TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
31 일차 (2018. 04. 05)

기능 분리를 하는 이유? 함수 분리, 파일 분할을 하는 이유는 재활용성이다.

"init_sock.h" 헤더 파일

```
#ifndef __INIT_SOCK_H__        //선언
#define __INIT_SOCK_H__        //중복 방지

#include <stdio.h>      //헤더 파일들. 이렇게 선언해놓으면 코드 쓸 때, 이 헤더파일 하나만 선언하면 된다.
#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);    // 에러 핸들러 함수
int init_sock(void);            //소켓 생성
void init_sock_addr(si *, int, char **, int);       //소켓 주소 지정 함수
void post_sock(int, si *, int);                     //bind, listen 설정 함수

#endif
```

"common.h"헤더 파일

```
#ifndef __COMMON_H__
#define __COMMON_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;

typedef struct __d{                     //구조체 선언
        int data;
        float fdata;    } d;

#define BUF_SIZE                        32
#endif
```

init_sock.c

```
#include "init_sock.h"

void err_handler(char *msg)     //에러 핸들러, 에러 출력. 서버/클라이언트 공통
{       fputs(msg, stderr);
        fputc('₩n', stderr);
        exit(1);                }

int init_sock(void) //소켓 에러 확인. 서버/클라이언트 공통
{       int sock;
        sock = socket(PF_INET, SOCK_STREAM, 0);      //소켓 생성
        if(sock == -1)                               //소켓 에러
                err_handler("socket() error!");      //에러 핸들러 구동
        return sock;                            }    //소켓  반환

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

        if(opt)   //1 이면
        {  serv_addr->sin_addr.s_addr = inet_addr(argv[1]);
           serv_addr->sin_port = htons(atoi(argv[2]));
        }
        Else      //0 일 때
        {  serv_addr->sin_addr.s_addr = htonl(INADDR_ANY); // INADDR_ANY 서버 본인이니까
           serv_addr->sin_port = htons(atoi(argv[1])); //port 번호. 어떤 번호를 설정할 것인지
        }
}

void post_sock(int serv_sock, si *serv_addr, int size)// 클라이언트 bind, listen 안 함. 서버만 있어서 분리함
{       if(bind(serv_sock, (sp)serv_addr, size) == -1) //bind
                err_handler("bind() error!");

        if(listen(serv_sock, 5) == -1)              //listen
                err_handler("listen() error!");
} //만약 에러가 났다면 에러 난 시점에서 더 이상 작동 안함.
```

basic_serv.c

```
#include "common.h"
#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(); // 소켓 설정
           init_sock_addr(&serv_addr, sizeof(serv_addr), argv, 0); //소켓 주소 설정
           post_sock(serv_sock, &serv_addr, sizeof(serv_addr));     //bind, listen 설정

           clnt_addr_size = sizeof(clnt_addr);        //클라이언트 주소 값
           clnt_sock = accept(serv_sock, (sp)&clnt_addr, &clnt_addr_size);//클라이언트 수용

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

           write(clnt_sock, msg, sizeof(msg));        // 받은 메시지 쓰기

           close(clnt_sock);
           close(serv_sock);

           return 0;                                 }
```

basic_clnt.c

```
#include "common.h"
#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]);     //3 개 아니면 출력
              exit(1);                                                  }

           sock = init_sock();        //소켓 생성
           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); //소켓에서 받은 거 읽어서 len 에 저장
        if(len == -1)
                err_handler("read() error!");

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

        return 0;                       }
```

웹서버

```
#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_handler(char *msg) //에러 메시지 출력
{  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);
```

