Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정
#26

강사:Innova Lee(이 상훈)
학생: 김시윤

수업내용 복습

---sigaction---

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

struct sigaction act_new;
//전역변수로 선언
struct sigaction act_old;

void sigint_handler(int signo)
//맨처음에 받아올때 엑트 올드에 아무것도 없으니 두번누르면 리턴을 해줄수 있는곳이 없으니 죽는다.
{
	printf("Ctrl + c\n");
	printf("If you push it one more time then exit\n");
	sigaction(SIGINT, &act_old, NULL);
//처음에 들어갔을때 sigaction 의 act_old 값이 없기 때문에 시그널이 발생되지 않는다.
//두번째 들어왔을때 act_old 에 SIGINT 값이 있기 때문에 Sigint 를  발생 하여 프로세스를종료한다.
}

int main(void)
{
	act_new.sa_handler = sigint_handler;
//act_new.sa_handler 에 sigint_handler 를 저장.
//시그날의 두번째 변수. 시그널의 첫번째변수는 시그날 번호 / 시그날 핸들러가 2 번째 인자. 사용할 핸들러 등록
	sigemptyset(&act_new.sa_mask);
//특정한 시그날이 들어오지 못하게 막는거 근데 EMPTY 니 아무것도 막지 않겠다 . *시그날 막는경우 중요해서 무조건 먼저 처리할때.

	sigaction(SIGINT, &act_new, &act_old);
//sigaction(시그널 번호 , 설정할 행동, 이전 행동);

//뭔가를 작업함//시그인트가 들어오면 위에 있는거 동작함 엑트올드는 이전에 구동시켰던 시그날 핸들러 관련된 정보들 처음 동작했을때 엑트 올드에 아무것도 없다.
	while(1)
	{
		printf("sigaction test\n");
//이동작중 컨트롤 씨 누르면 위함수로 간다.
		sleep(1);
	}
	return 0;
}

/*시그널이랑 비슷한다 시그널은 이전행동을 리턴하는데 시그엑션은 이전행동을 엑트올드에 넣어준다

남이 만든 코드를 분석할때 쉽게 보는법 &를 보면됨 &가 들어있다는건 주소를 전달 포인터를 전달했다.
그렇다는건 저안에서 &로보낸 애들을 변경시킬수 있다는것. &가 들어있는 애들은 저함수 안에서 값이 바뀌거나
값이 셋팅되겠구나 라고 생각, 함수는 리턴이 오직 하나다. 포인터를 쓰면 여러개에 대한 처리를 같이 할수 있다.
&들어있는걸 보면 함수를 통해서 값을 받아오거나 값이 변결될수 있겠구나 라고 생각하세요.
*/
```

---thread---

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

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

```
}

void *task2(void *X)
{
	printf("Thread B Complete\n");
}
int main(void)
{
	pthread_t ThreadA, ThreadB;
//pid 만드는거랑 같다. thread 두개를 만들기위해 아이디생성

	pthread_create(&ThreadA,NULL,task1,NULL);
//쓰레드 생성 포크랑 다른게 &가 threadA 와 B 에 먼가 값을 채워넣는다. void 포인터 리턴도 뭐든지 할수있고 인자도 모든지 받을수 있다.
//제귀해제하면서 푸시 하고 팝했었다.어떤것이든지 인자로 받고 리턴하겠다. 쓰레드 에이가 태스크 원을 구동시킨다는 뜻 이렇게 구동시키겠다고 등록만 해놓은것 쓰레도 생긴새 만든것.

	pthread_create(&ThreadB,NULL,task2,NULL);
	//thread 생성
	pthread_join(ThreadA,NULL);
//조인을 하는순간 메인을 올린것.
	pthread_join(ThreadB,NULL);
	//A 포로세스 또는 B 포로세스가 조인을 하면 위에 task 함수로 올라가 출력한다.
	return 0;
}
/* thread 는 종속적 메모리를 공유한다.
큰문제가 발생하는 구간 크리티컬 세션.
그래서 락을 걸어 줬었다.

