB_Data/daemon";
    str[1] = NULL;
    execve(str[0], str, environ);
}
```

```
void main(){
    int fd;

    printf("start ~ \n");

    //파일 오픈
    x = open("f1", O_RDWR | O_APPEND, 00777);
    if (x == -1){
        perror("error in open");
    }

    //프로세스 복제 후 부모 프로세스 제거
    fd = fork();
    if(fd == 0){
        kill(getppid(), 9);
    }

    //kill 명령어 사용 시 foo 실행
    signal(15, foo);

    //메시지 쓰기
    for(;;){
        write(x, "hello\n", 6);
        sleep(5);
    }
}
```

Kill 명령어 사용 시 위 프로그램은 15번 시그널을 받는다. 그에대한 대비로 foo라는 함수를 만들었다. Foo 함수에서는 execve함수를 사용해서 프로그램을 실행하도록 코드를 작성했다. Foo가 실행된다면 위 프로그램이 다시 실행될 것이다.

```

kyumin@DESKTOP-NUDFAPK ~
$ ls -l | grep f1
-rw-r--r--+ 1 kyumin 없음      0 Jun 30 23:40 f1

kyumin@DESKTOP-NUDFAPK ~
$ ls -l | grep f1
-rw-r--r--+ 1 kyumin 없음     12 Jun 30 23:44 f1

kyumin@DESKTOP-NUDFAPK ~
$ ps -ef
  UID      PID    PPID  TTY          STIME   COMMAND
  kyumin    1053    1052  ptty1      17:42:17 /usr/bin/bash
  kyumin     989        1  ?          17:15:53 /usr/bin/mintty
  kyumin    1627        1  ptty0      23:44:12 /cygdrive/c/Users/kyumi
ing/SPB_Data/daemon
  kyumin     990    989  ptty0      17:15:53 /usr/bin/bash
  kyumin    1630    1053  ptty1      23:44:22 /usr/bin/ps
  kyumin    1052        1  ?          17:42:17 /usr/bin/mintty

kyumin@DESKTOP-NUDFAPK ~
$ kill 1627

kyumin@DESKTOP-NUDFAPK ~
$ ps -ef
  UID      PID    PPID  TTY          STIME   COMMAND
  kyumin    1053    1052  ptty1      17:42:17 /usr/bin/bash
  kyumin     989        1  ?          17:15:53 /usr/bin/mintty
  kyumin     990    989  ptty0      17:15:53 /usr/bin/bash
  kyumin    1631        1  ptty0      23:44:27 /cygdrive/c/Users/kyumi
ing/SPB_Data/daemon
  kyumin    1632    1053  ptty1      23:44:34 /usr/bin/ps
  kyumin    1052        1  ?          17:42:17 /usr/bin/mintty

```

프로그램 실행 시 PID는 1627로 확인되고 ls -l을 이용해 f1파일의 크기를 보면 점차 늘어나는 것을 볼 수 있다.

이 프로세스를 kill할 경우 15번 시그널을 보내게된다. 위 프로그램은 foo 함수를 실행할 것이고, 이는 동일한 프로그램을 새로 시작하게 된다.

```

kyumin@DESKTOP-NUDFAPK ~
$ daemon
start ~
Killed

kyumin@DESKTOP-NUDFAPK ~
$ start ~
|

```

구분을 위해서 프로그램 실행 시 start 메시지가 출력되도록 만들었다. Kill할 경우 프로그램이 다시 실행되므로 start 메시지가 출력되는 것을 볼 수 있다.

```

kyumin@DESKTOP-NUDFAPK ~
$ kill 1631

kyumin@DESKTOP-NUDFAPK ~
$ ps -ef
  UID      PID    PPID  TTY          STIME COMMAND
  kyumin   1053    1052  pts/1    17:42:17 /usr/bin/bash
  kyumin    989         1  ?        17:15:53 /usr/bin/mintty
  kyumin   1633    1053  pts/1    23:44:41 /usr/bin/ps
  kyumin    990     989  pts/0    17:15:53 /usr/bin/bash
  kyumin   1631         1  pts/0    23:44:27 /cygdrive/c/Users/kyu
ing/SPB_Data/daemon
  kyumin   1052         1  ?        17:42:17 /usr/bin/mintty

kyumin@DESKTOP-NUDFAPK ~
$ ls -l | grep f1
-rw-r--r--+ 1 kyumin 없음      54 Jun 30 23:44 f1

```

```

kyumin@DESKTOP-NUDFAPK ~
$ daemon
start ~
Killed

kyumin@DESKTOP-NUDFAPK ~
$ start ~
|

```

그러나 위 프로그램을 실제로 실행해보면 문제가 있다. Kill을 한 번 할 경우 이전 페이지처럼 정상적으로 프로그램이 재실행되는 것을 볼 수 있다. 그러나 kill을 두 번째로 할 경우 위 사진처럼 프로그램이 재실행되지도 않고 멈추지도 않고 전혀 반응이 없다. Ls로 f1파일의 크기를 보면 점차 늘어나는데 이는 프로그램이 계속 실행 중인 것을 의미한다.

시그널 핸들러 함수에 문제가 있을까하여 여러 차례 수정해봤지만 해결이 되지 않는다. 코드의 문제인지 cygwin을 사용하기 때문에 발생한 문제인지는 파악이 되지 않는다.