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

강사 - Innov (이상훈)

gcccompil3r@gmail.com

학생 - 이유성

dbtjd1102@naver.com

1. 리눅스 시스템 프로그래밍

sigaction()-sytstemcall

```
#include<stdio.h>
#include<signal.h>
//전역변수로 선언
struct sigaction act_new; //설정할 행동 즉, 새롭게 지정할 처리 행동 ,,
sigaction = signal이랑 동일하게 사용할 수 있는 것중하나(차이는
?구조체)
struct sigaction act_old; // 이전 행동 이함수를 호출하기전에 지정된
행동 정보가 입력됩니다

void sigint_handler(int signo)
{
        printf("Ctrl+C\n");
        printf("If you push it one more tiome then exit\n");
        sigaction(SIGINT,&act_old,NULL);
 //sigint들어오면 동작  2번째 인자 NULL 꺼짐
}

int main(void)
{
        act_new.sa_handler = sigint_handler; //sigint_handler
시그액션의 2번째 인자처럼( sigint_handler함수행동을sa_handler에
넣는다)

//sa_handler -signum번호를 가지는 시그널이 발생했을 때  실행된
함수를 설정한다
sigemptyset(&act_new.sa_mask);//sa_handler에 등록된 시그널 핸들러
함수가 실행되는 동안 블럭(signal에서 ign에 해당)되어야 하는시그널의
마스크를 제공한다(sa_namask가 적용되어 있지 않다면)
// sigempty 아무것도 막지 않겠다는 뜻 (막아야하는 경우는
어떤경우일까? 중요한 정보 처리할때-) -함수의 인자에 포함된
signal지운다
        sigaction(SIGINT,&act_new,&act_old);
 // 전sigaction의 (행동 정보) 를 리턴해서 act_old에 저장.->NULL
        while(1)
        {
                printf("sigaction test\n");
```

```
                sleep(1);
        }
        return 0;
}
```

// 파일을 빠르게 파악하는 . & (주소) 를 본다 -> 포인터를 전달한 것들은 함수안에서 값이 바뀌겠구나(셋팅되겠구나)예상가능.

// 함수는 return 하나 . 포인터는 리턴을 여러개 같이 가능(내용 변경,리턴 여러개 예상가능)

```
kill Test
kill Test
./kkill Test
illkill Test
 32kill Test
15
SIGINT Accur!
[1]+  완료                ./test
```

2. 다른 프로세스로도 kill할 수 있다.

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>
void gogogo(int voidv)
{
	printf("SIGINT Accur!\n");
	exit(0);
}

int main(void)
{
	signal(SIGINT,gogogo);

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

```
//kill.c

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

int main(int argc,char *argv[])// 문자열로 인식
{
	if(argc<2)
		printf("Usage :./exe pid\n"); //./a.out
pid
	else
		kill(atoi(argv[1]),SIGINT);//문자열->int
	return 0;
}

//컴파일 실행 방법
//gcc -o test test1.c
//./test & or ps -ef |grep test
// ./kill pid[숫자]
```

결과

```
kill Test
kill Test
./kkill Test
illkill Test
 32kill Test
15
SIGINT Accur!
[1]+  완료                ./test
```

3.thread 종속적이다

```
#include<pthread.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
void *task1(void *X)//뭐든지 리턴할 수 있고 인자도 뭐든지 받을 수 있다.
//자료구조에서 재귀호출 해제하면서 push할 때 void포인터로 받고 pop할 때 강제 캐스팅 했다
{
	printf("Thread A complete\n"); //스레드는 자신의 스레드 ID를 얻을 수 있따.
}

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

int main(void)
{
	pthread_t ThreadA,ThreadB;

	pthread_create(&ThreadA,NULL,task1,NULL);//THREAD A,B값을 채운다 , task1은 thread A가 구동 ,create(형태를 만듬)
	pthread_create(&ThreadB,NULL,task2,NULL);//Thread 이렇게 구동시키겠다 등록 (생김새만 만들어놓음)

	pthread_join(ThreadA,NULL);//조인을 하는순간 메모리에 올림
	pthread_join(ThreadB,NULL);

	return 0;
}

// gcc thread.c -lpthread  <<뭔뜻??
```

