

# **TI DSP, MCU 및 Xilinx Zynq FPGA**

## **프로그래밍 전문가 과정**

**강사 - Innova Lee(이상훈)**  
**[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)**  
**학생 - 하성용**  
**[accept0108@naver.com](mailto:accept0108@naver.com)**

## gcInt.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

typedef struct sockaddr_in  si;
typedef struct sockaddr *   sp;

char msg[BUF_SIZE];

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

void *send_msg(void *arg)
{
    int sock = *((int *)arg); //만들어진 소켓의 정보가 전달됨
    char msg[BUF_SIZE];

    for(;;) //무한루프돌면서 사용자입력을 서버로 전송해주는 쓰레드
    {
        fgets(msg, BUF_SIZE, stdin);
        write(sock, msg, strlen(msg)); //입력한 메시지값이 서버로 전송
    }

    return NULL;
}

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

    for(;;) //무한 루프 돌면서 서버로부터 들어오는 정보를 msg 로 보내줌
    {
        str_len = read(sock, msg, BUF_SIZE - 1);

        msg[str_len] = 0;
        fputs(msg, stdout); //메세지 출력
    }

    return NULL;
}

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

```

```

    si serv_addr;
    pthread_t snd_thread, rcv_thread;
    void *thread_ret;

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

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

    //(void *)&sock -> 함수에 보내는 쓰레드의 인자가 됨
    //쓰레드만드는이유는 송신과 수신을 분리하기위해
    //샌드에 보내는쓰레드를 만듦,(즉 보내는쓰레드만든거),인자는 sock 이라는걸 사용

    pthread_create(&rcv_thread, NULL, rcv_msg, (void *)&sock);

    //rcv_쓰레드→ 수신하는쓰레드, 크리에이트 하는순간 쓰레드를 만듦

    pthread_join(snd_thread, &thread_ret);
    //구동은 join 할때

    pthread_join(rcv_thread, &thread_ret);

    close(sock);
    //끝나는건 송, 수신할게 없게될때 ex)ctrl+c 할때

    return 0;
}

```

## gserv.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/socket.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];
int data[MAX_CLNT];

```

```

int thread_pid[MAX_CLNT];
int idx;
int cnt[MAX_CLNT];
pthread_mutex_t mtx;
//mutex 타입의 t mtx -> 락의 키값

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

void sig_handler(int signo)
{
    int i;

    printf("Time Over!\n");

    pthread_mutex_lock(&mtx);

    for(i = 0; i < clnt_cnt; i++)
        if(thread_pid[i] == getpid())
            cnt[i] += 1;

    pthread_mutex_unlock(&mtx);

    alarm(3);
}

void proc_msg(char *msg, int len, int k)
{
    int i;
    int cmp = atoi(msg);
    char smsg[64] = {0};

    pthread_mutex_lock(&mtx);

    cnt[k] += 1;
    //몇번 입력했는지 카운트

    if(data[k] > cmp)
        //데이터가 큰지

        sprintf(smsg, "greater than %d\n", cmp);
        //입력한값을 cmp 로 바꿈

    else if(data[k] < cmp)
        //데이터가 작은지
        sprintf(smsg, "less than %d\n", cmp);
    else
    {
        strcpy(smsg, "You win!\n");
        printf("cnt = %d\n", cnt[k]); //write 값을 출력
    }

    strcat(smsg, "Input Number: \n");
    //string 에 붙히고 You win input number %d 출력

    write(clnt_socks[k], smsg, strlen(smsg));
}

