TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

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

Sigation 은 signal 과 비슷

이전 핸들러를 &act_old 를 넣어줌

tip : &가 들어있으면 주소를 전달 주소를 전달했다는건 포인터를 전달한것 !
즉 그말은 전달시킨 애들을 바꿀수있다는거 (함수밖에서 바뀌겠구나 알수있음)
함수는 오직 리턴이 하나인데 포인터를 쓰면 아웃풋을 여러개 처리할수있음
&를 통해 파악할수있다

CPU 는 밴드위스랑 개수가 중요
클럭이 중요한게 아님

네트워크 프로그래밍

1.CS (client, server)
2. 토폴로지 (위상수학은 아님)
네트워크 구성도(그래프 알고리즘)
3. TCP/IP 프로토콜(구현은 이걸로해야됨) . 라우터 , 스위칭, OS // 애초에 리눅스로 설계되어서 리눅스 유닉스 최적화됨
(OSI 7 LAYER)이론

ifconfig 를 치면 Ip 를 볼수있음

inet addr

ipv4 ipv6 NAT

255 는 브로드캐스트
하나의 망안에 들어가는 수는 256 개 정도됨
IP 의종류

1. 공인 (WAN 통신) IP(
2.사설 \(공유기)

*인터넷
IP

네트워크 = 원격 IPC

MAC :
스위치 장비에 남음

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~pid 를 입력함으로서 프로세스를 죽일수있음

```
#include <stdio.h>                    // gcc -o kill kill.c
#include <unistd.h>                   // ./kill pid[숫자]

int main(int argc, char *argv[]):
{
        if(argc < 2)
                printf("Usage : /exe pid");
        else
                kill(atoi(argv[1]), SIGINT);   //들어오는숫자는 스트링이라 문자로 바꿔줌
                                               //
        return 0;

}
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

이프로그램은 Ctrl + c 를 두번 눌러야 꺼짐

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

struct sigaction act_new;
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);
}

int main(void)
{
        act_new.sa_handler = sigint_handler;            //시그널 두번째 인자에
        sigemptyset(&act_new.sa_mask);          //아무것도 막지않는것  empty (시그널을 막아야하는
경우도있음)

        sigaction(SIGINT, &act_new, &act_old);          //인터럽스 시그널이 들어오면 액트뉴가 동작
함 액트올드는 이전에 등록했던 값
        while(1)
        {
                printf("sigaction test\n");
                sleep(1);                       //1 초마다 하나씩
        }
        return 0;
}
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~쓰레드

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

void *task1(void *X)
{
	printf("Thread A Complete\n");		//void 포인터 뭐든지 리턴가능. 인자도 리턴가능
}						//어떤것이든지 인자로 받고 리턴하겠다는 의미임

void *task2(void *X)
{
	printf("Thread B Complete\n");

}

int main(void)
{
	pthread_t ThreadA, ThreadB;		//쓰레드 지정할때 pthread_t 로 지정
						// 헤더파일 ( pthread.h 안에 들어있음 )
	pthread_create(&ThreadA, NULL, task1, NULL);//task1 을 threadA 가 구동시킬 것이라고 쓰레
드의 생김새를 만듬
	pthread_create(&ThreadB, NULL, task2, NULL);	//task2 를 threadB 가 구동시킬것
					//NULL 은 아직 신경쓰지말것
					// phtread P p 대소문자 주의!
	pthread_join(ThreadA, NULL);//실제 메모리에 올리는 구간 ThreadA 실행
	pthread_join(ThreadB, NULL);		// ThreadB 실행

	return 0;
}
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~basic_client.c ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
#include <stdio.h>		//gcc -o clnt basic_client.c
#include <stdlib.h>		//127.0.0.1 7777 은 로컬호스트 나임!
#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)
        {
                printf("use : %s <IP> <port>\n", argv[0]);
                exit(1);
        }       //


        sock = socket(PF_INET, SOCK_STREAM, 0);                 //소켓하면 파일디스크립터를 얻음 (네
트워크상의 , 내가통신할수있는 파일디스크립터 (네트워크에서는 소켓이 파일오픈과같음)
                                        //socket 는 이미 원격 IPC , 세마포어 안에있음
        if(sock == -1)
                err_handler("socket() error");

        memset(&serv_addr, 0, sizeof(serv_addr));  //서버어드레스 초기화
        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)          //sock 는 자기자신의 네크워
크 파일 디스크립터 , 이것을 서버어드레스에 연결 (
                err_handler("connect() error");

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

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

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

        return 0;
}
```

