TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
24 일차 (2018. 03. 27)

-signal

보통 #include <signal.h>의 속에 정의 된 시그널을 사용한다.

시그널 종류	설명
SIGHUP	Hangup을 위한 시그널로 터미널과 시스템 사이에 통신 접속이 끊어졌을 때 터미널에 연결된 프로세스들에게 커널이 보내는 시그널이다.
SIGINT	Interrupt를 위한 시그널로 유저가 인터럽트를 발생시키는 키를 입력했을 때 그와 연결된 프로세스에게 커널이 보내는 시그널이다. 이 시그널은 프로세스를 종료할 때 많이 사용되는 시그널이다.
SIGQUIT	Quit를 위한 시그널로 유저가 터미널에서 Quit 키를 치면 커널이 프로세스에게 SIGQUIT 시그널을 보낸다.
SIGILL	illegal 명령, 즉 비정상적인 명령을 수행할 때 OS가 발생시키는 시그널이다.
SIGTRAP	Trace Trap을 위한 시그널로 디버거들이 주로 사용하는 시그널이다.
SIGABRT	Abort 실행시 발생하는 시그널로 Abort는 시스템이 비정상적으로 종료될 때 해당 정보를 남기는 명령이다.
SIGIOT	SIGABRT와 유사한 작업을 수행시 발생하는 시그널이다.
SIGEMT	Emt 명령 실행시 사용되는 시그널이다.
SIGFPE	Floating 포인터 예외사항, 즉 부동소수점 사용에서 오버플로우나 언더플로우가 발생했을 때 사용되는 시그널이다.
SIGKILL	프로세스가 다른 프로세스를 Kill 시키기 위해 발생하는 시그널이다.
SIGBUS	Bus에러가 발생했을 때 사용되는 시그널이다.
SIGSEGV	메모리 세그먼트 등이 깨졌을 때 발생하는 시그널이다.
SIGSYS	시스템 호출을 할 때 잘못된 인수를 사용하면 발생하는 시그널이다.
SIGPIPE	파이프에서 사용하는 시그널로 아무도 읽지 않는 파이프에 데이터를 출력할 때 발생하는 시그널이다.
SIGALRM	알람 클락 시그널로 해당 타이머가 끝나면 발생되는 시그널이다.
SIGTERM	kill에 의해 프로세스가 종료할 때 발생되는 시그널이다.

학습 예제

```
#include <signal.h>
#include <stdio.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);              //함수 포인터
     void(*old_p2)(int);
     old_p = signal(SIGINT, my_sig);      //행동 지침 등록, old_p 는 NULL 값을 가진다.
     pause();                             //시그널을 수신할 때까지 대기
     old_p2 = signal(SIGINT, my_sig2);  //old_p2 에 signal 의 리턴 값인 my_sig 가 들어간다.
     pause();
     old_p2 = signal(SIGINT, old_p);
      pause();
      for(;;)
      pause();
      return 0;                                    }
```

결과 ctrl +c (시그널, SIGINT)세번 누른다.

^Cmy_sig called
^Cmy_sig2 called
^C
old_p 를 old_p2 로 바꾸면 안 끝난다. 무한 반복하게 된다. 왜 일까? 시그널이 뭔가를 리턴한다는 뜻이다. 시그널은 기본적으로 자기 자신 전에 등록했던 것을 즉, **한 단계 전에 등록했던 것을 리턴**해준다. 그래서 처음에 시그널이 작동했을 때, 그 전에 등록한 것이 없으므로 NULL 이 리턴이 되어 old_p 는 NULL 이 저장된 것이다. 그래서 처음 예제의 마지막에 시그널이 들어왔을 때, 작동할 것이 없으므로 SIGINT 3 번에 끝난 것이다.

^Cmy_sig called
^Cmy_sig2 called
^Cmy_sig called
^Cmy_sig called
^Cmy_sig called
old_p2 에는 my_sig 가 들어간다. 이는 시그널이 전에 등록했던 것을 리턴하기 때문이다. 그래서 old_p2 는 my_sig 를 계속 호출하게 되는 것이다.

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
int main (void)
{   signal(SIGINT, SIG_IGN);
   pause();
   return 0;                }
```

결과는 커서가 깜박인다
ps -ef | grep a.out 를 하면 pid 값을 확인할 수 있다. 그 pid 값을 입력하면 이 프로세스를 죽일 수 있다. 그러나 kill -2 3055[=pid] 로 kill -2 를 하면 이 프로세스는 죽지 않는다. 이유는 SIG_IGN 로 SIGINT 를 무시해 놨기 때문이다. 그래서 kill -9 3055[=pid]를 해야한다. 여기서 kill -9 [ppid 값]로 죽이면 실행되고 있던 터미널 창이 꺼진다.

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
int main (void)
{  int ret;
  char buf [1024] = "cmd";
  if((ret = read(0, buf, sizeof(buf)))>0) //키보드 입력(0)을 받아 buf 에 넣고 그 사이즈를 리턴
   goto err;

   return 0;
```

```
  err:
    perror("read()"); //read() 시스템 콜에 대한 동작 설명으로 동작이 어떻게 되었는지 출력해준다.
    exit(-1);                    }
```

결과

read(): Success
read 0 은 키보드로 입력을 받게 된다. enter 도 키보드로 입력 받는 것이기 때문에 이 예제는 무조건 0 보다 크게 되므로 if 문으로 들어가게 된다. 그래서 goto err 로 가게 된다.

