# TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 문한나

mhn97@naver.com

**파일분할**

**<init_sock.h>** //헤더파일
```
#ifndef __INIT_SOCK_H__
#define __INIT_SOCK_H__

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

typedef struct sockaddr_in     si;
typedef struct sockaddr *      sp;

void err_handler(char *msg);
int init_sock(void);
void init_sock_addr(si *, int, char **, int);
void post_sock(int, si *, int);

#endif
```

**<init_sock.c>**
```
#include "init_sock.h" //헤더파일 선언

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

int init_sock(void)
{
        int sock;

        sock = socket(PF_INET, SOCK_STREAM, 0); //소캣생성

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

        return sock;
}

// serv = 0, clnt = 1
void init_sock_addr(si *serv_addr, int size, char **argv, int opt) //어드레스 구조체 설정
{
        memset(serv_addr, 0, size);
        serv_addr->sin_family = AF_INET;
```

```
	if(opt) //클라이언트
	{
		serv_addr->sin_addr.s_addr = inet_addr(argv[1]);
		serv_addr->sin_port = htons(atoi(argv[2]));
	}
	else //서버
	{
		serv_addr->sin_addr.s_addr = htonl(INADDR_ANY);
		serv_addr->sin_port = htons(atoi(argv[1]));
	}
}

void post_sock(int serv_sock, si *serv_addr, int size)
{
	if(bind(serv_sock, (sp)serv_addr, size) == -1) //ip 주소를 세팅한다
		err_handler("bind() error!");

	if(listen(serv_sock, 5) == -1) // 클라이언트의 접속 요청 확인
		err_handler("listen() error!");
}
```

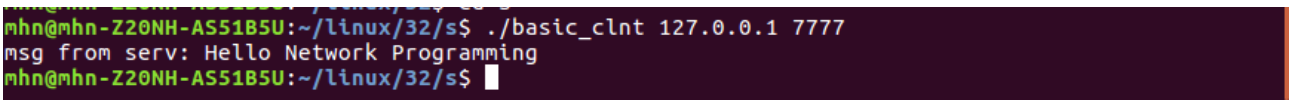
**<basic_server.c>**

```
#include "init_sock.h" //헤더파일 선언

int main(int argc, char **argv)
{
	int serv_sock, clnt_sock;
	si serv_addr, clnt_addr;
	socklen_t clnt_addr_size;
	char msg[] = "Hello Network Programming";

	if(argc != 2)
	{
		printf("use: %s <port>\n", argv[0]);
		exit(1);
	}

	serv_sock = init_sock(); //서버소캣을 생성한다
	init_sock_addr(&serv_addr, sizeof(serv_addr), argv, 0); //서버어드레스 구조체 설정
	post_sock(serv_sock, &serv_addr, sizeof(serv_addr)); //서버소켓으로 ip 를 등록하고 클라이언트의 접속 요청을 확인

	clnt_addr_size = sizeof(clnt_addr);
	clnt_sock = accept(serv_sock, (sp)&clnt_addr, &clnt_addr_size); //클라이언트 접속 요청 및 대기 허락

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

	write(clnt_sock, msg, sizeof(msg)); //클라이언트에게 메시지 전송
```

```
        close(clnt_sock);
        close(serv_sock);

        return 0;
}
```

**<basic_clnt.c>**

```
#include "init_sock.h"

int main(int argc, char **argv)
{
        int sock, len;
        si serv_addr;
        char msg[32] = {0};

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

        sock = init_sock(); //클라이언트 소캣을 생성한다.
        init_sock_addr(&serv_addr, sizeof(serv_addr), argv, 1); //클라이언트 어드레스 구조체 설정

        if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1) //서버와 연결한다.
                err_handler("connect() error!");

        len = read(sock, msg, sizeof(msg) – 1); //서버의 메시지를 읽는다

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

        printf("msg from serv: %s\n", msg); //읽은 데이터를 출력한다.
        close(sock);

        return 0;
}
```

**<web_server.c>**

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

#define BUF_SIZE 1024
#define SMALL_BUF 100

typedef struct sockaddr_in si;
typedef struct sockaddr *sp;

void error_handling(char *msg){

	fputs(msg,stderr);
	fputc('\n',stderr);
	exit(1);

}

void send_error(FILE *fp){

	char protocol[] = "HTTP/1.0 400 Bad Request\r\n";
	char server[] = "Server:Linux Web Server\r\n";
	char cnt_len[] = "Content-length:2048\r\n";
	char cnt_type[] = "Content-type:text/html\r\n\r\n";
	char content[] = "<html><head><title>Network</title></head>""<body><font
size=+5><br><br> 오류발생!  요청 파일명 및 방식 확인!""</font></body></html>";

	fputs(protocol, fp);
	fputs(server, fp);
	fputs(cnt_len, fp);
	fputs(cnt_type, fp);
	fflush(fp);