병렬처리 할때 쓰레드 이용해 연산을 극대화 시킴
시피유 하나에 그래픽카드 천개 만개 된다.
쥐피유는 클럭스피드가 떨어지지만 대역폭이 넓음. 대역폭이랑 갯수가 중요하다 쥐피유는 그래픽카드 병렬처리
씨피유는 순차처리에 특화
*/
```

--- KILL ---

Test.c

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


void gogogo(int voidv)
{
	printf("SIGINT Accur!\n");
	exit(0);
}

int main(void)
{
	signal(SIGINT, gogogo);
//시그날 등록 시그날 들어오면 함수로 가라.

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

		return 0;
}
/*영상처리나 모터제어 프로세스가 있다면 피더블유엠을 조정시키면
세어드 메모리 만들어놓고 시그날 발생시켜 빨리 이값 읽어다가
피더블유엠 조절해라
그럴때 킬 시그날 날려줘서 빠르게 조절.
물론 시그날이 등록되어있어야함.
*/
```

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

```c
#include <stdlib.h>
#include <signal.h>

int main(int argc, char *argv[])
{
	if(argc <2) // 0,1 만 사용 ./kill pid
		printf("Usage : ./exe pid\n");
	else
		kill(atoi(argv[1]),SIGINT);//sigint 를 이 피아이디를 날려라
	/*atoi(argv[1]) 에 SIGINT 를 날린다.
	  int kill(pid_t pid, int sig);
	  argv[1]에 프로세스 아이디를 입력하기 때문에 그 아이디로 SIGINT 가 날라간다.*/
	return 0;
}
```

---데이터 통신 ---

basic_server.c

```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 Embedded Class";//전달할라는 문자열

	if(argc !=2)
//두개가아니면 오류내라 <포트번호 입력해라> 포트번호는 통로 80 은 웹
//20 업로드 ,21 다운로드  ftp 22ssh 쉽게는 포트는 서비스 웹브라우저 접속하면 포트번호 무조건 80
//7777 은 우리가 만든 전용 커스텀 포트 서비스 번호 포트번호 겹치면 값이 이상해진다.
	{
		printf("use %s <prot> \n", argv[0]);
//서버랑 포트 입력해라.
//네트워크는 원격 아이피 모든것은 파일이다 소켓도 파일이다 파일 디스크립터가 넘어옴
		exit(1);
	}

	serv_sock = socket(PF_INET, SOCK_STREAM, 0);
//SOCK_STREAM TCP 사용
//PF_INET -> ipv4 인터넷 프로토콜
	if(serv_sock == -1)
//소켓이 리턴하는건 파일디스크립터 오픈이랑 똑같음
		err_handler("socket() error");

	memset(&serv_addr , 0, sizeof(serv_addr));
//구조체였음 si 라는 안에 세개 정보있음
//소켓 패밀리 소켓어드레스 소켓 포트 포트는 (서비스번호) 어떤 서비스를 줄것인가에 신포트에 넘김
```

```
//INADDR_ANY 어떤 주소든 받겠다 127.0.0.1 로컬주소 내주소
        serv_addr.sin_family = AF_INET;
//AF_INET ipv4 인터넷 프로토콜//4 바이트 주소체계
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
//INADDR_ANY 어떤 주소든 받겠다.
        serv_addr.sin_port = htons(atoi(argv[1]));
//memset , sin_family , sin_addr, sin_port 합쳐서 tcp
/*
htons(short s) -> short 데이터를 호스트의 바이트 순서에서 네트워크 바이트 순서로 변환해라.

ntohs(short s) -> short 데이터를 네트워크 바이트 순서에서 호스트 바이트 순서로 변환해라.

htonl(long s) -> long 데이터를 호스트의 바이트 순서에서 네트워크 바이트 순서로 변환해라.

ntohl(long s) → long 데이터를 네트워크 바이트 순서에서 호스트 바이트 순서로 변환해라.

*/
//위에 네줄 티시피 소켓 만들 기본 구조


        if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr))==-1)
//바인드 해당 아이피 셋팅하는거
//서버에 아이피 주소 셋팅하는게 바인드. 바인드하게되면 서버가 아이피 셋팅됨 127.0.0.1
                err_handler("bind() error");
        if(listen(serv_sock, 5) == -1)
//5 명 받겠다. 다섯명 이상 받으면 안됨//클라이언트 기다리다 클라이언트가 들어오면 접속 허용은 어셉트가 한다.
                err_handler("listen() error");
        clnt_addr_size = sizeof(clnt_addr);//30
        clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr,
&clnt_addr_size);
//클라이언트 접속을 기다린다 기다리다 어셉트 해준다 실제로 기다리는 구간 은 리슨 에셉트 안에서 저부분에 &값들을 채워줌

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

        write(clnt_sock, msg, sizeof(msg));
//클라이언트 소켓의 파일디스크립터가 들어와 원격에 있는 파일이 들어옴 원격 통신 ipc 원격으로 세마포어 만들어서 파일디스크립터 받음 그쪽 파일디스크립터에다가 라이트 메세지 클라이언트 소켓으로 날아감 원격에 있는 클라이언트한테 라이트를 한다.
        close(clnt_sock);
        close(serv_sock);

        return 0;
}
```

