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

<리눅스 네트워크 프로그래밍>

2018.04.05 - 31 일차

강사 - 이상훈

gcccompil3r@gmail.com

학생 - 안상재

sangjae2015@naver.com

* 소스코드 분석

1. 소프트웨어 파일 분할하기
- 프로젝트의 크기가 커지면 각각의 코드들을 함수로 만들어야 하고, 프로젝트의 크기가 더욱 더 커지면 함수들을 기능 별로 나누어서 c 파일에 라이브러리처럼 정리해야 한다. 각각의 c 파일마다 해당 c 파일에 존재하는 함수와 변수를 선언하는 헤더파일이 존재함.
- basic_serv.c 와 basic_clnt.c 에 존재하는 코드를 함수로 바꾸어서 init_sock.c 파일에 라이브러리 형태로 저장함.
- init_sock.c 파일에 있는 함수들과 변수는 init_sock.h 에 저장되어 있음.
- basic_serv.c 와 basic_clnt.c 의 코드는 상당히 간결해짐.

<서버 프로세스> 파일명 : basic_serv.c

```
#include "init_sock.h"

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();
	init_sock_addr(&serv_addr, sizeof(serv_addr), argv, 0);
	post_sock(serv_sock, &serv_addr, sizeof(serv_addr));

	clnt_addr_size = sizeof(clnt_addr);
	clnt_sock = accept(serv_sock, (sp)&clnt_addr, &clnt_addr_size);

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

	write(clnt_sock, msg, sizeof(msg));

	close(clnt_sock);
	close(serv_sock);

	return 0;
}
```

//

<클라이언트 프로세스> 파일명 : 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();
	init_sock_addr(&serv_addr, sizeof(serv_addr), argv, 1);

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

	len = read(sock, msg, sizeof(msg) - 1);

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

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

	return 0;
}
```

//

<소켓 초기화 설정 라이브러리> 파일명 : init_sock.c

```
#include "init_sock.h"

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);

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

	return sock;
}

void init_sock_addr(si *serv_addr, int size, char **argv, int opt)
{
	memset(serv_addr, 0, size);
	serv_addr->sin_family = AF_INET;

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

```
                err_handler("bind() error!");

        if(listen(serv_sock, 5) == -1)
                err_handler("listen() error!");
}
```

//

<헤더 파일>

```
#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
```

2. 웹 서버 만들기 (html)

[web_serv.c 파일]

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

typedef struct sockaddr_in si;
typedef struct sockaddr *     sp;

#define BUF_SIZE    10000
#define MAX_CLNT    10000
#define NAME_SIZE   32
#define SMALL_BUF 128

void error_handler(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> 오류 발생! 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 0;
	}

	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 0;
	}

	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_handler("bind() error");
	if(listen(serv_sock, 20) == -1)
		error_handler("listen() error");

