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

<리눅스 시스템 프로그래밍>

2018.03.29 - 26일차

강사 – 이상훈
gcccompil3r@gmail.com

학생 – 안상재
sangjae2015@naver.com

1) kill 을 쉘에서 명령어를 사용하지 않고 프로세스상에서 다른 프로세스에게 시그널을 보내기 위해 사용할 수 있다.

```
/* <test.c 파일> */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

void gogogo(int voidv)
{
        printf("SIGINT Accur!\n");
        exit(0);                        // 메시지 출력 후 종료
}

int main(void)
{
        signal(SIGINT, gogogo);      // SIGINT 시그널을 받으면 gogogo 함수로 이동함.

        for(;;)
        {
                printf("kill Test\n");
                sleep(2);
        }
        return 0;
}

//////////////////////////////////////////////////////////////////////////
/* kill.c */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

int main(int argc, char *argv[])
{
        if(argc < 2)
                printf("Usage : ./exe pid\n");    // 인자가 2개이상이 아니면 메시지 출력 후 종료시킴.
        else
                kill(atoi(argv[1]),SIGINT);    /* argv[1] 인자를 정수 int 형으로 형변환시켜서 pid 로
                                                  만든후 해당 pid 에게 SIGINT 의 시그널을 보낸다.*/
        return 0;
}
```

2) sigaction() 시스템 콜 이해(signal()과 유사)

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

```
struct sigaction act_new;      // 구조체 sigaction 변수를 선언함.
struct sigaction act_old;

void sigint_handler(int signo)
{
        printf("Ctrl + C\n");
        printf("if you push it one more time then exit\n");
        sigaction(SIGINT, &act_old, NULL);      // act_old 에는 NULL 값이 들어가므로 본래의 SIGINT
                                                    시그널에 대한 signal_handler()를 처리함.
}

int main(void)
{
        act_new.sa_handler = sigint_handler; // signal_handler 등록
        sigemptyset(&act_new.sa_mask);     // 특정 signal 막음(empty-> 아무것도 안막음)

        sigaction(SIGINT, &act_new, &act_old); // SIGINT 가 들어오면 act_new handler 실행 ,
                                        act_old 는 현재 아무것도 없음.(sigaction 을 처음 실행했기 때문에)

        while(1)
        {
                printf("sigaction test\n");
                sleep(1);
        }

        return 0;
}

/*
sa = signal action
*/
```

3) 각각의 스레드는 독립적인 task 를 수행함.

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

void *task1(void *X)
{
        printf("Thread A Complete\n");
}

void *task2(void *X) // 인자, 반환값 어느것이든 가능
{
        printf("Thread B Complete\n");
}

int main(void)
{
        pthread_t ThreadA, ThreadB;
```

```
	pthread_create(&ThreadA, NULL, task1, NULL);  // ThreadA 가 task1 구동시킴.
	pthread_create(&ThreadB, NULL, task2, NULL);

	pthread_join(ThreadA, NULL);	// ThreadA 를 메모리에 올림.
	pthread_join(ThreadB, NULL);

	return 0;
}

/*
pthread_t 은 pid_t 과 유사한 자료형임.
컴파일 방법 : gcc thread.c -lpthread
*/
```

4) 서버-클라이언트 TCP 통신

→ 서버 프로세스에서 자기 자신의 ip 주소 셋팅.

→ 클라이언트의 접속을 기다림.

→ 클라이언트 프로세스에서 서버의 소켓 파일과 연결함.

→ 서버 프로세스에서 클라이언트의 접속을 수용해줌.

→ 서버 프로세스에서 클라이언트 서버에 존재하는 원격의 fd 에 메시지를 write 함.

→ 클라이언트 프로세스에서 fd 파일의 메시지를 read 함.

→ 서버, 클라이언트 프로세스 모두 fd 파일을 close 함.