---

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)
//아이피주소가 어디에 접근하는지 알아야 해서 세개다 192.168.02.x 사설.
	{
		printf("use : %s <IP> <port>\n", argv[0]);
		exit(1);
	}

	sock = socket(PF_INET, SOCK_STREAM, 0);
//네트워크상의 파일디스크립터를 얻음 내가 통신을 할수있는 파일디스크립터 가져와서 소크에다 집어넣음 소크가 fd 와 같음 소켓은 오픈이랑 같음.

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

	memset(&serv_addr, 0, sizeof(serv_addr));
//서버 어드레스 초기화하고 //sizeof(serv_addr)이 사이즈만큼 다 0 으로 셋팅
	serv_addr.sin_family = AF_INET;
	serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
/*inet_addr(const char* string);
성공시 빅에디안으로 변환된 32 비트 정수 값, 실패시 INADDR_NONE 반환.*/

	serv_addr.sin_port = htons(atoi(argv[2]));
htons(short s) -> short 데이터를 호스트의 바이트 순서에서 네트워크 바이트 순서로 변환해라.


	if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
		err_handler("connect() error");

	str_len = read(sock, msg, sizeof(msg) -1);
//read 는 블로킹 들어올때까지 안움직임 fd 에서 읽어옴 sock 에서 msg 읽음

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

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

	return 0;
}
```

---

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

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

	if(argc != 3)//3 개가 아니면
	{
		printf("use: %s <IP> <port> \n",argv[0]);//파일이름 아이피 포트를 입력하라
		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))
//read_len 을 체크한다
//원격지에있는 라우터가 터졌다한다. 끊힌 데이터는
	{
		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;
}
```

//처음에 커넥트할때 전체 길이를 검사해서 처음 길이를 알수 있다 .
//user 컴퓨터와 라우터가 연결 되어있는데 데이터를 송신할때, 원래 연결되있던 라우터와의 접속이 끊히면 인접하는 다른 라우터에 데이터를 우회한다.
만약 데이터가 들어오고 있는 도중에 라우터와의 접속이 끊히면 남은 데이터는 다른 라우터를 통해 데이터가 우회해서 들어온다. 그래서 while 문을 통해 늦게 들어온 데이터까지 읽을 수 있게 처리한 소스코드이다. 처음에 connect 했을때 데이터의 전체 길이를 측정해 sock 에 저장했기 때문에 sock 에서 읽은 크기만큼의 값이 들어오지 않으면 데이터가 들어올때까지 기다렸다가 나중에 들어온 데이터를 더해준다.

TCP/IP 4 계층.

