

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

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com



I/O 제어,
프로세스 제어,
멀티 태스킹과 컨텍스트 스위칭,
signal 활용법,
IPC 기법

파일

signal

03/27- goto.c

```
#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: SIGINT 가 동작하면 my_sig를 동작해라. 과거것이 들어간다. 즉 =null이들어감
    pause();
    old_p2 = signal(SIGINT,my_sig2); // old_p2 = my_sig called 이 들어감.
    pause();
    old_p2=old_p2;
    old_p = signal(SIGINT,my_sig);
    pause();
    // old_p2 = signal(SIGINT,old_p); // old_p2 를 넣으면 my_sig 를 계속 출력한다.
    pause();
    for(;;)
        pause();
    return 0;
}
```

<pre>//프로세스는 시그널을 만나면 죽는다.</pre>

goto

03/27- goto.c

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

int main(void)
{
    int ret;
    char buf[1024]="hi";
    if((ret=read(0,buf,sizeof(buf)))>0)
        goto err;

    return 0;

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

```
void xhs(char *buf)
{
    int ret;
    if((ret=read(0,buf,sizeof(buf)))>0)
        goto err;
}

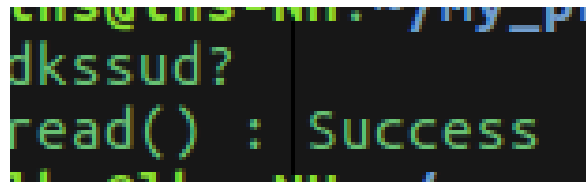
int main(void)
{
    char buf[1024]="hi";
    xhs(buf);
    return 0;

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

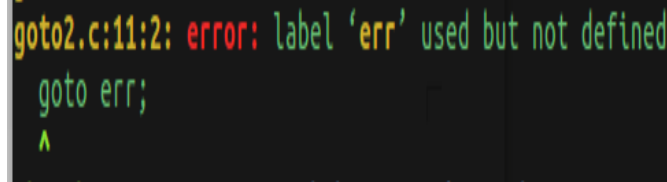
```
Void test(void)
{
    goto err;
}

int main(void)
{
}
```

오른쪽것은 왜 안될까
stack을 풀어 해칠 수 있는 능력이 없다.
이것이 되려면 jmp를 해줘야함.



Terminal output showing a successful read operation. The prompt is 'ch3@ch3:~\$'. The command 'cat /dev/urandom | tr -dc 'a-z' | fold -w 20 | xargs -n 1 sh' is executed. The output is 'read() : Success'.



Terminal output showing a compilation error. The command 'gcc goto2.c' is executed. The error message is 'goto2.c:11:2: error: label 'err' used but not defined'. The code snippet shown is 'goto err;'.

Setjmp

03/27- setjmp.c

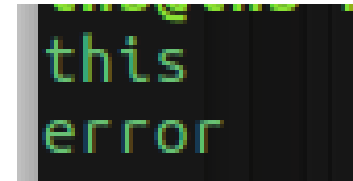
```
#include <fcntl.h>
#include <stdlib.h>
#include <setjmp.h>
#include <stdio.h>

jmp_buf env; // jmp_buf goto 사용했을 때 반드시 err(레이블) 이있어야한다. 이것을 setjmp함수가 대신해주는 것이다.
              // 즉 실제로 env가 레이블을 가지고 있다는것.

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

int main(void)
{
    int ret;
    if((ret = setjmp(env)) == 0) // setjmp(env)의 위치가 goto 레이블의 위치이다.
    {                             // setjmp를 하면 처음에 무조건 0나온다.
        printf("this\n");
        test();                  // longjmp하면 다시 돌아온다.
    }
    else if(ret>0)
        printf("error\n");
    return 0;
}
```

동작:



내코드 2

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

jmp_buf env; // jmp_buf goto 사용했을 때 반드시 err(레이블) 이있어야한다. 이것을 setjmp함수가 대신해주는 것이다.
// 즉 실제로 env가 레이블을 가지고 있다는것.

```
jmp_buf env2; // 두번째 jmp_buf
```

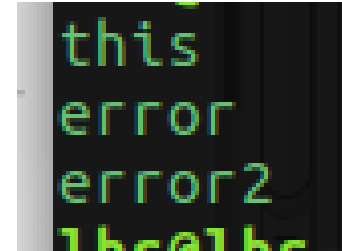
```
void test(void)
{
    longjmp(env,1);
}
```

```
void test2(void)
{
    longjmp(env2,1); // env2로 점프해라. 그리고 setjmp(ev2)의 값은 1이다.
}
```

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

    if((ret=setjmp(env2))==0)
    {
        if((ret = setjmp(env)) == 0) // setjmp(env)의 위치가 goto 레이블의 위치이다.
        { // setjmp를 하면 처음에 무조건 0의 값이 들어가있음.
            printf("this\n");
            test();
        }
        else if(ret>0)
        {
            printf("error\n");
            test2();
        }
    }
}
```