```

```

    #if 0
        for(i = 0; i < clnt_cnt; i++)
        {
            if(data[i] > cmp)
                sprintf(smsg, "greater than %d\n", cmp);
            else if(data[i] < cmp)
                sprintf(smsg, "less than %d\n", cmp);
            else
                strcpy(smsg, "You win!\n");

            strcat(smsg, "Input Number: ");
            write(clnt_socks[i], smsg, strlen(smsg));
        }
    #endif

    pthread_mutex_unlock(&mtx);
}

void *clnt_handler(void *arg) //arg 에 피쓰레드의 인자가 들어옴
{
    int clnt_sock = *((int *)arg);
    int str_len = 0, i;
    char msg[BUF_SIZE] = {0};
    char pattern[BUF_SIZE] = "Input Number: \n";

    signal(SIGALRM, sig_handler);

    pthread_mutex_lock(&mtx); //락 걸어주는이유는 데이터 꼬이지말라고,
                                //크리디트섹션방지
    thread_pid[idx++] = getpid(); //피쓰레드값에 getpid아이디값을 넣고있음
                                //쓰레드의 피아이디값을 여기에 저장
    i = idx - 1; // index 를 위해 1 을 빼줌
    printf("i = %d\n", i);
    write(clnt_socks[i], pattern, strlen(pattern));
    //clnt 소켓에서는? idx 는 0 부터시작
    //첫번째 클라이언트에 패턴을써주겠다.

    pthread_mutex_unlock(&mtx);
    //그리고 뮤텍스 언락->락을 풀어주겠다

    alarm(3);

    while((str_len = read(clnt_sock, msg, sizeof(msg))) != 0)
    //패턴보내서 read 는 클라이언트에 무언가 숫자를입력했을때 수신하는거
    //수신하는건 msg 로, 수신해줬으니 알람으로들어가서 초기화, 꺼줌
    //str_len->얼마만큼들어왔는지
    {
        alarm(0);
        proc_msg(msg, str_len, i);
        alarm(3);
    }

    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;
                            }
                        }

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

                    return NULL;
                }

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock;
    struct serv_addr, clnt_addr;
    socklen_t addr_size;
    pthread_t t_id;
    int idx = 0;

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

    srand(time(NULL));

    pthread_mutex_init(&mtx, NULL); //락 건네없으니 mtx 락을 널로초기화

    serv_sock = socket(PF_INET, SOCK_STREAM, 0);

    if(serv_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 = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("bind() error");

    if(listen(serv_sock, 2) == -1)
        //2 명받음
        err_handler("listen() error");

    for(;;) //무한루프
    {
        addr_size = sizeof(clnt_addr);
        clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_addr, &addr_size);
        // 기다림

        thread_pid[idx++] = getpid();

        pthread_mutex_lock(&mtx);
        //락을 걸고있음 ->데이터포이기 방지이유
        //쓰레드가 돌고있는상황이라도 socket에 접근불가
    }
}

```

```

        data[clnt_cnt] = rand() % 3333 + 1;
        clnt_socks[clnt_cnt++] = clnt_sock;
        pthread_mutex_unlock(&mtx);

        pthread_create(&t_id, NULL, clnt_handler, (void *)&clnt_sock);
        //피쓰레드 만드는중
        //clnt_hadler ->쓰레드가 되는 함수 그자체
        //&cln_sock->쓰레드가 전달되는 인자

        pthread_detach(t_id);
        //사용법을 잘모르겠으면 man_pthread_detach
        //detach 뜻은 떼어내다, attach 는 붙이다,
        //t_id 의 위치는 메인에 피쓰레드 t_id 지역변수로있음
        //t_id 는 쓰레드 아이디값, 식별자 아이디
        //즉, 쓰레드를 분리시키겠다

        printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
    }

    close(serv_sock);

    return 0;
}

```

컴파일방법

```

gcc -o serv gserv.c -lpthread
gcc -o clnt gclnt.c -lpthread

```

## file\_client.c

```

#include<fcntl.h>
#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;

#define BUF_SIZE 32

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

int main(int argc, char **argv)
{
    char buf[BUF_SIZE] = {0};
    int fd, sock, read_cnt;
    si serv_addr;

    if(argc !=3)
    {

```

```

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

    fd = open("receive.txt", O_CREAT | O_WRONLY);
    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");

    else
        puts("Connected .....");

    while((read_cnt = read(sock, buf, BUF_SIZE)) != 0)
        write(fd, buf, read_cnt);
    puts("Recelved File Data");
    write(sock, "Thank you", 10);
    close(fd);
    close(sock);

return 0;
}

```

파일로 서버만들기

## file\_server.c

//패턴은 반복되고 알고리즘은 for 문 옆사람테 파일전송

```

#include<fcntl.h>
#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;

#define BUF_SIZE          32

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

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock, fd;

```



```

char buf[BUF_SIZE] = {0};
int read_cnt;

si serv_addr, clnt_addr;
socklen_t clnt_addr_size;

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

fd = open("file_server.c", O_RDONLY);//파일을 바꿔주려면 여기수정
serv_sock = socket(PF_INET, SOCK_STREAM, 0);

if(serv_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 = htonl(INADDR_ANY);
serv_addr.sin_port = htons(atoi(argv[1]));

if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("bind() error");

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

clnt_addr_size = sizeof(clnt_addr);

clnt_sock = accept(serv_sock, (sap)&clnt_addr, &clnt_addr_size);

for(;;)
{
    read_cnt = read(fd, buf, BUF_SIZE);

    if(read_cnt < BUF_SIZE)//읽을양이 더많다면
    {
        write(clnt_sock, buf, read_cnt);//클라이언트 sock 에 써줌
        //read_cnt 는 읽은 바이트수
        break; //끝냄
    }
    write(clnt_sock, buf, BUF_SIZE);
}

shutdown(clnt_sock, SHUT_WR);
read(clnt_sock, buf, BUF_SIZE);
printf("msg form client: %s\n", buf);

close(fd);
close(clnt_sock);
close(serv_sock);

return 0;
}

