

*Xilinx Zynq FPGA, TI DSP, MCU*  
*기반의 프로그래밍 및 회로 설계*  
*전문가 과정*

<리눅스 시스템 프로그래밍>  
2018.03.27 - 24 일차

강사 - 이상훈  
gcccompil3r@gmail.com

학생 - 안상재  
sangjae2015@naver.com

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

void my_sig(int signo)
{
    printf("my_sig called\n");
}

void my_sig2(int signo)
{
    printf("my_sig2 called\n");
}

int main(void)
{
    void (*old_p)(int);
    void (*old_p2)(int);
    old_p = signal(SIGINT, my_sig);    // 과거에 등록한것이 없어서 0 이 반환됨.
    pause();

    old_p2 = signal(SIGINT, my_sig2); // signal()은 my_sig()함수를 반환함.
    pause();
    old_p2 = signal(SIGINT, old_p2);   // old_p2 는 my_sig2 이다.
    pause();

    for(;;)
        pause();
    return 0;
}
```

- SIGINT는 'CTRL + C' 단축키를 의미하는 신호이다. 아래의 결과 사진에서 첫문장은 my\_sig()에서 출력이 되었고, 두번째 문장은 my\_sig2에 의해 출력이 되고, 세번째 문장부터는 signal(SIGINT, my\_sig2)의 반환값이 my\_sig()이므로 old\_p2에 my\_sig2()가 들어가고 signal(SIGINT, old\_p2)에 의해 my\_sig2()함수로 들어가고 결국 my\_sig2()함수에 의해 출력이 된다.

[illegible]

2) signal() 시스템 콜은 첫번째 인자의 signal 를 해당 프로세스가 받게 되면, 두번째 인자의 함수를 호출한다. SIGINT signal 은 “CITL + C” 이고 SIG\_IGN 은 명시된 시그널의 무시 또는 버림이다.

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

int main(void)
{
    signal(SIGINT, SIG_IGN); // SIGINT(“CITL + C”) 을 막음.
    pause();
    return 0;
}

/*
SIGINT 는 'CITL + C'를 막는 신호이다.
kill -2 PID 해도 'CITL + C' 단축키가 안살아나지만,
kill -9 PID 하면 프로세스 죽음.
*/
```

### 2-1) 결과 분석

- 위의 소스코드를 실행시키면 pause()에 의해 계속 대기상태가 된다. 그림 2 와 같이 터미널 창을 하나 더 열어서 “ps -ef|grep a.out” 명령어를 실행시키면 PID 2404(소스코드의 프로세스), 2415(명령어에 의한 프로세스) 인 프로세스 2 개가 존재한다는 것을 알 수 있다. 소스코드에서 signal(SIGINT, SIG\_IGN)에 의해 “CITL + C”를 막아놓았기 때문에 그림 2 와 같이 “kill -2 2404” 명령어를 입력해도 2404 프로세스에 “CITL + C” 시그널이 전달되지 않는다(2 번 시그널은 “CITL + C” 명령어). “kill -9 2404” 명령어를 입력하면 그림 1 에서 “죽었음” 메시지와 함께 2404 프로세스가 사라지는 것을 확인할 수 있다.