응용계층(Application) (Telent,FTP,HTTP,SMTP ….)
전송계층(Transport) (TCP,UDP)
인터넷 또는 네트워크 계층(Internet or NETWORK) (IP,MAC)
물리계층 (Network Driver, Hardware)

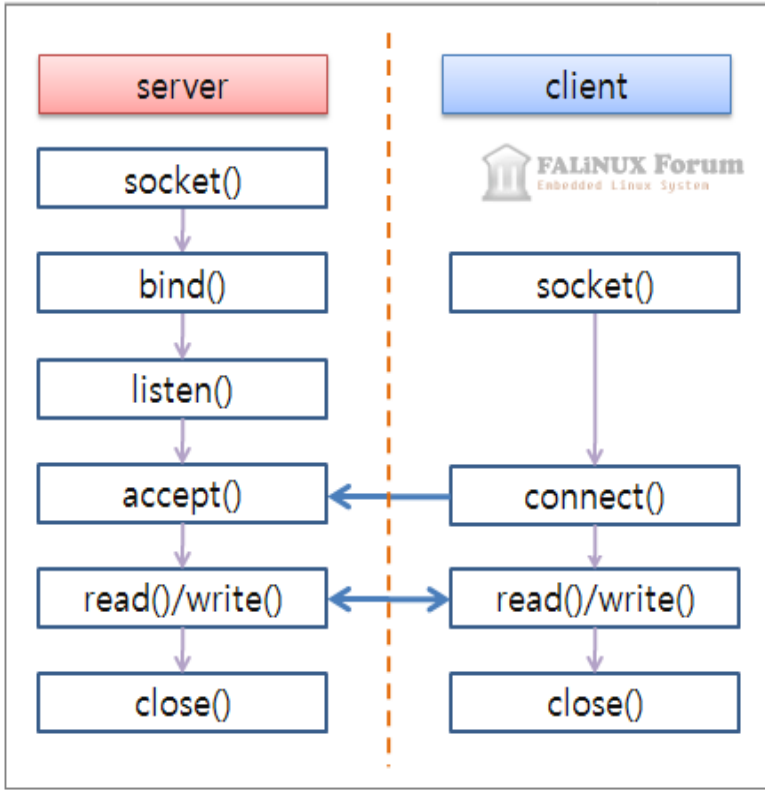
4 계층은 캡슐화 해서 데이터가 전송된다.

				User data
			Appl,header	User data
		TCP header		Application data
	IP header	TCPheader	Application	data
Ethernetheader	IP header	TCPheader	Application	data

응용
TCP
IP
Ethernet Driver

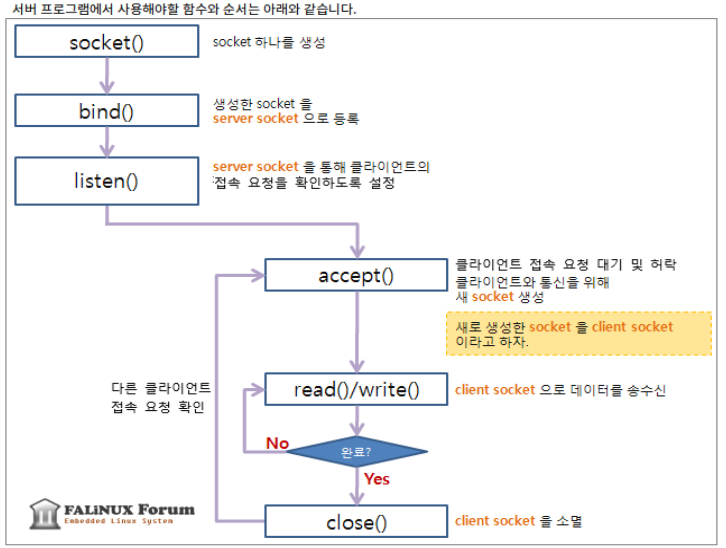
최종 물리메모리에 전송되는 데이터는 맨 밑에 형식이다.