위의 예제를 아래와 같이 main 함수가 아닌 다른 곳에서 이동하는 것으로 만들어 보았다.

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

int main (void)
{   int ret;
   char buf [1024];
    go (ret, buf);
   return 0;

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

```
sue100012@sue100012-Z20NH-AS51B5U:~/project/3_27$ gcc gotest.c
gotest.c: In function ‘go’:
gotest.c:10:4: error: label ‘err’ used but not defined
    goto err;
    ^
```

goto 문을 통해 다른 함수로 이동하는 코드를 작성하면 위와 같이 error 가 발생한다. 이는 함수는 스택을 호출하고 작동을 한 후, 스택을 해제해주어야 하는데, goto 는 해제할 능력이 없다. 그래서 main 으로 갈 수가 없다. 이를 해결하기 위해서 나온 것이 아래 예제이다.

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

jmp_buf env;    // 전역 변수

void test(void)
```

{ longjmp(env, 1); //첫번째 인자를 공통으로 가지는 setjmp 에 반환값을 준다. 두번째 인자가 반환값
Printf("call test₩n"); } //longjmp 를 하면 다시 돌아오지 않으므로, 밑의 코드는 의미가 없다.

```
int main (void)
{  int ret;
  if((ret = setjmp(env)) == 0) //goto 레이블. setjmp 는 longjmp 두번째 인자가 반환값이나 처음 호출되면
   test();                    무조건 0 이다.
  else if(ret >0)
   printf("error₩n");
  return 0;                }
```

결과
error
err: 를 setjmp 가 대신하기 때문이다. If 문 아래 test();를
{ printf("test₩n");
test(); } 로 변경하면 test 도 같이 출력된다. 이는 setjmp 가 처음으로 호출되면 0 을 리턴하기 때문이다.

```
#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#include <stdlib.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);    //SIGALRM 은 밑의 alarm 에서 지정한 시간 안에 입력을 못 받으면
                                 my_sig 호출한다.
   alarm(3);                   //3 초안에 키보드 입력을 받는다.
   read(0, buf, sizeof(buf));
   alarm(0);                   //alarm 을 초기화 해주는 것으로 alarm 을 끈다.
   return 0;                   }
```

오류가 나지 않으려면 alarm(0);으로 알람을 꺼주는 작업을 꼭 해주어야 한다. 위의 코드를 이용하면 스무고개, 행맨등을 만들 수 있다. 밑의 예제는 이를 이용하여 숫자 업다운을 만든 예제이다. 랜덤으로 숫자 1~100 생성하고 사용자에게 숫자를 입력 받고 그 숫자보다 작은지 큰지 알려준다. 1 초 내에 답을 입력하지 않으면 지게 되고 기회제한 있다.

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

void sig_handler(int signo)       //1 초 안에 입력하지 않으면 져서 알림을 뜨게 하는 함수이다.
{         printf("You lose! Input should be within 1 second!₩n");
        exit(0);                                                    }
void make_game(int *data)       //숫자 지정
{         *data = rand() % 100 + 1;         }

bool check_correct(int data, int cmp)   // 입력한 수가 지정된 수와 같은지를 알아보는 함수이다.
{         if(data == cmp)
                return true;
        else
                return false;                    }

void start_game(int data)
{         char buf[32] = {0};
          bool fin;
          int i, cmp;

        for(i = 0; i < 10; i++)                         // 기회 10 번 지정
        {       signal(SIGALRM, sig_handler);
                printf("숫자를 맞춰봐!₩n");
                alarm(1);
                read(0, buf, sizeof(buf));
                alarm(0);                       // 알람을 초기화을 해주는 것.
                cmp = atoi(buf);

                fin = check_correct(data, cmp);
                if(fin)                                 //위의 함수로 참인지 알아보고 나서 참을 반환하였을 때
                {       printf("You Win!!!₩n");
                        exit(0);                                }
                else                                    // 거짓을 리턴했을 때, 큰지 작은지를 알려주는 것
                {       if(data > cmp)
                                printf("%d 보다 크다₩n", cmp);
                        else
                                printf("%d 보다 작다₩n", cmp);                          }
```

```
        }
        printf("You Lose!!! You Babo₩n");          //기회 10 번이 끝났으므로 졌음을 출력해주는 것
}

int main(void)
{         int data;

        srand(time(NULL));          // 난수 생성.
        make_game(&data);
        start_game(data);
        return 0;                   }
```