TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
26 일차 (2018. 03. 29)

※ 다른 사람이 만든 코드를 분석할 때 쉽게 볼 수 있는 방법

> 헤더파일, 구조체, 메인함수, 함수명 : 괜찮지만 범주가 넓다.

> **& 를 보면 된다.** &는 주소를 전달한다. 이는 포인터를 전달한다는 것이다. 이 뜻은 &로 보낸 것을 변경할 수 있다는 것이다. 무언가 함수 안에서 값이 바뀌거나 값이 세팅된다는 것을 알 수 있다. 함수는 리턴이 하나인데, 포인터를 쓰면 여러 개를 같이 사용할 수 있다. 그래서 &가 들어 있는 것을 보면 함수를 통해서 값을 받아오거나 변경될 수 있다는 것을 파악할 수 있다.

※ 시그널을 막아야만 하는 경우? – 중요한 작업이 있어서 무조건 먼저 처리해야 할 때.

Sigaction()

시그널이랑 동일하게 사용가능하다. 시그널을 취급할 방법을 선택할 수 있다. 시그널이 언제 발생되고 어떻게 그 핸들러가 호출될 것인지에 대해 제어할 수 있는 부가적인 플래그를 지정하도록 허용한다.

형태 : int sigaction(int signo, const struct sigaction *act, struct sigaction *oact);

인자 : signo : 행동을 지정할 시그널, 잡을 시그널의 번호(SIGKILL, SIGSTOP 은 제외)

act : 지정하고 싶은 행동. 어떻게 처리할 것이라고 설정해 놓은 sigaction 구조체 값을 넘겨서 그에 맞는 실행을 한다. NULL 값을 주게 되면 signo 와 연관된 동작이 변하지 않는다. 이전에 지정해놓은 signo 에 대한 설정이 동작된다.

oact : 나중에 복구를 위해 현재 값을 저장한다. NULL 값을 주면 이전 행동 저장한 것이 생략된다.

리턴값 : 성공시 0, 실패시 -1 을 리턴한다. ?????

struct sigaction 타입의 구조체들은 어떻게 특정한 시그널을 처리할 것인지에 대한 모든 정보를 지정하기 위해서 sigaction 함수에서 사용된다.

Sighandler_t sa_handler : signal 함수의 action 인수와 같은 방법으로 사용되어지며 SIG_IGN(해당 시그널을 무시), SIG_DFL(시그널 자체의 행동 수행) 또는 핸들러의 포인터 / 함수포인터(정의된 행동)가 될 수 있다.

sa_mask : 핸들러가 작동되고 있는 동안 block 혹은 non-block 될 시그널의 집합을 설정한다. 시그널이 전달 되었을 때, 핸들러가 작동되기 전에 디폴트로써 자동적으로 블록이 된다. 즉, 시그널이 처리되는 동안 나머지 시그널들을 블로킹상태에 있도록 해서 순차적으로 시그널을 발생시키는 역할을 한다.

```
#include <stdio.h>
#include <signal.h>        // sigaction
#include <unistd.h>        // sleep 함수 헤더파일

struct sigaction act_new;
struct sigaction act_old;    //전역변수

void sigint_handler(int signo)
{   printf("Ctrl + C₩n");
    printf("If you push it one more time them exit₩n");
    sigaction(SIGINT, &act_old, NULL);        }   //SIGINT 면 act_old 가 동작한다. 처음 들어갔을 때는
```

act_old 값이 없기 때문에 시그널이 발생되지 않고, 두번째에 SIGINT 값이 들어있어 프로세스를 종료함.

```
int main(void)
{   act_new.sa_handler = sigint_handler;      //시그널 두번째 인자에 핸들러 등록
    sigemptyset(&act_new.sa_mask);            //특정 시그널이 막음. 이 코드에서는 비어있어서 아무것도
                                              막지 않겠다는 뜻이 된다.

    sigaction(SIGINT, &act_new, &act_old); //act_new 를 동작시킨다. act_old 는 예전에 등록시켰던
                                              시그널의 정보를 받아온다. 이전 핸들러를 넣어준다.
    while(1)
    {    printf("sigaction test₩n");
         sleep(1);                        }        //ctrl +c 를 누르면 sigint_handler 그리고 두번째는 NULL
    return 0;       }
```

test.c / kill.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
void gogogo(int voidv)
{    printf("SIGINT Accur!₩n");
    exit(0);                          }
int main (void)
{    signal(SIGINT, gogogo);          //SIGINT 오면 gogogo 함수 실행
    for(;;)                           //무한 루프
    {   printf("kill Test₩n");
        sleep(2);                 }
    return 0;                         }