//나중에 병렬에서 쓰레드를 이용해서 연산을 극대화 할 것,

// 병렬 처리 그래픽카드 clock개수 중요x ,, 밴드개수 ? 가 중요.
// cpu가 순차처리->그래픽카드에 cpu가 많아서 한번에 처리가능
// 빠르게 처리는 DSP

//프로세스 vm이 독립적. thread종속적 ->메모리 공유 -critical section충돌 -lock걸어줌

결과

Thread B complete
Thread A complete

네트워크 프로그래밍 (팀장님꺼 참고 함)

1. cs(client ,server)
2. 토폴로지 (위상수학 제외)

   -네트워크의 요소(링크,노드 등)을 물리적으로 연결해 놓은 것 또는

   - 그 연결 방식(구성도)을 말한다

   - 그래프 알고리즘

3.TCP/IP 프로토콜 - OSI 7 Layer - 버클리 ->TCP/IP 4계층

(TCP/IP프로토콜)

이론적으로는 OSI 7 Layer 실제로는 4계층으로 구현

만들 때 리눅스에 포팅하여 만들어 짐.

리눅스 ,유닉스에 최적화 됨.그래서 네트워크 장비들은 리눅스나 유닉스를 사용.

*인터넷

물리적으론 랜선

*ip (인터넷 프로토콜) -(ifconfig입력)-> inet addr

공인 ip : wan통신

사설 ip : 공유기

ipv4 : 127.0.0.1 (32비트)(0~255)씩 4자리를 끊어 쓴다

0번 ->gateway 255번->broadcast로 예약됨.(하나의 망 안에 250여개 들어감)

게이트웨이란 서로 다른 네트워크로 들어가는 네트워크 포인트
라우터 : ip packet이 원하는 목적까지 원활하게 갈 수 있도록 경로를 정해주는 역할을 하는 장비

ipv6 :ip4 프로토콜 주소가 32비트로 제한(2의 32승 개)되어 주소 공간 및 국가별로 할당된 주소가 거의 소진되고 있다는 한계점으로 인한 것
-휴대폰,가전제품,스마트홈,스마트카 등등 ip통신하는 장치들이 늘어나면서 모두 공인 ip를 필요로 하게 된다.
자동차에 공유기를 따로 달아줄 수는 없다
즉 센서 네트워크를 구축하기 위해 만들어진 것이다.

NAT -NAT프로토콜(공인ip , 사설ip)
공인 ip는 하나인데 여러명이 인터넷을 사용할 수 있는 이유는 NAT프로토콜 덕분이다.

MAC통신과 IP통신(스위치 장비와 라우터 장비)
*MAC통신이란
-MAC:HW장치의 고유번호 ->LAN카드의 고유번호 ,식별자
스위치에 MAC주소가 흔적으로 남는다
(MAC주소를 보고 어디로 보낼지 결정하는게 스위치이기 때문 - MAC을 관리)
ip를 보고 경로설정을 하는 것은 라우터

-스위치에 MAC을 찾고 싶다고 요청하면 Braodcating한다(DDOS공격이 가능한 이유)

해당 MAC주소를 가진 기기가 반응하면 세션이 맺어지면서 통신이 이뤄짐

스위치가 본인의 MAC Table을 검색하여 요청받은 MAC을 찾지 못하면
(스위치는 192.168.0.X대역을 관리한다)
관리하는 대역이 아니므로 라우터에 ip를 보내어 경로를 물어본다
해당 ip를 가진 장치가 이를 인식하면 통신이 이루어진다
-> 사실상 이과정은 ipc이다
즉 네트워킹은 많이 느린 IPC이다 (외부의 사람과 통신이 가능하다)