TCP/IP 통신 함수 사용 순서



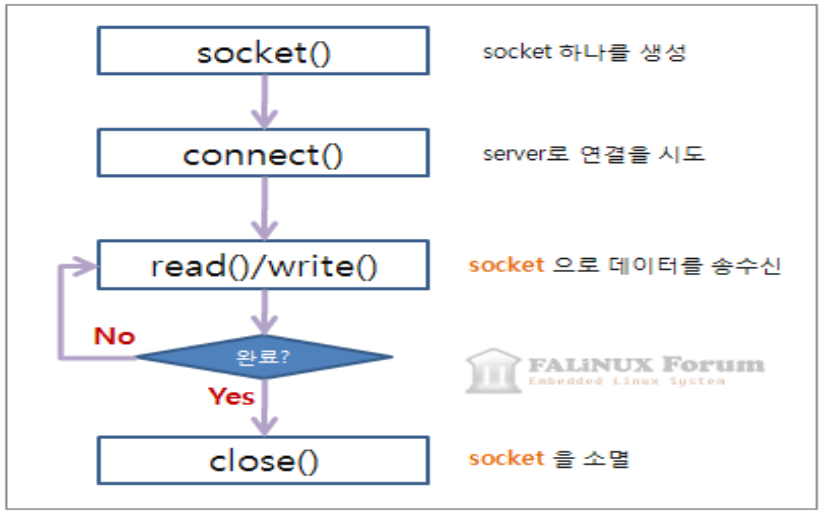
서버 프로그램

서버 프로그램에서 사용해야할 함수와 순서는 아래와 같습니다.



클라이언트 프로그램

클라이언트 프로그램은 서버에 비해 간단합니다. 바로 설명 들어갑니다.



bind() 함수를 이용하여 socket 에 server socket 에 필요한 정보를 할당하고 커널에 등록
htonl(**INADDR_ANY**) 는 주소를 지정해 주는 것으로 **inet_addr**("내 시스템의 IP ")로도 지정할 수 있다. 그러나 프로그램이 실행되는 시스템 마다 IP 가 다를 것이므로 주소 지정을 고정 IP 로 하지 않고 **htonl**(**INADDR_ANY**) 를 사용하는 것이 편리다.하
listen() 함수로 클라이언트 접속 요청을 확인
클라이언트 접속 요청에 따라 **accept**()로 접속을 허락
accept()로 접속 요청을 허락하게 되면 클라이언트와 통신을 하기 위해서 커널이 자동으로 소켓을 생성
accept()를 호출 후에 에러가 없으면 커널이 생성한 client socket 을 반환
client socket 이 생기면 read wirte 함수를 통해 데이터를 송수신 가능.

---sock(fd)---

```
#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);//tcp
	fd[1] = socket(PF_INET, SOCK_DGRAM, 0);//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;
}
//issock? ls -l 에서 sock 해서 만든거? s 셋팅하는 부분 보면 됨 . 에들어가는 것들? = 3,4,5
//socket 을 하나 open 을 하나 모두 파일이다.

//배열 0 티시피 배열 1 번 유디피 둘다 파일리스트로 받아들임 파일을 받아들이는데 중요하지 않다 소켓은 전부 파일이다. 그게 핵심
```

---convert_endian---

```
#include <stdio.h>

int main(void)
{
	unsigned short host_port = 0x5678;
	unsigned short net_port;
	unsigned long host_addr = 0x87654321;
	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: %#lx\n", host_addr);
	printf("Networ Ordered Address: %#lx\n", net_addr);

	return 0;
}
/*
Host Ordered Port : 0x5678 2 바이트
Network Ordered Port: 0x7856 Little Endian? 저장방식 메모리에 실제 배치되
```

는 방식으로 바꿔줌 크로스매칭
Host Ordered Address: 0x87654321 4 바이트 크로스 매칭
Networ Ordered Address: 0x21436587
시피유마다 엔디안이 다르다 빅앤디안에서 리틀앤디안 정보를
빅앤디안에서 리틀앤디안 정보를 받을수도 있어 서로 다른 방식에서 정보를 받을때
꼬이지 말라고 하나의 통일 방식을 만들어 줌.
*/

---inet_addr---

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

int main(int argc , char **argv)
{
	char *addr1 = "3.7.5.9";
	char *addr2 = "1.3.5.7";

	unsigned long conv_addr = inet_addr(addr1);
	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;
}
//주소체계 네트워크 형식에 맞게 변경시킨다 .
//9050703 7050301
//메모리에 들어가는 형식을 봐꿔놓고 쓴다.
```

//03070509
//01030507