> gcc -o test test.c

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
int main (int argc, char *argv[])
{     if(argc < 2)                    //인자가 2 개보다 작을 때
        printf("Usage : ./exe pid₩n");          //정확한 값을 입력하라는 메시지를 띄운다.
     else
        kill(atoi(argv[1]), SIGINT);     //문자열을 숫자로 변환하여 SIGINT 에 해당하는 시그널을 날린다.
    return 0;       }
  > gcc kill kill.c
  > ./test >ps -ef |grep .test > pid 값 확인 > ./kill pid 값
   여기서 ./test &을 쓰면 pid 값을 출력해준다.
```

```
sue100012@sue100012-Z20NH-AS51B5U:~/project/3_29$ ./test
kill Test
kill Test
SIGINT Accur!
sue100012@sue100012-Z20NH-AS51B5U:~/project/3_29$ ./kill 3868
```

※ CPU 는 순차처리에 특화되어있다. 클락 스피드가 빠르기 때문에 빠른 연산에 유리하다. 그래픽카드와 dsp 는 용도가 비슷하지만 dsp 는 하나를 고속처리하고 그래픽카드는 BW 가 넓고(밴드 수↑대역폭↑) 속도가 느리지만 병렬처리가 우세하다.

Thread

종속적이다. 프로세스는 vm 이 분리되어 독립적이나 쓰레드는 종속적이다. 쓰레드는 메모리를 공유한다. 그래서 공유하게 되면서 가장 크게 문제가 되는 건 크리티컬 섹션이라 락을 걸어주었다. 병렬 처리를 쓰레드를 이용하게 된다.

```
#include <stdio.h>
#include <pthread.h>
void *task1(void *X) // 뭔가 구동시키는 함수. void 포인터 쓰는 이유? 어떤 것이든 인자로 받고 어떤
                        것이던지 리턴하겠다는 뜻이다.
{   printf("Thread A Complete₩n");          }

void *task2(void *X) // 뭔가 구동시키는 함수
{    printf("Thread B Complete₩n");          }

int main (void)
{    pthread_t ThreadA, ThreadB; // pthread _t 가 쓰레드 지정할 때 필요하다. 헤더는 위에 있다.

    pthread_create(&ThreadA, NULL, task1, NULL); //메모리에 올린 것은 아니고 쓰레드를 구동 등록
    pthread_create(&ThreadB, NULL, task2, NULL); //task2 는 ThreadB 가 구동시키겠다. 쓰레드의
                                                    생김새만 만들어 놨다.
    pthread_join(ThreadA, NULL); // 조인을 하는 순간 메모리에 올리는 것. 실제 메모리에 올리는 구간
    pthread_join(ThreadB, NULL); // ThreadB 실행

    return 0;       }
```

처음에 컴파일이 안 된다. 왜일까? 옵션을 주어야 한다
gcc thread.c -lpthread (옵션을 뒤에 붙여야 한다.) 그럼 컴파일이 된다. gcc thread.c -lpthread
결과는

```
Thread B Complete
Thread A Complete
```

네트워크 프로그래밍

1. CS(Client Server)
2. 토클로지(위상 수학 x) 네크워크 구성도(그래프 알고리즘)
3. TCP / IP 프로토콜 (4 계층 구현)
 계층이 많으면 무거워진다. 4 계층이기 때문에 가볍다.

basic_client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef struct sockaddr_in     si;
typedef struct sockaddr *    sap;

void err_handler(char *msg)
{   fputs(msg, stderr);
   fputc('₩n', stderr);
   exit(1);                          }

int main(int argc, char **argv)
{   int sock;
   int str_len;
   si serv_addr;
   char msg[32];

   if(argc !=3)      // 어디로 접속하는지 알기 위해서. 옆사람 IP 주소 된다.
   {     printf("use : %s <IP> <port>₩n", argv[0]);       //사설 IP 192.168.0.
         exit(1);                                          }

    sock = socket(PF_INET, SOCK_STREAM, 0);
 //파일 디스크립터 리턴하는 것. 네트워크 상의 파일 디스크립터를 얻은 것으로 통신할 수 있는 파일 디스크립터를 얻은 것이다.
    if(sock == -1)
    err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr)); // 서버 어드레스 초기화. 이 단락은 패턴으로 기억할 것.
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
```

```
    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1) // 서버 addr 로 연결
        err_handler("connect() error");

    str_len = read(sock, msg, sizeof(msg)-1);

    if(str_len == -1)
       err_handler("read() error!");

    printf("msg form serv :%s ₩n", msg);
    close(sock);

  return 0;                                    }
```