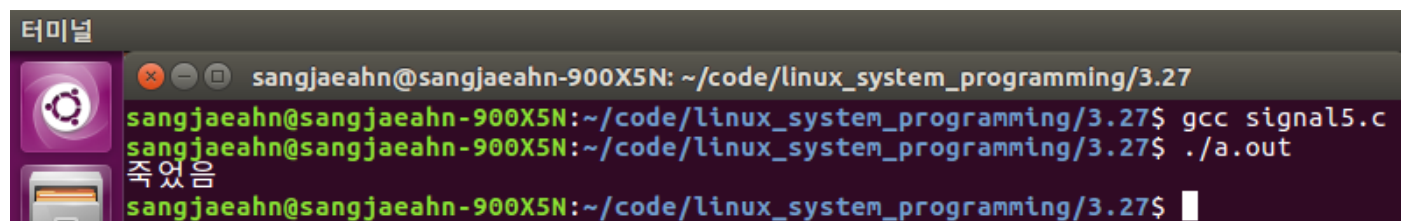


그림 1

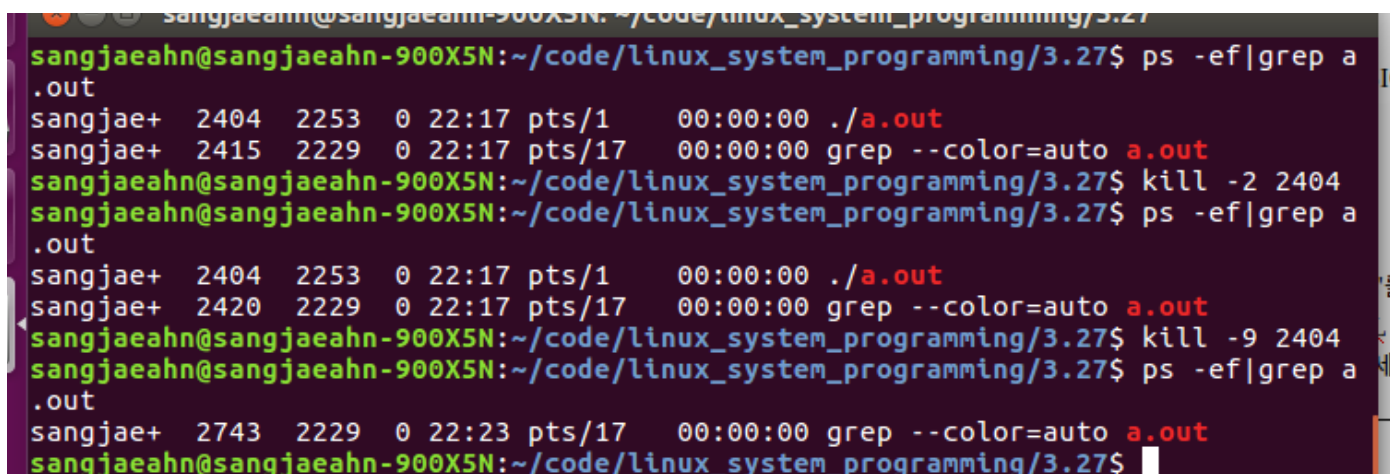


그림 2

3) goto 문은 다른 스택에 대한 권한이 없음. (스택 해제 권한이 없음) 다른 함수로 이동불가능함.

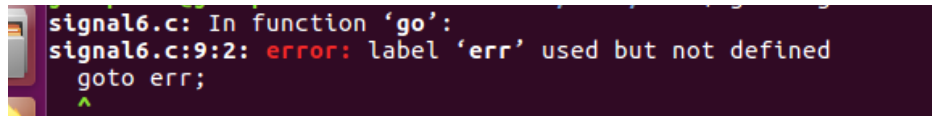
```
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

void go(char buf[])
{
    goto err;
}

int main(void)
{
    int ret;
    char buf[1024] = "test";
    if((ret = read(0,buf,sizeof(buf))) > 0)
        go(buf);
    return 0;
    err :
        perror("read() "); // " " 안의 시스템콜의 동작이 어떻게 되었는지 나타냄.
        exit(-1);
}
```

### 3-1) 결과 분석

- goto 문을 통해 다른 함수로 이동하는 코드를 작성하면 아래 그림과 같이 컴파일 에러가 생긴다.



```
signal6.c: In function 'go':
signal6.c:9:2: error: label 'err' used but not defined
goto err;
```

4) setjmp()와 longjmp()를 같이 사용하면 goto 문과 같은 기능을 구현할 수 있으며, 추가로 함수간에 이동도 가능하다.