	for(;;){
		clnt_addr_size = sizeof(clnt_addr);
		clnt_sock = accept(serv_sock, (sp)&clnt_addr, &clnt_addr_size);
		printf("Connection Reques: %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;
}
```

//

[html 파일]

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

2-1. 결과 분석

- web_serv.c 파일을 컴파일하고 웹 브라우저에서 http://localhost:7777/first.html 를 입력하면 아래와 같은 메세지가 출력됨.

* 운영체제 동작 비유 분석

<비유 1 : 회사>

얼마 전 회사를 차린 A 씨는 요즘 행복한 고민에 빠졌다. 처음 회사를 차릴 때 우려 했던 바와는 달리 회사는 날로 번창하여 수입은 계속 늘어났지만, 이에 비례해서 일거리도 계속 늘어나면서 더 이상 모든 일을 혼자 처리하기엔 역부족이었기 때문이었다.

어쩔 수 없이 A 씨는 회사의 운영을 책임질 사장을 뽑기로 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. A 씨는 너무도 편했고, 이에 따라 자신 본연의 임무에 충실할 수 있었다.

B 씨는 A 씨가 시킨 일들을 모두 완수하기 위해 부하직원을 채용했다. 24 시간 근무하며 회사의 전화를 받는 C 씨와 D 씨. C 씨와 D 씨는 평소 할 일이 없을 땐 그저 사무실에서 쉬고 있었다. 그러나 C 씨는 누가 일을 시키지 않더라도 회사로 전화가 걸려오면 언제든 그 전화를 받아 응대하는 일을 수행하였고, D 씨는 A 씨가 시키는 일이 있을 때마다 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 제품을 조립하는 E 씨 조립된 제품을 포장하는 F 씨, 포장된 제품을 발송하는 G 씨를 추가로 채용하였다.

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 포장 해야 할 때에는 F 씨가, 제품을 발송할 때는 G 씨가 공간을 더 쓸 수 있도록 B 씨가 항상 공간 배정을 해주었기에 모두들 사무실이 좁다는 생각을 하지 않고 자신의 맡은바 임무를 완수할 수 있었다.

<운영체제 동작>

Firmware 프로그래머 A 씨는 여러 가지 기능을 하는 프로그램을 만들고 있다. 프로그램의 흐름을 계획하며 모든 것을 Firmware Level 에서 코딩하다 보니 한계가 생겼다. 이런 한계를 극복하기 위해 리눅스라는 운영체제를 사용하기로 했다. 메모리 양은 한정이 되어 있지만 운영체제는 여러 프로세스들이 실행될 때, 필요한 메모리 크기만큼 할당을 하고 해제를 해줌으로써 메모리를 관리한다. 데이터량이 많은 프로세스에게는 더 많은 메모리를 할당해주고, 데이터량이 적은 프로세스에게는 적은 메모리를 할당해준다. 그러므로 프로세스들은 메모리의 부족함을 전혀 느끼지 못한다.

또한 프로세스들마다 역할을 부여해서 할일이 없는 프로세스는 idle 상태를 유지하며 쉬게 되고, 데몬 프로세스느 누가 일을 시키지 않더라도 해야 할 일이 생기면 Running 상태가 되어 자신에게 주어진 일을 수행하였고, 쉘은 A 씨가 시키는 일이 있을 때마다 Running 상태가 되어 시킨 일을 수행하였다.

해야할 일이 점점 더 많아져서 기존에 존재하는 프로세스들로 충당할 수 없어지자 운영체제는 fork() 시스템 콜을 호출해서 프로세스를 원하는 만큼 더 만들고 각각의 프로세스들에게 역할을 부여한다.

<비유 2 : 카사노바 박씨 이야기>

카사노바 박씨에겐 24 명의 여자 친구가 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, 24 명의 여자 친구를 가장 예쁜 순서대로 정해진 시간 만큼씩만 만나기로 하였다. 따라서 단 한명의 여자 친구만 카사노바 박씨를 만나는 상태이고, 다른 모든 여자 친구는 박씨를 기다리는 상태이다.

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 비효율적 이었다. 그래서 카사노바 박씨는 할 얘기가 많은 여자 친구와는 보다 더 오랜 시간 만나기로 결정하였다.

또 다른 문제가 생겼다. 오늘 새로 알게 된 여자 친구가 급히 만나달라고 조르기 시작했다. 카사노바 박씨는 고민에 빠졌다. 여자 친구를 만나던 중 다른 여자 친구가 급히 만나자고 하면 응할 것인가 말것인가. 카사노바 박씨는 급한 일이 있는 경우에만 다른 여자 친구를 만나기로 결정하였다.

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. 그래서 박씨는 여러 여자친구를 만나더라도 했던 얘기를 또 하는 등의 실수를 범하지 않을 수 있었다.

<운영체제 동작>

운영체제의 종류중에 하나인 리눅스는 여러 개의 태스크를 구동시킨다. CPU 는 한번에 하나의 태스크만 처리할 수 있기 때문에 리눅스는 여러 개의 태스크들을 어떤 순서로 실행시킬지에 대한 계획을 가지고 있다. 우선 중요도에 따라 태스크마다 우선순위를 배정해 우선순위가 높은 순으로 차례로 실행시킨다. 그러나 우선순위가 낮은 태스크가 갑자기 급한 일이 생겨서 당장 실행이 되어야 하는 경우에는 리눅스가 선점기능을 제공해서 당장 실행시켜 준다. 또한 각각의 태스크들을 수행할 작업이 많은 순서로 수행시간을 배정한다.