}

char *content_type(char *file){

	char extension[SMALL_BUF];
	char file_name[SMALL_BUF];
	strcpy(file_name, file);
	strtok(file_name, ".");
	strcpy(extension, strtok(NULL, "."));

	if(!strcmp(extension,"html") || !strcmp(extension,"htm"))
		return "text/html";
	else
```

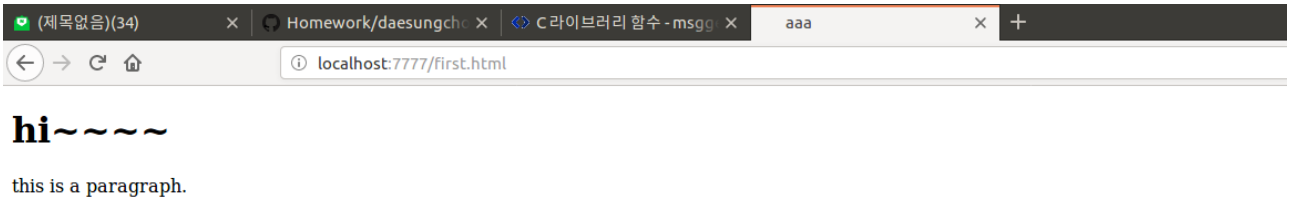
```
		return "text/plain";

}

void send_data(FILE *fp, char *ct, char *file_name){

	char protocol[] = "HTTP/1.0 200 OK\r\n";
	char server[] = "Server:Linux Web Server\r\n";
	char cnt_len[] = "Content-length:2048\r\n";
	char cnt_type[SMALL_BUF];
	char buf[BUF_SIZE];
	FILE *send_file;

	sprintf(cnt_type, "Content-type:%s\r\n\r\n",ct);
	send_file = fopen(file_name, "r");

	if(send_file == NULL){

		send_error(fp);
		return;

	}

	fputs(protocol, fp);
	fputs(server, fp);
	fputs(cnt_len, fp);
	fputs(cnt_type, fp);

	while(fgets(buf, BUF_SIZE, send_file) != NULL){

		fputs(buf, fp);
		fflush(fp);

	}

	fflush(fp);
	fclose(fp);

}

void *request_handler(void *arg){

	int clnt_sock = *((int *)arg);
	char req_line[SMALL_BUF];
	FILE *clnt_read;
	FILE *clnt_write;

	char method[10];
	char ct[15];
	char file_name[30];

	clnt_read = fdopen(clnt_sock, "r");
```

```
        clnt_write = fdopen(dup(clnt_sock), "w");
        fgets(req_line,SMALL_BUF, clnt_read);

        if(strstr(req_line, "HTTP/") == NULL){

        send_error(clnt_write);
        fclose(clnt_read);
        fclose(clnt_write);
        return NULL;

        }

        strcpy(method, strtok(req_line, " /"));
        strcpy(file_name,strtok(NULL, " /"));
        strcpy(ct, content_type(file_name));

        if(strcmp(method, "GET") != 0){
        send_error(clnt_write);
        fclose(clnt_read);
        fclose(clnt_write);
        return NULL;
}

fclose(clnt_read);
send_data(clnt_write, ct, file_name);

}

int main(int argc,char **argv){

        int serv_sock,clnt_sock;
        si serv_addr, clnt_addr;
        int clnt_addr_size;
        char buf[BUF_SIZE];
        pthread_t t_id;

        if(argc != 2){

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

        serv_sock = socket(PF_INET, SOCK_STREAM,0);
        memset(&serv_addr,0,sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons(atoi(argv[1]));

        if(bind(serv_sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
                error_handling("bind() error");
        if(listen(serv_sock,20) == -1)
                error_handling("listen() error");
```

```
	for(;;){

		clnt_addr_size = sizeof(clnt_addr);
		clnt_sock = accept(serv_sock, (sp)&clnt_addr, &clnt_addr_size);
		printf("Connection Request: %s:
%d\n",inet_ntoa(clnt_addr.sin_addr),ntohs(clnt_addr.sin_port));
		pthread_create(&t_id,NULL,request_handler,&clnt_sock);
		pthread_detach(t_id);