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

jmp_buf env;

void test(void)
{
    longjmp(env, 1);    /* env 와 같은 인자를 가지는 setjmp 로 이동하고, setjmp 의
                        리턴값이 1 이 될것이다.*/
}

int main(void)
{
    int ret;
    if((ret = setjmp(env)) == 0) /* setjmp 의 반환값은 longjmp 의 두번째 인자에
                                의해 정해지지만 첫번째 setjmp 의 반환값은 0 이다.*/
    {
        // ...
    }
}
```

```

    test();
    else if(ret > 0)
        printf("error\n");

    return 0;
}

```

5) longjmp 를 통해 다른 곳으로 이동을 하면 다시 원래 위치로 돌아오지 않음.

```

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

jmp_buf env;

void test(void)
{
    int flag = -1;
    if(flag < 0)
        longjmp(env,1);
    printf("call test\n"); /* longjmp 를 하면 다시 돌아오지 않음 longjmp 밑의
                           코드는 의미 없음.*/
}

int main(void)
{
    int ret;

    if((ret = setjmp(env)) == 0)
        test();
    else if(ret > 0)
        printf("error\n");
    return 0;
}

```

6) 여러개의 longjmp 를 사용한 프로그램.

```

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

jmp_buf env1;
jmp_buf env2; // jmp 할 인자를 2 개 선언함.

void test1(void)
{
    longjmp(env1, 1); // env1 인자를 갖는 setjmp 로 이동하고 setjmp 의 반환값은 1 이 됨.
}

void test2(void)
{
    longjmp(env1, 2); // env1 인자를 갖는 setjmp 로 이동하고 setjmp 의 반환값은 2 이 됨.
}

```

```

}

void test3(void)
{
    longjmp(env2, 1); // env2 인자를 갖는 setjmp 로 이동하고 setjmp 의 반환값은 1 이 됨.
}

int main(void)
{
    int ret;
    if((ret = setjmp(env1)) == 0) // 첫번째 이동
    {
        printf("this\n");
        test1();
    }
    else if(ret == 1) // 두번째 이동
    {
        printf("1\n");
        test2();
    }
    else if(ret == 2) // 세번째 이동
        printf("2\n");
    else // 여섯번째 이동
    {
        printf("goto letsgo label\n");
        goto letsgo;
    }

    if((ret = setjmp(env2)) == 0) // 네번째 이동
    {
        printf("second label\n");
        test3();
    }
    else // 다섯번째 이동
        longjmp(env1, 3);
    letsgo: // 일곱번째 이동
        goto err;

    return 0;

err: // 여덟번째 이동
    printf("Error!!!\n");
    exit(-1);
}

```

## 6-1) 결과 분석

```

sangjaeahn@sangjaeahn-900X5N: ~/code/linux_system_programming/3.27
sangjaeahn@sangjaeahn-900X5N:~/code/linux_system_programming/3.27$ ./a.out
this
1
2
second label
goto letsgo label
Error!!!

```

7) alarm() 시스템 콜은 () 안의 시간(초) 후에 프로세스에 SIGALRM 을 전달한다. 만약 () 안이 0 이라면 SIGALRM 은 전달되지 않는다. 만약 alarm 이 여러개 쓰인다면 기존에 설정되었던 alarm 설정값은 취소되고 가장 최근의 alarm 설정값으로 지정된다. 그러므로 alarm 을 사용 할때는 alarm 이 겹치지 않도록 주의해야 한다.

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

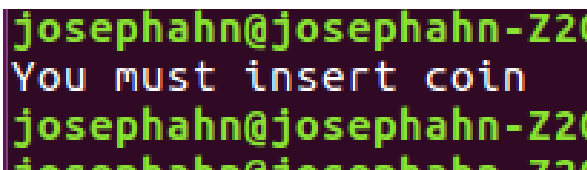
void my_sig(int signo)
{
    printf("You must insert coin\n");
    exit(0);
}

int main(void)
{
    char buf[1024];
    int ret;
    signal(SIGALRM, my_sig); // alarm 이 끝나면 my_sig()함수로 이동한다.
    alarm(3);               // 3 초 알람을 만듦.
    read(0,buf,sizeof(buf));
    alarm(0);               // alarm 을 종료한다.
    return 0;
}
```

