TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 윤연성
whatmatters@naver.com

Signal

SIGINT //키보드로 부터 오는 인터럽트 시그널로 실행을 중지시킴 (컨트롤 + c)같은기능

ps- ef

ps -ef |grep a.out //현재 ./a.out 된 pid 번호를 볼수있음

kill -2 pid

kill -9 pid //pid 번호를 입력하면 그 번호의 프로세스를 죽인다

kill -l 꼭 알아야 하는 명령어 모음

SIGHUP : 프로세스 재실행

SIGINT : 현재프로세스 강제종료 = CTRL + C

SIGKILL : 프로세스 강제종료

SIGTERM : 프로세스 종료

SIGTSTP : 프로세스 강제종료 = CTRL + z

kill -SIGKILL (= -9) PID

```
ys@ys-Z20NH-AS51B5U:~/my_proj/24day$ ps -ef |grep a.out
ys        3781  3000  0 10:26 pts/2    00:00:00 ./a.out
ys        3785  3000  0 10:26 pts/2    00:00:00 ./a.out
ys        3786  3000  0 10:26 pts/2    00:00:00 ./a.out
ys        3788  3000  0 10:26 pts/2    00:00:00 grep --color=auto a.out
```

a.out 를 여러번 해서 3781, 3685, 3786,3788 이 여러개 생김
3000 의 같은 부모를 가지고있음

```
ys@ys-Z20NH-AS51B5U:~/my_proj/24day$ kill -9 3781
ys@ys-Z20NH-AS51B5U:~/my_proj/24day$ kill -9 3785
[1]   Killed                  ./a.out
[2]-  Killed                  ./a.out
ys@ys-Z20NH-AS51B5U:~/my_proj/24day$ kill -9 3786
```

```
ys@ys-Z20NH-AS51B5U:~/my_proj/24day$ ps -ef |grep a.out
ys        3944  3000  0 10:27 pts/2    00:00:00 grep --color=auto a.out
```

//kill -9 pid(번호 해서 프로세스들이 죽은것을 확인할수있다)

ctrl+z : 백그라운드로 작업전환
ctrl+d : 정상종료
ctrl+c : 강제종료

시그널과 KILL 쓰다보니 저거 사용해야됨

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

 jmp_buf env1;
 jmp_buf env2;

 void test1(void)
 {
        longjmp(env1, 1);               //goto 랑 동일한 역할을 함
 }                                      //setjmp 는 goto 의 레이블
                                        //env1 인 setjmp 에 ret 에 1 을 반환
 void test2(void)
 {
        longjmp(env1, 2);                       //env1 인 setjmp 에 ret 에 2 를 반환
 }

 void test3(void)
 {
        longjmp(env2, 1);               //env2 인 setjmp 에 ret 에 1 을 반환
 }

 int main(void)
 {
        int ret;
        if((ret = setjmp(env1)) == 0)
        {
                printf("this\n");
                test1();                //ret 에 1 값이 들어가고 다시 if 문을 봄
        }
        else if(ret == 1)
        {
                printf("1\n");
                test2();                //ret 에 2 값이 들어가고 다시 if 문을 봄
        }
        else if(ret == 2)
```

```
		printf("2\n");					//2 를 출력하고 그 다음 if 문으로 들어간다
	else
	{
		printf("goto letsgo label\n");
	goto letsgo;			//letsgo 로 간다
	}
	if((ret = setjmp(env2)) == 0)				//처음엔 무조건 0 이 되어 실행됨
	{
		printf("second label\n");
		test3();   //test3 을 실행하면 setjmp(env2)인 2 번째 if 문을 다시 들름
	}
	else
		longjmp(env1, 3);			//setjmp(env1)인 첫번째 if 문을 다시 돌게된다
letsgo:
		goto err;					//err 로 간다

	return 0;

err:
	printf("Error!!!\n");				//문자 출력
	exit(-1);
}
```

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

jmp_buf env;
{
	int flag = -1;
	if(flah < 0)
		longjmp(env, 1);		//롱점프를 하면 밑에 내용은 중요하지않음
	printf("call test\n");
}

int main(void)
{
	int ret;
	if((ret = setjmp(env)) ==0)
		test();
```

```
        else if(ret >0)
                printf("error\n");
        return 0;
}
```

```
#include <signal.h>
#include <unistd.h>
#include <stdio.h>                      //버퍼에 문자 아무거나 입력 0 보다 크다로
#include <stdlib.h>
#include <fcntl.h>
#include <setjmp.h>

jmp_buf env;

void func(void)
{
        longjmp(env, 1);
}

int main(void)
{
        int ret;
        func();

//      char buf[1024] = "full";
//      if((ret = read(0, buf ,sizeof(buf))) > 0)               //read 0 번 = scanf ,  입력된게 버프로감 ,
//              goto err;
        return 0;

        err:
        printf("doesn't work\n");
//      perror("read() ");                      // 시스템 콜에 대한 동작을 알려줌
        exit(-1);
}
```

setjmp(jmp_buf jmpbuf)

- 함수가 실행된 코드 지점을 점프할 위치로 설정.

- 파라미터가 같은 longjmp()만 이곳으로 점프

longjmp(jmp_buf jmpbuf, int num)

- 파라미터가 jmpbuf 인 setjmp()로 점프한다.
- num 은점프 이후에 setjmp()의 리턴값이 된다. (0 으로 설정할 경우 1 을 리턴)

atoi = string 형의 숫자를 integer 형으로 변환해준다