	}
	close(serv_sock);
	return 0;
}
```

**<first.html>**

```
<!DOCTYPE html>
<html>
<head>
<title>aaa</title>
</head>

<body>
<h1>hi~~~~</h1>
<p>this is a paragraph.</p>

</body>
</html>
```

**비유 1.회사**

얼마 전 회사를 차린 A 씨는 요즘 행복한 고민에 빠졌다. 처음 회사를 차릴 때 우려했던 바와는 달리 회사는 날로 번창하여 수입은 계속 늘어났지만, 이에 비례해서 일거리도 계속 늘어나면서 더 이상 모든 일을 혼자 처리하기엔 역부족이었기 때문이다. (→ Firmware Level 에서 모든 것을 직접 코딩하기에는 역부족이다.)
어쩔 수 없이 A 씨는 회사의 운영을 책임질 사장을 뽑기로 결정한다. (→ 운영체제를 사용하기로 결정한다.)
그래서 A 씨는 인력회사에 전화하여 컴퓨터 활용과 외국어 구사가 가능하며 성실한 사람을 보내달라고 요청하였다. 그런 뒤 A 씨는 새로 회사에 출근할 사람을 위해 책상과 의자, 그리고 사무를 보는데 필요한 기타 용품을 준비해 두었다.

다음날, A 씨의 회사로 출근한 B 씨는 A 씨로부터 어떻게 회사를 끌어가라는 지시를 받았다.
그러자 B 씨는 원래 A 씨가 했었던 수많은 귀찮은 일들을 모두 처리해 주었다. (→ Firmware Level 에서코딩해야했던 일들을 처리해 주었다.)
A 씨는 너무도 편했고, 이에 따라 자신 본연의 임무를 충실할 수 있었다. B 씨는 A 씨가 시킨 일들을 모두 완수하기 위해 부하직원을 채용했다. (→ 태스크를 생성했다.)
24 시간 근무하며 회사의 전화를 받는 C 씨와, D 씨. (→ Background Daemon, Shell Task)
C 씨와 D 씨는 평소 할 일이 없을 땐 그저 사무실에서 쉬고 있었다. 그러나 C 씨는 누가 일을 시키지 않더라고 회사로 전화가 걸려오면 언제든 그 전화를 받아 응대하는 일을 수행하였고, D 씨는 A 씨가 시키는 일이 있을때마다 그 일을 수행하였다.

A 씨의 회사가 날로 번창하자 B 씨는 더 많은 부하직원을 채용하기로 결심하였다. 우선 C 씨의 업무가 너무 과중한 것 같아 C 씨와 동일한 일을 수행하는 C2 씨와, (→ fork()로 자식프로세스를 만든다.)
제품을 조립하는 E 씨, 조립된 제품을 포장하는 F 씨, 포장된 제품을 발송하는 G 씨를 추가로 채용하였다. (→ exec()로 새로운 프로그램을 적재시킨다,)
C, C2, D, E, F 그리고 G 씨는 한정된 사무실 공간에 있었지만 E 씨가 제품을 조립해야 할 때에는 남들보다 더 많은 공간을 쓸 수 있도록 배려하였다. 또한 제품을 포장해야 할때에는 F 씨가, 제품을 발송할 때는 G 씨가 공간을 더 쓸 수 있도록 B 씨가 항상 공간 배정을 해 주었기에 (→ 운영체제가 메모리를 할당/해제한다)
모두들 사무실이 좁다는 생각을 하지 않고, 자신의 맡은바 임무를 완수할 수 있었다.

**비유 2. 카사노바 박씨 이야기**

카사노바 박씨에겐 24 명의 여자친구가 있다. 모두 내일 꼭 만나달라고 아우성이었다. 대체 내일 누굴 만날 것인가?

친구들에게 물어보았다. 하지만 친구들마다 다른 충고를 해주었다. 친구 이씨는 가장 예쁜 여자를 만나겠다고 하였고, 또 다른 친구 김씨는 가장 착한 여자를 만나겠다고 하였다.

하지만 진정한 카사노바 박씨는 어떻게 하면 더 많은 여자 친구를 만날 수 있을까 고민하던 중, (→ cpu 는 한 순간에 한가지 일 밖에 못한다.)
24 명의 여자 친구를 가장 예쁜 순서대로 줄을 서게 하고 (→ 프로세스를 경쟁시켜 우선순위로 정렬한다.)
순서대로 정해진 시간 만큼씩만 만나기로 하였다. (→ 정해진 시간만큼만 cpu 자원을 할당)
따라서 단 한명의 여자 친구만 카사노바 박씨를 만나는 상태이고, (→ 물리적인 메모리에 올라가있는 상태(프로세스) → run Queue)
다른 모든 여자 친구는 박씨를 기다리는 상태이다. (→ wait Queue)

그런데 문제가 생겼다. 얼마나 오랫동안 만날 것인가? 1 시간으로 고정하면 어떨까? 할 얘기가 많은 여자 친구와도 1 시간, 할 얘기가 별로 없는 여자 친구와도 1 시간 만난다는 것은 비효율 적이었다. 그래서 카사노바 박씨는 할 얘기가 많은 여자 친구와는 보다 더 오랜 시간 만나기로 결정하였다. (→ time slice)

또 다른 문제가 생겼다. 오늘 새로 알게 된 여자 친구가 급히 만나달라고 조르기 시작했다. (→Interrupt)
카사노바 박씨는 고민에 빠졌다. 여자 친구를 만나던 중 다른 여자 친구가 급히 만나자고 하면 응할 것인가 말 것인가. 카사노바 박씨는 급한 일이 있는 경우에만 다른 여자 친구를 만나기로 결정하였다. (→ Interrupt 발생 시 작업중이던 프로세스를 wait Queue 에 넣어두고 요청이 들어 온 일을 먼저 처리한다.)
그렇게 여러 여자 친구를 만나고 있던 카사노바 박씨는 여러 여자친구를 만나고 있는 사실을 들키지 않게 치밀한 준비를 하였다. 지난번에 만났을 때 어디까지 얘기 했는지 수첩에 꼼꼼히 기록해 두었던 것이다.
(→ Context_Swithing 은 빠른 속도로 여러 프로세스들이 제한된 시간에 서로 제어권을 넘겨주면서 cpu 를 사용하는 것이다. 프로세스마다 task 구조체를 생성 후 제어권을 넘기기 전에 정보를 저장한다.)
그래서 박씨는 여러 여자친구를 만나더라도 했던 얘기를 또 하는 등의 실수를 범하지 않을 수 있었다.
(→Multi_Tasking)

**아주 빠른속도로 프로세스들이 제어권을 넘겨주면서 cpu 를 사용한다면(→ Context_Switcing 을 한다면) 우리가 느끼지 못하는 순간에 작업이 완료된다.(→Multi_Tasking)**

**비유 3. 철수와 영희 이야기**

철수는 오늘도 영희가 만나고 싶었다. 그래서 철수는 영희네 집 앞으로 찾아가 만나달라고 영희 방 창문에 돌을 던졌다. (→ 시그널을 보냈다.)
돌을 던지면 창문은? 기본적으로는 깨진다. (→ 종료된다)
창문이 개지는 것이 싫다면, 영희는 누군가 돌 던질 때를 대비해서 항상 받을 준비를 하고 있어야 한다. (→ 핸들러를 등록 시켜야 한다.)

그런데 영희는 갑자기 궁금해졌다. 철수가 왜 돌을 던졌을까? 보고 싶으니 나오라고? 채팅 하고 싶으니 메신져에 들어오라고? 아님 화가 나서 그냥? 그래서 철수와 영희는 왜 돌을 던졌는지를 나타내는 번호를 돌에 쓰기로 결정하였다. (→ 시그널 번호. 번호로 어떤 작업을 수행할 지 알 수 있음. 이 번호는 kill -l 로 확인가능)

