TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 - 이상훈
gcccompil3r@gmail.com

학생 – 이우석
colre99@naver.com

[4/5(목) – 31 일차]

[chat 복습편] – 서버에서 클라이언트의 입력을 출력한다.
추후, 서버에서도 입력을 주어 클라이언트쪽에서 출력할 수 있게할 예정.

[chat_serv.c]

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

#define BUF_SIZE     128
#define MAX_CLNT     256



typedef struct sockaddr_in  si;
typedef struct sockaddr * sp;

int clnt_cnt = 0;
int clnt_socks[MAX_CLNT];
```

[chat_clnt.c]

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

#define BUF_SIZE  128
#define NAME_SIZE  32

typedef struct sockaddr_in si;
typedef struct sockaddr * sp;

char name[NAME_SIZE] = "[DEFAULT]";
char msg[BUF_SIZE];
```

```
pthread_mutex_t mtx;
```

→ 뮤텍스의 특징을 결정하기 위해 사용한다.
→ (fast mutex, recursive mutex, mutex 에러체크용)
=> 3 가지 상수

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

void send_msg(char *msg, int len)
{
        int i;

        pthread_mutex_lock(&mtx);
```

→ lock 은 임계영역 진입하기 위한 요청. 뮤텍스 잠금요청

```
        for(i = 0; i < clnt_cnt; i++)
                write(clnt_socks[i], msg, len);
                printf("%s", msg);
        pthread_mutex_unlock(&mtx);
}
```

→ 뮤텍스 잠금을 되돌려준다. unlock 은 임계영역을 빠져나오면서 다른 쓰레드에게 임계영역을 되돌려

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

void *send_msg(void *arg)
{
        int sock = *((int *)arg);
        char name_msg[NAME_SIZE + BUF_SIZE];

        for(;;)
        {
                fgets(msg, BUF_SIZE, stdin);
          //strcmp : string1 및 string2 를 비교합니다.
                if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
                {
                        close(sock);
                        exit(0);
                }
```

→ 구성: 버퍼에 이름, 이름%s,

```
                sprintf(name_msg, "%s %s", name, msg);
```

→ spritntf : 버퍼에 포맷을 지정하여 출력하는 함수

```
                write(sock, name_msg, strlen(name_msg));/
```

주기 위해서 사용한다.

```
void *clnt_handler(void *arg)
{
        int clnt_sock = *((int *)arg);
        int str_len = 0, i;
        char msg[BUF_SIZE];

        while((str_len = read(clnt_sock, msg, sizeof(msg))) !
= 0)
                send_msg(msg, str_len);

                 printf("%s", msg);
```

→ 클라이언트쪽에서 입력하는 문자를 출력함.

```
        pthread_mutex_lock(&mtx);

        for(i = 0; i < clnt_cnt; i++)
        {
                if(clnt_sock == clnt_socks[i])
                {
                        while(i++ < clnt_cnt -1)
                                clnt_socks[i] = clnt_socks[i + 1];
                        break;
                }
        }
```

→ 반환값: 버퍼에 출력한 문자 개수

```
        }
```

→ strlen: string length 로 문자열의 길이를 반환해주는 함수

```
        return NULL;
}

void *recv_msg(void *arg)
{
        int sock = *((int *)arg);
        char name_msg[NAME_SIZE + BUF_SIZE];
        int str_len;

        for(;;)
       {
        str_len = read(sock, name_msg, NAME_SIZE +
       BUF_SIZE - 1);
```

→ 문자열 길이를 구하는 함수

```
                if(str_len == -1)
                        return (void *)-1;

                name_msg[str_len] = 0;
                fputs(name_msg, stdout);
        }
```

→ fputs 는 str 이 가리키는 문자열을 NULL 문자('\0')에 도달할때 까지 스트림에 복사한다.
마지막 NULL 문자는 스트림에 복사되지 않는다.

```
        clnt_cnt--;
        pthread_mutex_unlock(&mtx);
        close(clnt_sock);

        return NULL;
}

int main(int argc, char **argv)
{
        int serv_sock, clnt_sock;
        si serv_addr, clnt_addr;
        socklen_t addr_size;
        pthread_t t_id;

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

        pthread_mutex_init(&mtx, NULL);

        serv_sock = socket(PF_INET, SOCK_STREAM, 0);

        if(serv_sock == -1)
                err_handler("socket() error");
```

```
return NULL;
}

int main(int argc, char **argv)
{
        int sock;
        si serv_addr;
        pthread_t snd_thread, rcv_thread;
        void *thread_ret;

        if(argc != 4)
        {
        printf("Usage: %s <IP> <port> <name>\n", argv[0]);
        exit(1);
        }

        sprintf(name, "[%s]", argv[3]);
        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, (sp)&serv_addr, sizeof(serv_addr))
```

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

        if(listen(serv_sock, 10) == -1)
                err_handler("listen() error!");

        for(;;)
        {
                addr_size = sizeof(clnt_addr);
                clnt_sock = accept(serv_sock, (sp)&clnt_addr,
&addr_size);

                pthread_mutex_lock(&mtx);
                clnt_socks[clnt_cnt++] = clnt_sock;
                pthread_mutex_unlock(&mtx);

                pthread_create(&t_id, NULL, clnt_handler,
(void *)&clnt_sock);
                pthread_detach(t_id);
                printf("Connected Client IP: %s\n",
```

```
        == -1)
                err_handler("connect() error");

        pthread_create(&snd_thread, NULL, send_msg,
        (void *)&sock);

        pthread_create(&rcv_thread, NULL, recv_msg, (void
        *)&sock);
        pthread_join(snd_thread, &thread_ret);
        pthread_join(rcv_thread, &thread_ret);

        close(sock);
        return 0;
}       /*
```

pthread_create:pthread 생성.

(첫번째인자: pthread 의 식별자로 thread 가 성공적으로 생성되면 식별값이 주어진다.

두번째인자: pthread 속성(옵션), 기본적인 thread 속성을 사용할 경우 NULL.

세번째인자: pthread 로 분기할 함수. 반환값이 void* 타입이고 매개변수도 void*으로 선언된 함수만 가능.

ex) void* handler(void*arg){...})
pthread_join: pthread 가 종료될때까지 기다리다가

```
inet_ntoa(clnt_addr.sin_addr));

        }

        close(serv_sock);

        return 0;
}
```

특정 pthread 가 종료시 자원 해제시켜준다.

(첫번째인자: 어떤 pthread 를 기다릴지 정하는 식별자.

두번째인자: pthread 의 return 값. 포인트로 값을 받아오는 점을 주의.)

pthred_detach: th_id 식별자를 가지는 pthread 가 부모 pthread 로 부터 독립한다.

즉, 이렇게 독립된 pthread 는 따로 pthread_join()이 없이도 종료시 자동으로 리소스 해제된다.

pthread_t : pthread 의 자료형을 의미

sys/epoll.h : 한대의 서버에서 아주많은 동시접속자를 처하기 위한 수단.

*/

[오늘 수업한 내용]

[web_serv.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)
```

[first.html]

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

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

</body>
</html>
```

```
{
	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_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;

}
```