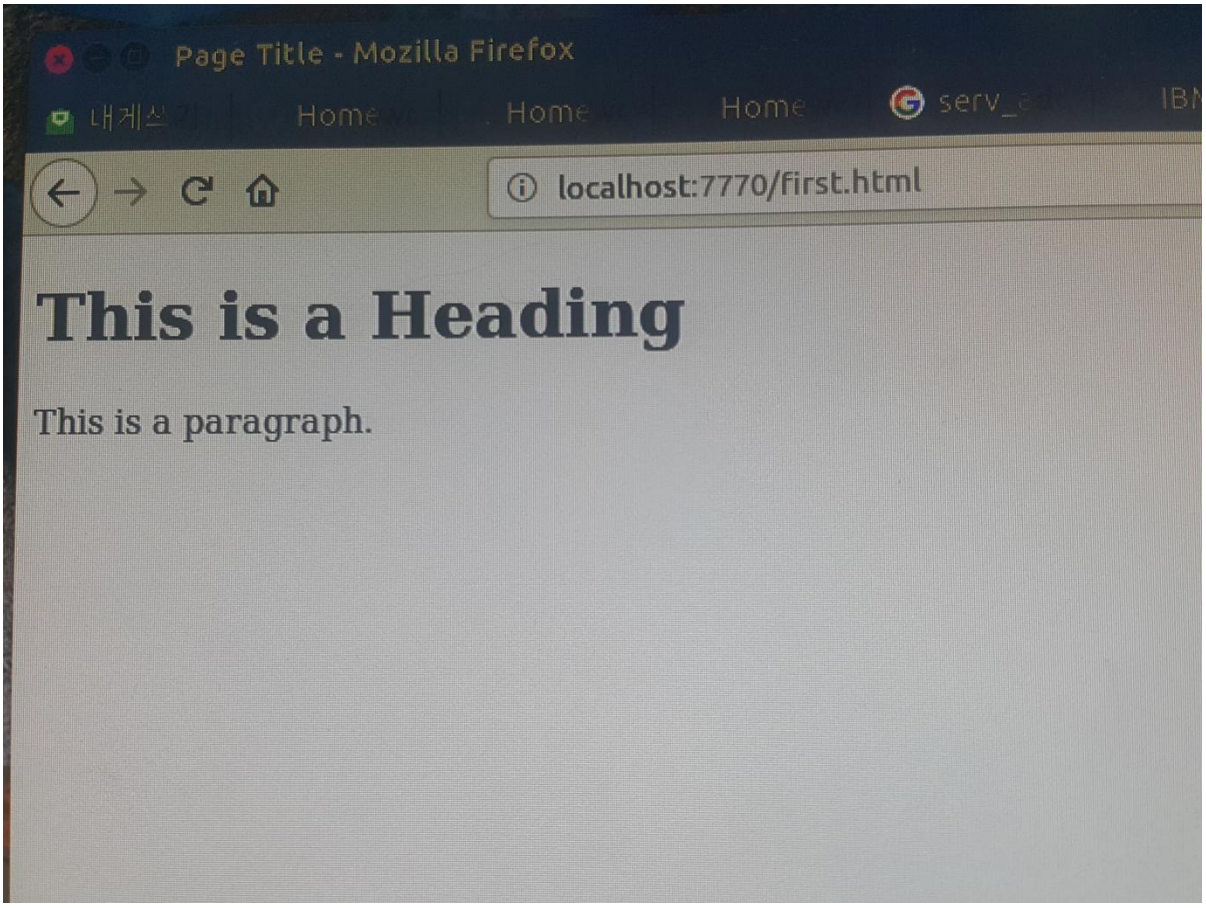
```
    }

    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;
    }

    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;
    }

    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)          //20 명
      error_handler("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>Page Title</title>
</head>
<body>
```