동작:



```
    }  
}  
else if(ret>0)  
{  
    printf("error2\n");  
}  
return 0;  
}
```

Techer → 해석해보도록

```
#include <fcntl.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <setjmp.h>  
  
jmp_buf env1;  
jmp_buf env2;  
  
void test1(void)  
{  
    longjmp(env1, 1);  
}  
  
void test2(void)  
{  
    longjmp(env1, 2);  
}  
  
void test3(void)  
{  
    longjmp(env2, 1);  
}
```



```

int main(void)
{
    int ret;
    if((ret = setjmp(env1)) == 0) // env1 레이블
    {
        printf("this\n");    // this가 제일 먼저 출력
        test1();             // env1=1로
    }
    else if(ret == 1)
    {
        printf("1\n");        //두번째로 출력
        test2();              // env1=2로   ret=setjmp(env) == 2
    }
    else if(ret == 2)
        printf("2\n");        // 세번째 출력
    else
    {
        printf("goto letsgo label\n"); // 5번째로 출력
        goto letsgo;              // goto레이블 letsgo → goto err;
    }

    if((ret = setjmp(env2)) == 0)
    {
        printf("second label\n"); // 4 번째 출력
        test3();                  // env2=1
    }
    else
        longjmp(env1, 3); // env1=3으로 즉 else 문으로 가라
letsgo:
    goto err;

    return 0;

err:

```

```
printf("Error!!!\n"); //3번째출력
exit(-1);           // → exit(-1)은? Exit(0) 은 정상종료 나머지는 비정상종료라는 뜻
}
```

→ goto 와 setjmp의 차이점은 뭘까?

우선 goto는 쓰기 쉽다.setjmp 는 함수 사이를 뛰어넘을 수 있다.

사용방법은

Goto = longjmp 이고 , goto 레이블 = setjmp 와 마찬가지로이다.

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

```
jmp_buf env;
```

```
void test(void)
{
    int flag = -1;
    if(flag < 0)
        longjmp(env,1);
    printf("call test\n");
}
```

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

    if((ret=setjmp(env))==0)
        test();
    else if(ret>0)
```

```
    printf("error\n");  
    return 0;  
}  
→ longjmp 밑에가 출력되지 않는 것을 확인할 수 있다.
```

```
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.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);
    read(0,buf,sizeof(buf));
    alarm(0);
    return 0;
}
```

Alarm 함수에 시간이 지나면 SIGALRM 이라는 시그널이 발생한다.

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

jmp_buf env;

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

void my_sig(int signo)
{
    printf("빨리해이썻 \n");
    exit(0);
}

int main(void)
{
    char buf[1024];
    int ans;
    int ret;
    int arr;
```

```
srand(time(NULL));
arr = rand()%100+1;

signal(SIGALRM,my_sig);
printf("arr=%d\n",arr);
alarm(3);
read(0,buf,sizeof(buf));
ans = atoi(buf);
// scanf("%d",&ans);
alarm(0);

while((ret=setjmp(env))==0)
{
    if(arr == ans)
    {
        right();
    }
    else if( arr > ans)
    {
        printf("high\n");
    }
    else
    {
        printf("row\n");
    }
    alarm(3);
    read(0,buf,sizeof(buf));
    ans=atoi(buf);
    // scanf("%d",&ans);
    alarm(0);
}
printf("정답");
return 0;
}
```

어제 빠진 부분 정리. → 현우씨꺼 보면서 공부

fork()- 자식 프로세서 생성

abort() - **signal** 로 자식 프로세스를 죽인다.

Signal 맞아 죽었기 때문에 term_status에서 status와 0x7f 비트로 core dump확인하고 6을 출력

core는 프로그램의 비정상적인 종료가 발생하는 경우 커널에서 해당 프로세스와 관련된 메모리를 덤프시킨 파일을 말함.

<core dump bit>

어느 메모리상태에서 어떻게 죽었는지를 확인 할지 말지 지정해주는 비트가

코어 덤프이다.

1 코어덤프를 끈다

0 코어덤프를 안끈다.

Extract Status

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

int main(void)
{
    pid_t pid;
    int status;
    if((pid=fork())>0)
    {
        wait(&status);
        printf("status : %d\n",status);
    }
    else if(pid == 0)
        exit(7);
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
```

비정상 종료란?

통칭 하자면 시그널이다.

process는 시그널을 맞으면 죽게된다.

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

```
int main(void)
{
    pid_t pid;
    int status;
    if((pid=fork())>0)
    {
        wait(&status);
        printf("status : 0x%x\n",(status>>8)&0xff);
    }
    else if(pid == 0)
        exit(7);
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
```

```
printf("status : 0x%x\n",WEXITSTATUS(status)); == printf("status : 0x%x\n",(status>>8)&0xff);
```

같은 뜻. 취향

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