ARP스푸핑 : ip충돌(중복으로 인해)나는 스위치는 MAC을 찾을 수가 없다
-> 스위치는 계속 Broadcating한다 ->전기신호는 ADC를 통해 디지털 신호로 바뀌어 메모리에 쌓이게 된다 -> 계속 쌓이면 서버가 뻗는다
(이를 막기 위하여 스위치 대역폭을 넓혀서 공격자를 인식하고 접속하지 못하도록 차단한다)

4. socket : 서로 다른 통신환경을 이어주는 것

```
#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 ); //fd
파일식별자,socket 은 open과 비슷한 개념,socket도 파일만듬
                                                  // 0 ?
        fd[1] = socket(PF_INET, SOCK_DGRAM,0); //stream
=tcp/ip프로토콜 , dgram= udp/ip프로토콜  알필요 없다
        fd[2] = open("test.txt" ,O_CREAT | O_WRONLY | O_TRUNC);
//trunc

//PF_INET = Ipv4인터넷 프로토콜을 사용합니다?

        for(i = 0; i<5; i++ )  //i = 3
                printf("fd[%d] = %d\n",i,fd[i]);

        for(i = 0 ; i <5 ; i++)
                close(fd[i]);
        return 0;
}


// fd[i] = 0   ?  결과 .. 3 4 -1 0 ~~


//**결론 모두 파일이다.task_struct ->file_struct ->file

// ls -l perm자리에 이스속 걸렸을 때 s걸려 ->socket파일이라는 것.
```

5. basic_server.c (서버 정리가 아직 덜 되었습니다..)

```
#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;  //si poket address
	socklen_t clnt_addr_size;//길이 값,

	char msg[] = "Hello Network Programming"; //전달하려는 메시지

	if(argc !=2)//2개아니면
	{
		printf("use : %s <port>\n",argv[0]);//포트번호 입력하라
포트번호 7777(통로) 권한은 아니고
// (웹)80=www ,10 20ftp업로드 , 22=ssh다운로드   포트번호 (특정역할
service)  7777우리가 만든 전용 커스텀. 즉service번호
		exit(1);
	}
```

```
        serv_sock = socket(PF_INET,SOCK_STREAM,0); //ipv4를쓴다 ,
,0(특정 프로토콜 사용하기 위한 변수 ) 2번째 인자로
연다.tcp/ip로연다?(통신프로토콜)사용하는 socket 파일 디스크립

        if(serv_sock == -1) //socket =(파일) 리턴값= 파일
디스크립0터(오픈이랑 결과값 같음)
                        //가 나온다.         네트워크 = 원격IPC
                err_handler("socket() error");//

        memset(&serv_addr , 0 , sizeof(serv_addr));  //serv_addr 구조체
si라는 si sockaddr_in 이라는 정보 3개
                //사이즈를 0으로          //포켓 페밀리(port서비스  포켓
어드레스(어떤 ip주소든지 다 받겠다
// (자기 자신이 되겠다.127.0.0.1 로local hose (나))
        serv_addr.sin_family = AF_INET;           //memset부터
4줄(포켓사이즈(만드는 형식)은 형태 똑같으니까 패턴 익혀 다르게 쓸
일은 없다.
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons(atoi(argv[1])); //변수3개
패밀리:af_inet ==ipv4형태의 주소를 사용하겠다)
어드레스:127.0.0.01로컬 어드레스  포트:내가 입력한 포트를 쓰겠다.그
포트 = 통로(7777) (클라이언트) 로 다른사람이 들어올 수 있다.

        if(bind(serv_sock , (sap)&serv_addr,sizeof(serv_addr)) ==-1
)//bind=ip주소 셋팅하는 것.->서버에 ip셋팅
                                                        //자기 주소로
정하겠다
                err_handler("bind() error");
        if(listen(serv_sock , 5) == -1)//listen 5명 받겠다 . 5명 넘으면
안됨.
                err_handler("listen() error");

        clnt_addr_size = sizeof(clnt_addr); //32
        clnt_sock = accept(serv_sock , (struct sockaddr
*)&clnt_addr,&clnt_addr_size);//서버 소켓이 클라이언트의 접속을
기다림
                        //실제로 클라이언트가 기다리는 구간 listen을 기다림
(패턴을 봐라 해석하지말고 쓸 때없이)
                        // 실제로 클라이언트 접속허용이 이루어지는구간
accept  .. listen -> accept넘어옴
```

```
                        //accept하면  클라이언트의 주소가 잡힌다
터i                     //어쎕트 성공하면 접속에 성공한 클라이언트의 소켓
파일디스크립터를 반환.
        //그러면 똑같은클라이언트의  파일 디스크립터를 받음.
        if(clnt_sock ==-1)
                err_handler("accept() error");

        write(clnt_sock,msg,sizeof(msg));// socket은 파일(원격에 있는
파일) 네트워크 원격으로 동기화(semaphore)
                                        //파일 디스크립터 받으면 읽고 쓰기 다
가능,
                                        //메유세지 적어 원격에 있는
클라이언트에서 메시지가 나온다
        close(clnt_sock);
        close(serv_sock);

        return 0;
}

// gcc -o clnt basic_client.c
// gcc -o serv basic_server.c

// 터미널 2개 띄우고 ./serv 이후 ./clnt
//./serv 7777                   - 7777로 포트를 열어?
// ./clnt 127.0.0.1 7777


// 서버 닫는데 시간 걸리나봄.
```