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

</body>
</html>
```



gcc -o web_serv web_serv.c –lpthread 로 컴파일해서 포트 번호와 함께 실행시킨다. 그 후 인터넷 창을 켜서 localhost:7777/first.html 을 입력하면 위의 first.html 이 나온다. 이 때, localhost 부분에 다른 이의 주소값을 넣으면 그 값은 변겨경이 된다.

운영체제 동작 비유
비유 1 : 회사
운영체제 동작에서 fork()로 비유 설명을 했다. A 씨를 fork()해서 만든 자식 프로세스가 B 씨이다. B 씨가 A 씨와 다르게 어떠한 일을 공통적으로 참조하는 C, D 를 고용했다. 이는 쓰레드로 볼 수 있다. 여기서 더 추가해서 생각하자면, 이를 전체적으로 볼 때는 프로세스 계통도도 관련이 있다고 본다. 프로세스 계통도에서의 관계도 fork()를 사용할 수도 있고 아닐 수도 있었으며, 내용이 같을 수도 있고 아닐 수도 있었다. 메모리 자체를 공유하는 thread 때문에 형제 프로세스도 관리하는데, 이는 전반적인 회사 정보를 같이 공유하는 것과 같은 것으로 설명할 수 잇다. 회사 직급 체계도를 프로세스 계통도와 연관 지어서 생각하면, cpth 과 ppth 은 선임과 후임의 관계 혹은 상사와 부하직원과 비슷한 관계라고 볼 수 있다. 선임(상사)은 후임의 일에 대한 결과 보고는 받지만, 모든 후임에게 바라보지는 않는다. 새로운 후임이 일을 적응할 때까지 그 곳에 관심이 집중 되는 것과 같이 ppth 가 새로운 자식 프로세스가 생성되면 옮겨가는 것과 같은 실행 동작이라 생각된다. yptr 과 optr 은 직급이 같은 선임과 후임 관계로 설명할 수 있다.

또한 사무실 공간 배정은 wait queue 와 run queue 으로 context switching 으로 모두 같은 공간에서 일은 하지만 우선 순위가 정해져 일을 처리하는 순위가 정해지고 그 일처리가 끝나면 다른 사람이 이용하여 Multi-Tasking 이 가능하도록 한다.

비유 2 : 카사노바 박씨 이야기

Multi-Tasking 을 하기 위한 context switching 이다. 여자 친구를 각각 프로세스라고 두고, 카사노바 박씨를 메모리인 경우이다. task 가 그냥 가만히 있으면 프로세스가 아니고 메모리에 올라가야 프로세스가 된다. 이는 실행파일이 실행 중에만 프로세스고 실행이 끝나면 파일이 되는 것과 같은 관계이다. 상대방과 연인 관계일 때가 실행파일에 동작하는 순간이고 메모리에서 내려가면 그냥 파일이 되듯이 여자 친구에서 그냥 이성이 되는 것과 같은 동작이다.

Context switching 을 하기 위한 우선순위가 정해지고, 우선 순위마다 주어진 할당시간이 다르다. 이는 박씨가 여성을 만날 때, 우선 순위를 정하고 그에 따른 시간을 정한 것과 같은 동작이다. cpu 는 오로지 한 순간에 한 가지 연산만 수행하듯이 박씨도 한 순간에는 한 여자친구만 만날 수 있다.

새로운 여자친구가 급히 만나달라고 한 것은 시그널의 비동기 처리와 같다고 할 수 있다.

비유 3 : 철수와 영희의 이야기

그 동안 배운 시그널과 관계가 있다. 이는 시그널을 받고 나서 그 번호로 어떠한 시그널을 날린 건지 알아볼 수 있는 것과 동일하게 돌에 번호를 기입하여 서로 알아볼 수 있도록 했기 때문이다. 하지만 그에 뜻하는 배열이나 구조체처럼 많은 데이터를 가지고 있지 않으므로, 정확한 값을 알 수 없다. 그래서 다양하고 넓은 정보전달을 위해 다른 방식을 찾아보자는 말이 나온 것이다.

비유 4 : 통신병 김군

그 동안 했던 네트워크 프로그래밍과 같다. 통신병이 지휘관 옆에서 상주하고 있는 것은 서버는 listen 으로 클라이언트로부터 값을 받는 것을 기다리고 있기 때문이다. 어디서 언제 불특정 다수가 들어오지 모르기에 기다리는 것이다. 지휘관의 지시로 연락해보라고 하는 것은 클라이언트에게 서버가 정보를 전달하는 것이다. 그리고 '~오버'는 null 문자와 같이 현재 문장이 끝난 것을 알려준다. '무전끝'은 클라이언트와 서버 소켓을 닫아서 메모리 누수 되는 것을 막는 것과 같은 이치이다.

비유 5: 크게 성공할 미래의 주방장 이군

운영체제는 디스크의 일부 공간을 할당 받는다. 그 공간에 파일의 내용을 저장한다. 이는 식재료가 공간을 할당 받아 그 곳에 저장 된 것과 같은 이치이다. 이 때, 막무가내로 저장이 될 경우 어디 있는지 찾기가 힘든데, inode 라는 것이 있는데, 이는 파일을 넣었다 뺐을 때, 디스크에서 읽을 때, 몇 번째로 어디에 있는지에 대한 정보가 있다. 이처럼 이군은 장부를 두어 용이하게 관리하고 찾을 수 있게 만들어 놓았다.