basic_server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef struct sockaddr_in si;
typedef struct sockaddr *   sap;

void err_handler(char *msg)
{    fputs(msg, stderr);
    fputc('₩n', stderr);
    exit(1);                    }

int main (int argc, char **argv)
{    int serv_sock;
    int clnt_sock;

    si serv_addr;
    si clnt_addr;
    socklen_t clnt_addr_size;

    char msg[] = "Hello Network Programming";      //전달하려고 하는 문자열
```

```
  if (argc !=2)//포트 번호 입력하라는 뜻이다. 포토 번호는 통로이다. = 서비스 번호이다 여기선 7777
  {      printf("use : %s <port>₩n", argv[0]);
           exit(1);                                    }
//포트 번호를 알면 포트번호의 특정역할을 알 수 있다. 80 : WWW(웹브라우저) 20, 21 : ftp 22 : ssh
   serv_sock = socket(PF_INET, SOCK_STREAM,0); // 소켓도 파일이어서 파일 디스크립터가 넘어오는 것

  if(serv_sock == -1)       //소켓이 리턴하는 것은 파일 디스크립터 오픈이랑 같다.
    err_handler("socket() error");

  memset(&serv_addr, 0, sizeof(serv_addr)); //서버 어드레스 주소 127.0.0.1 은 로컬주소로 내 주소다
  serv_addr.sin_family = AF_INET;              //tcp 소켓 형식
  serv_addr.sin_addr.s_addr = htonl(INADDR_ANY); //어떤 아이피 주소든 다 받겠다.
  serv_addr.sin_port = htons(atoi(argv[1])); //포트 번호 =  서비스 번호 어떤 서비스를 열 것인가

  if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1) //아이피 주소 세팅 서버에다가 그것이
          err_handler("bind() error");                          bind 다. 서버에 아이피가 세팅 된다.

  if(listen(serv_sock, 5) == -1) // 5 명 받겠다는 뜻으로 그 이상은 안 된다. 실제 클라이언트 접속을
          err_handler("listen() error");                                  기다리는 구간이다.

  clnt_addr_size = sizeof(clnt_addr); //32
  clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size);// 서버 소켓이 무엇을
기다리는가 클라이언트의 접속. 접속 허용 구간

  if(clnt_sock == -1)
    err_handler("accept() error");

  write(clnt_sock, msg, sizeof(msg));
  close(clnt_sock);
  close(serv_sock);

  return 0;                    }
```

결과 7777 를 붙여주어야 한다!

```
sue100012@sue100012-Z20NH-AS51B5U:~/project/3_29$ ./serv
use : ./serv <port>
sue100012@sue100012-Z20NH-AS51B5U:~/project/3_29$ ./clnt
use : ./clnt <IP> <port>
sue100012@sue100012-Z20NH-AS51B5U:~/project/3_29$ ./serv 7777
sue100012@sue100012-Z20NH-AS51B5U:~/project/3_29$ ./clnt 127.0.0.1 7777
msg form serv :Hello Network Programming
```

```
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

int main(void)
{     int fd[3];
      int i;

    fd[0] = socket(PF_INET, SOCK_STREAM, 0);
    fd[1] = socket(PF_INET, SOCK_DGRAM, 0);
    fd[2] = open("test.txt", O_CREAT | O_WRONLY |O_TRUNC);

    for(i=0;i<3; i++)
      printf("fd[%d] = %d₩n", i, fd[i]);

    for(i=0; i<3; i++)
       close(fd[i]);

return 0;                          }
```

결과

```
sue100012@sue100012-Z20NH-AS51B5U:~/project/3_29$ ./a.out
fd[0] = 3
fd[1] = 4
fd[2] = 5
```

소켓도 파일 디스트렉 만든다는 것이다. socket()을 하나 open()을 하나 결국 모두 파일이다. 즉, socket()은 open()과 결과값이 같고, 파일 디스크립터를 반환한다. 0 은 표준입력, 1 은 표준출력, 2 는 표준에러 이기때문에 결과에는 3,4,5 가 출력 된다. **소켓은 파일이다**(리눅스 핵심).