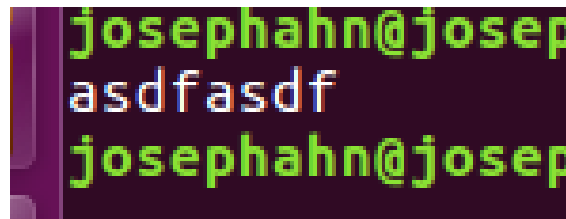
#### 7-1) 결과 분석

- alarm(3) 밑줄에 표준입력(키보드)로부터 입력을 받는 코드가 있고, 그 밑에 줄에 alarm(0) (alarm 을 종료) 이 있다. 그러므로 alarm(3) 이 실행되고 3 초동안 아무 입력도 받지 않으면 my\_sig()함수로 이동하고, 3 초이 내에 어떤 입력을 받는다면 my\_sig()함수로 이동하지 않고 그대로 종료된다.

좌측은 3 초이내에 아무 입력도 받지 않았을 때의 결과이고, 우측은 3 초이내에 어떤 입력을 받은 결과이다.



```
josephahn@josephahn-Z20
You must insert coin
josephahn@josephahn-Z20
```



```
josephahn@josephahn-Z20
asdfasdf
josephahn@josephahn-Z20
```

8) 1~100 까지의 임의의 숫자를 셋팅해서, 사용자로 하여금 입력을 하게 해서 숫자를 맞추는 게임. 단, 3 초 이 내에 입력을 해야함.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <signal.h>
#include <setjmp.h>
#include <unistd.h>
```

```
void my_sig(int signo)
{
    printf("3 초 이내에 입력해야 합니다.\n");
    exit(0);          // 바로 프로세스를 종료함.
}

int main(void)
{
    int input = 1, num;
    srand(time(NULL));
    num = rand()%100 + 1;    // num 은 1~100 까지 임의로 셋팅함.
    signal(SIGALRM, my_sig); // SIGALRM 이 발생하면 my_sig()함수로 이동함.

    while(input)
    {
        alarm(3);          // 3 초 알람을 만들어서, 반드시 3 초 이내에 입력을 받게함.
        scanf("%d",&input);
        alarm(0);          // alarm 을 꺼버린다.
        if(input<num)
        {
            printf("up\n");
        }
        else if(input>num)
        {
            printf("down\n");
        }
        else
        {
            printf("정답입니다.\n");
            break;
        }
    }
    return 0;
}
```



### 8-1) 결과 분석

- 실행을 하고 3 초동안 아무 입력도 하지 않으면 “3 초 이내에 입력해야 합니다.”라는 메시지가 나타난다.
- 정답을 맞출 때까지 임의의 숫자를 계속 입력한다. 입력한 숫자가 정답보다 작으면 “up” 메시지가 나타나고, 입력한 숫자가 정답보다 크면 “down” 메시지가 나타난다. 정답을 맞추면 “정답입니다.” 메시지가 나타난다.

```
sangjaeahn@sangjaeahn-900X5N: ~/code/linux_system_programming/3.27
sangjaeahn@sangjaeahn-900X5N:~/code/linux_system_programming/3.27$ ./a.out
3초 이내에 입력해야 합니다.
sangjaeahn@sangjaeahn-900X5N:~/code/linux_system_programming/3.27$ ./a.out
10
up
20
up
30
up
40
up
50
up
60
up
70
down
66
up
67
정답입니다.
sangjaeahn@sangjaeahn-900X5N:~/code/linux_system_programming/3.27$
```