*INADDR_ANY : 자기자신을 주소로 받겠다.?127.0.0.1(로컬호스트 = 나)

와 같은 의미

serv_addr.sin_port = htons(atoi(argv[1]));

스코프 바인딩 : 해당스코프를 지정하겠다

네트워크의 바인딩:ip주소를 셋팅한다 (바인드 하면 서버에 ip가 셋팅된다

127.0.0.1이 셋팅

6.basic_client.c (정리가…)

```
#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) // ip주소 ./clnt 127.0.0.1 7777
	{
		printf("use : %s <IP> <port>\n",argv[0]);
		exit(1);
	}
	sock = socket(PF_INET,SOCK_STREAM, 0);// 소켓하면
네트워크상 파일 디스크립터 반환받음. 한마디로 내가 통신할 수 있는
파일 디스크립터 그걸 sock에 집어넣는다(=fd) socket-기능은 open과
같다.

	if(sock ==-1)
		err_handler("socket() error");
	memset(&serv_addr, 0, sizeof(serv_addr));//멤셋을 한다. 서버
어드레스를 초기화,
```

```
        serv_addr.sin_family = AF_INET;//사설주소셋팅.TCP를 쓸것이다
        serv_addr.sin_addr.s_addr = inet_addr(argv[1]);//아이피주소
        serv_addr.sin_port = htons(atoi(argv[2]));//어떤 서비스를 열
것인가(포트번호 7777)  &serv_addr은 si구조체 안에 위의 3개 멤버가
들어있음

        if(connect(sock,(sap)&serv_addr,sizeof(serv_addr)) ==
-1)//커넥트를하면 서버의 listen에서 받는다 accept한다
// accept하면 서버랑 클라이언트랑 통신이 된다. 서버가 write해서
메시지 하나 보냄
                err_handler("connect() error");
        str_len = read(sock , msg,sizeof(msg) -1);//서버에서
write했으니까(버퍼에서 msg로 전달->sock으로)  read
                                        //파일 디스크립터에서 읽어옴
msg로 가져옴 msg에는 서버에서 보낸
                                        //메시지가 들어있음
        if(str_len ==-1)
                err_handler("read() error!");

        printf("msg from serv : %s\n",msg);
        close(sock);
        return 0;
}

//socket안에 semaphore들어있음 이것은 원격 ipc

//이 패턴은 안 바뀔거다
```

serv_addr와 connet하겠다 - >서버의 listen에서 받아서 accept하여

서버와 클라이언트간 통신이 된다.