서버를 먼저 실행하고 클라이언트를 나중에 실행

클라이언트에 출력됨

```
  gcc -o serv basic_sever.c
        터미널 두개띄우고
         serv 7777
         clnt 127.0 .0.1 7777
```

gcc -o clnt read_client.c

./clnt 127.0.0.1 7777 //

~~~~~~~~~~~~basic_server.c ~~~~~~~~~~~~~~~~~~~~~~~~`

```
#include <stdio.h>              //gcc -o serv basic_sever.c
#include <stdlib.h>             //터미널 두개띄우고
                                //serv 7777
                                //clnt 127.0 .0.1 7777
#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";

        if(argc !=2)    //포트번호 입력하는소리 포트번호는 7777 통로임
        {
                //


        printf("use: %s <port>\n", argv[0]);
        exit(1);
        }

        serv_sock = socket(PF_INET, SOCK_STREAM, 0);                    //리턴 파일디스크립터

        if(serv_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 = htonl(INADDR_ANY);                  //어떤 아이피주소든 받겠다
        serv_addr.sin_port = htons(atoi(argv[1]));            //이패턴은 그냥 외우기

        if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("bind() error");   //ip 주소셋팅 bind
        if(listen(serv_sock, 5) == -1)                  //5 명 받겠다는 소리임
                err_handler("listen() error");


clnt_addr_size = sizeof(clnt_addr);
clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size);  //억셉트 클라이언트의 접속을 기다림 실제 기다리는 구간은 listen 에서 기다린다
        //클라리언트의 접속을 허용
        //accept 하면 클라이언트의 &clnt_addr, 파일디스크립터가 넘어오고,원격에 있는 파일(원격으로 세마포어를 만듬 ) 파일디스크립터를 받아서 read wirte 다할수있

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

        write(clnt_sock, msg, sizeof(msg));  //원격에있는 클라이언트한테 write 한다는 소리
        close(clnt_sock);
        close(serv_sock);

        return 0;
}


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#include <stdio.h>

int main(void)          //크로스매칭 다른 앤디안끼리 꼬이지않게
{

        unsigned short host_port = 0x5678;  //short 2 바이트 16 비트
        unsigned short net_port;
        unsigned long host_addr = 0x87654321;
        unsigned long net_addr;

        net_port = htons(host_port);   //host to network
        net_addr = htonl(host_addr);           // long 4 바이트 32 비트

        printf("Host Ordered Port: %#x\n", host_port);

        printf("network ordered port : %#x\n", net_port);

        printf("host ordered address : %#lx\n", host_addr);

        printf("network ordered address : %#lx\n", net_addr);
```

```
	return 0;
}
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~사실상 위에 한것과 비슷함
하지만 서버가 터졌을때 라우터의 우회로를 이용할수있음

```
#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] = {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));  //서버어드레스 초기화
	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)		//sock 는 자기자신의 네크워
크 파일 디스크립터 , 이것을 서버어드레스에 연결 (
		err_handler("connect() error");
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
	while(read_len = read(sock, &msg[idx++], 1))
	{
		if(read_len == -1)
			err_handler("read() error!");
		str_len += read_len;		//중간에 끊겨도 우회를 해서 받아오는것
	}

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

	return 0;
}
```

크로스 매칭

03. 07 . 05 .09

09 05 07 03

애초에 메모리에 들어가는 형식으로

읽어들일때는 변경한 방식으로 그대로 읽어들이면됨 그러면 꼬일일이없음

빅앤디안
리틀앤디안은 서로 반대