리눅스는 하나의 태스크를 실행시키다가 다른 태스크로 바꾸어서 실행시킬 때, 실행 중이었던 태스크의 상태(레지스터 상태, 프로그램 카운터, 스택 포인터) 를 프로세스 제어 블록(PCB)에 저장해 둔다. 그러다가 나중에 다시 해당 태스크를 실행시킬 때 프로세스 제어 블록에서 저장되었던 태스크의 상태정보를 불러와서 다시 실행시킨다.

<비유 3 : 철수와 영희 이야기>

철수는 오늘도 영희가 만나고 싶었다. 그래서 철수는 영희네 집 앞으로 찾아가 만나달라고 영희 방 창문에 돌을 던졌다. 돌을 던지면 창문은? 기본적으로는 깨진다. 창문이 깨지는 것이 싫다면, 영희는 누군가 돌 던질 때를 대비해서 항상 받을 준비를 하고 있어야 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. 한편, 철수가 던진 돌을 받은 영희는 만나자는 번호가 써 있음을 알게 되었다. 그래서 세수도 하고 옷도 갈아입고, 만날 준비를 하고 있었다. 이때 철수가 만나달라고 또 돌을 던지면 어떨까? 준비하고 있는데 또 돌을 던지니 매우 짜증이 날 것이다. 그래서 영희는 '나갈 준비하고 있으니 또 돌을 던지지 마시오'라고 창문에 써서 붙여 놓기로 하였다.

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? 전화를 하는 건 어떨까?

<운영체제 동작>

프로세스는 프로그램의 main 문에 해당한다. 그러므로 프로세스들은 메모리를 공유하지 않고 독립적으로 수행이 된다. 기본적으로 메모리를 공유하지 않기 때문에 프로세스들끼리 통신이 불가능하다. 프로세스들 끼리 통신할 수 있는 것이 시그널이다. 시그널 마다 각각의 번호가 있고, 각각의 시그널을 받았을 때, 어떤 행동을 취해야 하는지 시그널 핸들러라는 함수를 통해 미리 정해 놓는다. 이것을 비동기식 이라고 한다.

프로세스가 다른 프로세스로부터 시그널을 받으면 사전에 등록되어 있는 시그널 핸들러로 이동해서 미리 약속된 코드를 실행한다.

시그널은 시스템 콜이기 때문에 간단한 코드만 수행하고, 많은 행동을 취하기에는 무리가 있다. 많은 정보를 주고 받

기 위해서는 소켓을 생성하고 통신을 하는 방법이 존재한다.

<비유 4 : 통신병 김군>

오늘은 새벽 6 시부터 훈련이 시작되었다. 다른 소대/중대/대대와 연락을 주고받을 필요가 없는 특수한 훈련이 아닌 이상 통신병 김군은 훈련의 시작부터 끝까지 반드시 지휘관 옆에 상주해야 한다.

언제 다른 부대에서 연락이 올지 몰랐기 때문에 통신병 김군은 항상 무전기를 지니고 다녔다. 그러다 연락이 오면 그 내용을 지휘관에게 보고하는 것이 바로 통신병의 임무였기 때문에…

그런데 갑자기 김군의 상관이 지시했다. "야~! 옆 중대랑 연락 좀 해봐라~." 그래서 김군은 무전기를 들고 연락을 시도했다. "여기는 1 중대, 2 중대 나오라 오버" 그러자 바로 응답이 왔다. "여기는 2 중대. 1 중대 말하라 오버" . 그러자 김군은 말했다. "잠시만 기다리시지 말입니다~~오버." 그리곤 상관에게 무전기를 넘겼다.

한참 뒤 하고 싶은 대화를 끝낸 김군의 상관은 무전기를 김군에게 돌려주었다. 김군은 "이상 무전끝~" 이라고 말했고, 상대방은 "무전 끝, 확인~" 이라고 대답했다. 그래서 김군은 "그럼 수고하시지 말입니다~"라고 익살스럽게 무전을 보냈다.

<운영체제 동작>

통신을 허용해 주거나 여러 프로세스들의 요청을 받아주는 프로세스를 서버라고 한다. 반대로 서버에 접속해서 통신에 참여하는 프로세스를 클라이언트라고 한다. 서버는 항상 클라이언트의 접속에 대기하고 있어야 하고 클라이언트가 접속하면 허용해 주어서 통신이 가능해 진다.

클라이언트에서 서버로 데이터를 전송하면 서버에서 잘 받았다는 신호로 다시 클라이언트에게 데이터를 전송해 준다. 클라이언트가 데이터를 잘 수신하면 통신이 잘 되고 있다는 의미이다. 통신이 끝나고 더 이상 할 필요가 없게 되면 클라이언트가 소켓을 close 시키고, 서버도 클라이언트의 소켓을 close 한다.

<비유 5 : 크게 성공할 미래의 주방장 이군>

식사 시간이 되어 주방장은 주방 보조 이군을 불렀다. "야~! 어서 창고에 가서 재료 가져와" 라고 이군에게 지시했다.

이군은 급히 창고로 이동하여 식량을 찾기 시작했다. 사실 이군은 전혀 당황하지 않았다. 각종 물건들을 창고 안에 여기 저기 대충 쌓아 놓으면 나중에 찾기가 불편할 것이라고 판단한 이군은 평소 물건을 창고에 넣어둘 때 기가 막히게 정리를 해두었던 것이다. 우선 창고의 문 앞에는 창고 안에 어떤 물건들이 들어있고, 어떤 방식으로 정리되어 있으며, 또한 창고 안 어디에 빈 공간이 있는지를 나타내는 장부를 붙여 두었고, 창고 내부는 저장한 물건 종류별로 정리가 되어 있었으며, 각 종류별로 어떤 물건이 얼마나 있는지를 나타내는 정리표를 완벽하게 만들어서 붙여 놓았다.

<운영체제 동작>

하드디스크에는 수많은 파일이 저장되어 있다. 이 수많은 파일 중에 사용자가 원하는 파일을 빠르고 쉽게 찾기 위해서는 일정한 규칙에 따라 정리가 되어 있어야 한다. 이 규칙을 파일시스템이라고 하고 파일시스템에는 FAT. LFS, Ext2 등 여러가지가 존재한다. 또한 inode 라는 구조체가 존재해서 디스크에 해당 파일이 어디에 있는지를 알려준다.