```
mhn@mhn-Z20NH-AS51B5U:~/linux/32$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
mhn@mhn-Z20NH-AS51B5U:~/linux/32$
```

한편, 철수가 던진 돌을 받은 영희는 만나자는 번호가 써 있음을 알게 되었다. (→ 시그널 번호)
그래서 세수도 하고 옷도 갈아입고, 만날 준비를 하고 있었다. (→ 등록한 핸들러함수 호출하여 수행)
이때 철수가 만나달라고 또 돌을 던지면 어떨까? (→ 시그널을 또 보낸다면)
준비하고있는데 또 돌을 던지니 매우 짜증이 날 것이다. 그래서 영희는 '나갈 준비하고 있으니 또 돌 던지지 마시오' 라고 창문에 서서 붙여 놓기로 하였다. (→ signal mask 를 사용해 특정 Signal 을 Block/Unblock 시킬 수 있)다
돌을 통해 간단한 번호라는 정보를 주고받던 철수와 영희는 문득 돌을 던지는 것이 빠르고 간단하긴 하지만 자세한 정보를 주고 받을 수 없다는 생각을 하게 되었다.
편지를 쓰면 어떨까? (→ Message Queue. IPC 방법 중에 하나로 자료를 다른 프로세스로 전송한다)
전화를 하는 건 어떨까? (→ Socket. 네트워크로 통신하거나 같은 시스템 내에서 프로세스끼리 통신한다)