```
int main(void)
{
    pid_t pid;
    int status;
    if((pid=fork())>0)
    {
        wait(&status);
        printf("status : 0x%x\n",status);
    }
    else if(pid == 0)
        abort();
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
```

Abort : 8비트

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

void term_status(int status)
{
    if(WIFEXITED(status))
        printf("(exit)status : 0x%x\n",WEXITSTATUS(status));
    else if(WTERMSIG(status))
        printf("(signal)status: 0x%x,%s\n",status % 0x7f,
WCOREDUMP(status) ? "core dumped" : ""); //0x7f core dump
}

int main(void)
{
    pid_t pid;
    int status;
    if((pid = fork())>0)
    {
        wait(&status);
        term_status(status);
    }
```

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

void term_status(int status)
{
    if(WIFEXITED(status))
        printf("(exit)status : 0x%x\n",WEXITSTATUS(status));
    else if(WTERMSIG(status))
        printf("(signal)status: 0x%x,%s\n",status % 0x7f,
WCOREDUMP(status) ? "core dumped" : "");
}

void my_sig(int signo)
{
    int status; //sigchld 넘어오고
    wait(&status);
    term_status(status); // child 상태확인하고
}
```

<pre> } else if(pid == 0) abort(); else { perror("fork() "); exit(-1); } return 0; } </pre>	<pre> int main(void) { pid_t pid; int i; signal(SIGCHLD, my_sig); // sigchld 날라오면 my_sig 동작시켜서(signal- manual 비동기 처리 - 상황에대한 대처를 하고 싶으면 모두 signal을 한다. if((pid = fork())>0) for(i=0;i<10000;i++) { usleep(50000); printf("%d\n",i+1); } else if(pid == 0) sleep(5); else { perror("fork() "); exit(-1); } return 0; } </pre>
<p>sigant 함수는 비동기 처리 방식으로 특정한 상황에 대처하고 싶을 때 사용된다. 그렇기 때문에 비상 대피가 필요할 때의 manual과 같은 역할을 수행한다.</p>	<p>시그널은 매뉴얼 → 어떤 상황에서 어떤 동작을 시킬 것인가를 지침. 불이 났을 때, 동작시켜야함. 자식이 죽으면 my_sig을 호출한다. 정상종료이기때문에 exit에 걸린다. 리턴 0 했으므로 0이냐옴. 해당 포인터는 정상종료됨을 의미함. Signal char가 동작하자마자 my_sig이 동작함 0 → qi</p>

Execl()

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

int main(void)
{
    int status;
    pid_t pid;
    if((pid=fork())>0)
    {
        wait(&status);
        printf("prompt >\n");
    }
    else if(pid == 0)
        execl("./newpgm", "newpgm", "one", "two", (char*)0);
    return 0;
}
```

Excl()
newpgm 실행 시키고 argv로 newpgm, one 과 two를 인자로 취한다.

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

int main(void)
{
    int status;
    pid_t pid;
    char *argv[] = { "./newpgm", "newpgm", "one", "two", 0 };
    char *env[] = { "name = OS_Hacker", "age = 20", 0 };
    if((pid=fork())>0)
```

Execve()
argv 와 envp 두 인자를 모두 사용함.

<pre>{ wait(&status); printf("prompt > \n"); } else if(pid == 0) execve("./newpgm",argv,env); return 0; }</pre>	
--	--

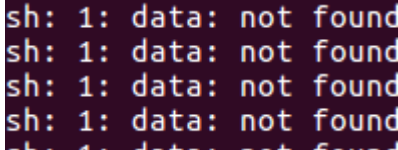
system()

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

int my_system(char *cmd)
{
    pid_t pid;
    int status;
    char *argv[] = {"sh", "-c", cmd, 0};
    char *envp[] = {0};

    if((pid=fork())>0)
    {
        wait(&status);
    }
    else if(pid==0)
        execve("/bin/sh",argv,envp);
}

int main(void)
{
    for(;;)
    {
        sleep(1);
        my_system("data");
    }
}
```



```
sh: 1: data: not found
sh: 1: data: not found
sh: 1: data: not found
sh: 1: data: not found
```

Sh: 1: data: not found 반복 될까?

// sh, command, cmd

// shell 만들고 argv 를 실행

<pre>printf("after\n"); return 0; }</pre>	
---	--

daemon()

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

int daemon2(void)
{
    int i;
    if(fork()>0)
        exit(0);
    setsid();          //프로세스 상태 ? 로 만듦
    chdir("/");         // 실행 시켜야 할 파일이 있으니 root로보냄
    umask(0);          // 권한 설정
    for(i=0;i<64;i++)
        close(i);      // 리눅스 기본 64개 파일 열으니 전부 닫음
    signal(SIGCHLD,SIG_IGN); //자식이 생기면 무시 -> signal 매
    return 0;
}

int main(void)
{
    daemon2();
    for(;;);
    return 0;
}
```

Demon 함수는 서비스 프로그램을 만들려면 필수적으로 만들어야하는 함수이다.
데몬 함수를 끄려면 kill -9를 사용하면된다.

→ kill -9 하는법 까먹었다..