```
/* <basic_server.c> 파일 */
#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; // 소켓(family,addr,port 멤버가 있음)
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 구조체
	si clnt_addr;
```

```
	socklen_t clnt_addr_size;

	char msg[] = "Hello Network Programming";

	if(argc != 2)
	{
		printf("use : %s <port>\n", argv[0]); // port 는 통로
		exit(1);
	}
	serv_sock = socket(PF_INET, SOCK_STREAM, 0);  /* 파일 디스크립터(fd)를 반환함.
                                                  SOCK_STREAM 으로 소켓을 초기화하면 TCP 를
                                                  쓰겠다*/
	if(serv_sock == -1)
		err_handler("socket() error");

	memset(&serv_addr, 0, sizeof(serv_addr));
	serv_addr.sin_family = AF_INET;    // tcp 소켓 형식
	serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  //inaddr_any 어떤 ip 주소도받겠다(자기자신을 받겠다)
	serv_addr.sin_port = htons(atoi(argv[1]));    // 어떤 서비스를 열것인가?

	if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1) // 서버에 ip 주소 셋팅
                                                  (즉, 127.0.0.1을 받겠다)
		err_handler("bind() error");

	if(listen(serv_sock, 5) == -1) // 클라이언트 5개의 접속을 기다림(5명 이상은 안됨)
		err_handler("listen() eror");
	/* listen 후에 accept 를 함*/
	clnt_addr_size = sizeof(clnt_addr);   // 32
	clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size);  // 클라이언트의
접속이 들어온것을 수용해줌, 클라이언트의 주소가 잡힘, clnt_sock 에 fd(원격의)가 넘어옴
	/* accept 인자에 주소가 들어감. 함수 안에서 처리함. */
	if(clnt_sock == -1)
		err_handler("accept() error");

	write(clnt_sock, msg, sizeof(msg));  // fd 에 메시지 적음(원격에 있는 client 에 write 함) 결국
                                         client 에 메시지가 나오게 됨.
	close(clnt_sock);
	close(serv_sock);

	return 0;
}
/* 컴파일 방법 : gcc -o clnt basic_client.c
                 gcc -o serv basic_server.c
                 터미널 2개 띄우고 ./serv
                                   ./clnt  */
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
/* <basic_client.c 파일> */
#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)    // 어디에 접근하는지 알아야해서 (옆사람 ip 주소)
        {
                printf("use : %s <IP> <port>\n", argv[0]);
                exit(1);
        }
/*네트워크의 socket = 파일의 open*/
        sock = socket(PF_INET, SOCK_STREAM,0);       // fd 얻음(네트워크상의 fd) sock=fd

        if(sock == -1)
                err_handler("socket() error");

        memset(&serv_addr,0,sizeof(serv_addr));    // serv_addr 초기화 , 포트번호 초기화, TCP/UDP
                                                   선택
        serv_addr.sin_family = AF_INET;          // 주소 체계
        serv_addr.sin_addr.s_addr = inet_addr(argv[1]);      // 입력한 ip 주소 들어감
        serv_addr.sin_port = htons(atoi(argv[2]));    // 포트번호 7777 들어감

/* sock 은 자기자신에 대한 네트워크 fd*/
        if(connect(sock,(sap)&serv_addr, sizeof(serv_addr)) == -1)  // 서버 프로세스에서 LISTEN 에서
                                                          받고 ACCEPT 하면 서버-클라이언트 통신가능해짐.
                err_handler("connect() error");
```

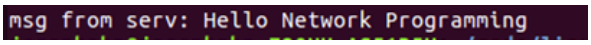
```
        str_len = read(sock, msg, sizeof(msg)-1);  // read 는 블로킹, msg 는 서버에서 보낸  메시지
        if(str_len == -1)
                err_handler("read() error!");

        printf("msg from serv: %s\n", msg);
        close(sock);