서버에서 write했으니까 read (read는 블록킹함수가 들어올 때까지

움직이지 않는다)

7. read_client.c (정리가 덜 되었습니다..)

read 중간에 데이터의 손실이 없게 만들기

```
#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)) //추가됨.   예외적 ,
16 -(끊힘) > 12 , 4 총길이는 read로 먼저 계산.. 들어오다가 폭격..
        {
                if(read_len == -1)
                        err_handler("read() error !");

                str_len +=read_len;
        }
        printf("msg from serv :%s\n",msg);
        printf("read count : %d\n" ,str_len);
        close(sock);
        return 0;
}


cat inet_addr.c//이것도 convert
//inet_addr() IP문자열에서 long형 숫자 IP주소를 구함
```

네트워크 상에서 예기치 못한 문제 발생 , 중간에 끊기게 되면 라우터가 우회하여 전송시킨다 이때 그래프 알고리즘이 사용된다 (다른 것들과 연결되어서 끊어져도 괜찮다.)

data 크기가 전부 16일 때, 12바트이만 들어오고 나머지 4바이트가 남은상황에서 read_len은 12바이트 str_len도 12바이트 다시 read에 올라가서 남은 4바이트를 받아온다 -> 중간에 끊기더라도 데이터의 손실이 없게된다.

8.convert_endian.c 변수에 저장되는 타입에 따라 리틀엔디안과 빅엔디안으로 나뉜다.예제)

```
int main(void)
{
	unsigned short host_port = 0x5678;//2바이트
	unsigned short net_port; //리트엔디안의 저장 방식.->78 56
	unsigned long host_addr = 0x87654321;//4바이트
	unsigned long net_addr;

	net_port = htons(host_port);//short 2byt   1byte씩 읽음.
	net_addr = htonl(host_addr);

	printf("Host Ordered Port : %#x\n",host_port); //x는 2byte로
처리해라
	printf("Network Ordered Port : %#x\n" ,net_port);
	printf("Host Ordered Address : %#lx\n",host_addr);//lx는 4byte로
처리해라
	printf("Network Ordered Address : %#lx\n",net_addr);

	return 0;
}


//엔디안 : 데이터 저장 순서 방식
// 방향 0x12345678 data가 있다고 하면 빅엔디안 같은경우 그대로
읽어 이렇게 저장을함 메모리에
//  리틀엔디안 같은 경우는 뒤에서 부터 1byte씩 저장 2글자씩 받아
// 78 56 34 12
// 1. 효율 때문에 나눔.

//2. 이걸 하는 이유 cpu마다 엔디안이 다르다 빅-리트엔디안
정보교환할 시 포멧형식이 없으면 정보가 꼬이니 꼬이지 말라고
바꿔주는 것.
실제 메모리에 배치되는 형태로 크로스 매칭한다.
```

..그래픽 cpu얘기 추가.할 것

9.inet_addr()

```
//이것도 convert

//inet_addr() IP문자열에서 long형 숫자 IP주소를 구함

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

int main(int argc, char **argv)
{
	char *addr1 = "3.7.5.9"; //주소체계   big,lit 엔디안 매핑,? 공유한다고 한다면, 네트웤 안 할시 꼬입니다
                                 // 맨앞에 0 출력할 필요 없죠.?
	char *addr2 = "1.3.5.7";
	unsigned long conv_addr = inet_addr(addr1);  //반환 -1실패 외 long형의 IP주소
	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;
}
결과
Network Ordered Integer Addr :0x9050703
Network Ordered Integer Addr :0x7050301
```

inet_addr()함수

-함수 파라미터 값에 IP주소 문자열의 시작주소를 넣어주면 이 함수가 알아서

빅엔디안 32비트 unsigned long형의 값으로 만들어 줍니다

-성공하면 빅엔디안 형식의 32비트 값을, 실패하면 INADDR_NONE을 리턴.