TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – GJ (박현우)
uc820@naver.com

1. 시스템 프로그래밍 – 6

```c
#include<unistd.h>
#include<signal.h>
#include<stdio.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); //  signal 이전에 입력 했던 signal값 리턴
    pause();
    old_p2 = signal(SIGINT, my_sig2);
    pause();
    old_p2 = signal(SIGINT, old_p2);
    pause();
    for(;;)
        pause();
    return 0;
}
```

- signal()

이전에 입력 했던 signal값을 리턴한다.

이전에 사용했던 signal이 없기 때문에

old_p = 0이고

old_p2에는 이전 signal 리턴 값이 입력된다.

그리고 무한루프를 돈다.

1. 시스템 프로그래밍 – 6

```c
#include<stdio.h>
#include<signal.h>
#include<fcntl.h>
#include<stdlib.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(3);   // alarm 3 은 3초후 signal 날림!
    read(0,buf, sizeof(buf));
    alarm(0);   // alarm 0 은 알람 초기화!!

    return 0;
}
```

- alarm()

이 함수를 사용하면 3초 후 signal을 보낼 수 있다.

alarm(0)은 알람 기능 초기화!

1. 시스템 프로그래밍 – 6

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

void my_sig(int signo){

    printf("Game end\n");
    exit(0);
}

void game(void){

    int num , ret;
    int i = 1;
    char buf[1024];
    char str[64] = "숫자를 입력해주세요 : \n";
    num = rand()%100 +1;
re:
    write(1, str, sizeof(str));
    signal(SIGALRM, my_sig);
    alarm(1);
    read(0, buf,sizeof(buf));

    ret = atoi(buf);

    if(ret == num){
        printf("정답입니다 %d번째\n", i);
    }else if( ret > num){
        printf("아래 입니다. %d번째\n",i);
        i++;
        goto re;
    }else if( ret < num){
        printf("위 입니다. %d번째\n", i);
        i++;
        goto re;
    }
}

int main(void){

    game();

    return 0;
}
```

- Up & down 숫자 게임 만들기

signal과 alarm을 활용하여 구현한

숫자 맞추기 게임이다.

1. 시스템 프로그래밍 – 6

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

void print(void){
    printf("aa");
    goto err;   // stack을 풀어 다른 스택으로 갈 수 없다.
}

int main(void){

    int ret;
    char buf[1024] = "hello world";
    if ((ret = read(0, buf, sizeof(buf))) > 0)
            print();

    return 0;

err:
    perror("read() ");
    exit(-1);
}
```

- goto()

이 방식으로는 함수 끼리 jmp가 불가능하다.
왜냐하면, 함수를 만들 때 마다 stack이 만들어지는데, goto로만은 stack을 풀 수가 없다.

1. 시스템 프로그래밍 – 6

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


jmp_buf env1, env2;

void test(int ret, void (*p)(void)){

    if(ret ==0)
        longjmp(env1, 1);   // 2번째 인자는 리턴값
    p();
}

void test2(void){
    longjmp(env2, 1);
}

int main(void){
    int ret1, ret2;

    if( (ret1 = setjmp(env1)) == 0 ){  // setjmp = goto: , env = lable;
        printf("this1\n");              // setjmp초기 리턴값 0
        test(ret1, test2);
    }

    else if(ret1 > 0){
        printf("error1\n");

        if( (ret2 = setjmp(env2)) == 0){
            printf("this2\n");
            test(ret1, test2);
        }
        else if(ret2 > 0)
            printf("error2\n");
    }

    return 0;
}
```

- setjmp, longjmp

Goto의 문제점을 해결하고자 나온 게 위 두 함수이다.

이제는 함수끼리도 점프가 가능하다.

setjmp의 첫 리턴 값은 0이며, test()함수 안에

longjmp에 들어가 첫 번째 evn1을 통해 다시

setjmp로 돌아오고 두 번째 인자가 리턴 값이다.