        return 0;
}
```

4-1) 결과 분석

- 포트번호를 7777 로 정한다. 터미널 창을 2개 열어서 한쪽에서는 ./serv 7777(서버 프로세스) , ./clnt 127.0.0.1 7777 을 입력 한다. 127.0.0.1 은 ip 주소이다.



5) 리눅스의 철학은 “모든것이 파일이다!” 이다. 결국 소켓도 파일과 같다고 볼 수 있으므로 socket() 은 open()과 결과값이 같고, fd(파일 디스크립터)를 반환한다. 0은 표준입력, 1은 표준출력, 2는 표준에러 이기때문에 결과에는 3,4,5가 출력된다.

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

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

	fd[0] = socket(PF_INET, SOCK_STREAM,0);	// STREAM 은 TCP
	fd[1] = socket(PF_INET, SOCK_DGRAM,0);	// DGRAM 은 UDP
	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;
}
```

6) 통신중이던 라우터가 끊어졌을 때, 서버 프로세스가 소켓 fd 에 write 한 메시지를 클라이언트 프로세스에서 모두 read 하기 위한 프로그래밍 기법

```
#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 = 0;
	si serv_addr;
	char msg[32] = {0};
```

```
	int idx = 0, read_len = 0;

	if(argc != 3)
	{
		printf("use: %s <IP> <port>\n", argv[0]);
		exit(1);
	}

	sock = socket(PF_INET, SOCK_STREAM, 0);

	if(sock == -1)
		err_handler("socket() error");

	memset(&serv_addr, 0, sizeof(serv_addr));
	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)
		err_handler("connect() error");

	while(read_len = read(sock, &msg[idx++],1))  // 서버와 클라이언트의 네트워크 통신 중에
			라우터가 소손될 경우 다른 라우터로 연결해서 통신을 계속 해야됨. 그러한
			  상황에 대비해 while 문 안에서 read 를 해서 어떻게든 서버의 메시지를
			                                                  모두 read 하도록 함.
	{
		if(read_len == -1)
		{
			err_handler("read() error");
			idx--;
		 }
		 else if(read_len != -1)
			str_len += read_len;     // read 한 문자열의 길이
	}

	printf("msg from serv: %s\n", msg);
	printf("read count: %d\n", str_len);
	close(sock);

	return 0;
}
```

7)

- uint32_t htonl(uint32_t hostlong) : unsigned integer hostlong 을 host byte order 에서 network byte order 로 바꿈.

- htons 은 short 형에 대한 인자를 network byte order 로 바꿈.

- “네트워크의 바이트 정렬은 big endian 방식이다!”

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

```
int main(void)
{
	unsigned short host_port = 0x5678;  // 2바이트
	unsigned short net_port;
	unsigned long host_addr = 0x87654321;	// 4바이트
	unsigned long net_addr;

	net_port = htons(host_port);
	net_addr = htonl(host_addr);

	printf("Host Ordered Port : %#x\n", host_port);
	printf("Network Ordered Port : %#x\n", net_port);
	printf("Host Ordered Address : %#x\n", host_addr);
	printf("Network Ordered Address: %#x\n", net_addr);

	return 0;
}
```

8) inet_addr() 는 십진수 주소값을 바이너리 네트워크 바이트 순서(big-endian 32bit)로 변환해주는 함수이다.

```
#include <stdio.h>
#include <arpa/inet.h>

int main(int argc, char **argv)
{
	char *addr1 = "3.7.5.9";	// 맨 앞의 0은 출력 안함.
	char *addr2 = "1.3.5.7";

	unsigned long conv_addr = inet_addr(addr1);  // 네트워크 바이트 순서로 바꾸어줌.

	if(conv_addr == INADDR_NONE)
		printf("Error!\n");
	else
		printf("Network Ordered Integer Addr : %#lx\n", conv_addr);

	conv_addr = inet_addr(addr2);
	if(conv_addr == INADDR_NONE)
		printf("Error!\n");
	else
		printf("Network Ordered Integer Addr : %#lx\n", conv_addr);

	return 0;
}
```