```

# gethostbyname.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>

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

int main(int argc, char **argv)
{
    int i;
    struct hostent *host;

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

    host = gethostbyname(argv[1]);

    if(!host)
        err_handler("gethost ... error!");

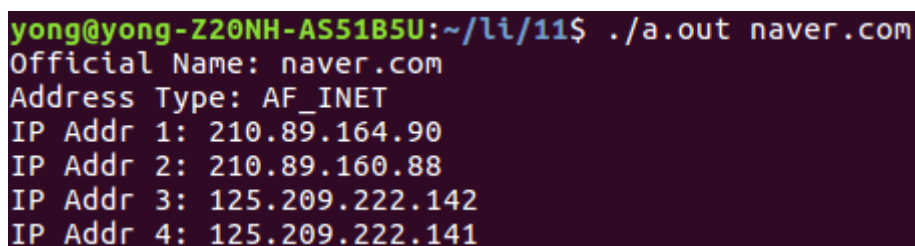
    printf("Official Name: %s\n", host->h_name);

    for(i=0; host->h_aliases[i]; i++)
        printf("Aliases %d: %s\n", i+1, host->h_aliases[i]);

    printf("Address Type: %s\n",
           (host->h_addrtype == AF_INET) ? "AF_INET" :
"AF_INET6");

    for(i=0; host->h_addr_list[i]; i++)
        printf("IP Addr %d: %s\n", i+1,
               inet_ntoa(*(struct in_addr *)host-
>h_addr_list[i]));

    return 0;
}
```



```
yong@yong-Z20NH-AS51BSU:~/li/11$ ./a.out naver.com
Official Name: naver.com
Address Type: AF_INET
IP Addr 1: 210.89.164.90
IP Addr 2: 210.89.160.88
IP Addr 3: 125.209.222.142
IP Addr 4: 125.209.222.141
```

ex) ./a.out naver.com  
보안 위하여 ip 값은 계속 바뀐다

## 1. pthread 활용법

### pthread\_create //쓰레드 생성을 위해서 사용

```
ex)int pthread_create(pthread_t *thread, const pthread_attr_t *attr,  
void *(*start_routine)(void *), void *arg);
```

첫번째 아규먼트인 thread 는 쓰레드가 성공적으로 생성되었을때 생성된 쓰레드를 식별하기 위해 사용되는 쓰레드 식별자이다.

두번째 아규먼트인 attr 은 쓰레드 특성을 지정하기위해서 사용하며, 기본 쓰레드 특성을 이용하고자 할경우에 NULL 을 사용한다.

3 번째 아규먼트인 start\_routine 는 분기시켜서 실행할 쓰레드 함수이며,

4 번째 아규먼트인 arg 는 쓰레드 함수의 인자이다.

성공적으로 생성될경우 0 을 리턴한다.

### pthread\_join

```
ex)#include <pthread.h>  
int pthread_join(pthread_t th, void **thread_return);
```

첫번째 아규먼트 **th** 는 기다릴(join)할 쓰레드 식별자이며, 두번째 아규먼트 **thread\_return** 은 쓰레드의 리턴(return) 값이다. thread\_return 이 NULL 이 아닐경우 해당 포인터로 쓰레드 리턴 값을 받아올수 있다.

### pthread\_detach

```
ex)int pthread_detach(pthread_t th);
```

detach 는 "떼어내다" 라는 뜻을 가지며 main 쓰레드에서 pthread\_create 를 이용해 생성된 쓰레드를 분리시킨다. 이 함수는 식별번호 **th** 인 쓰레드를 detach 시키는데, detach 되었을경우 해당 (detach 된) 쓰레드가 종료될경우 pthread\_join 을 호출하지 않더라도 즉시 모든 자원이 해제(free)된다.

### pthread\_mutex\_init

```
ex)int pthread_mutex_init(pthread_mutex_t * mutex,  
const pthread_mutex_attr *attr);
```

mutex 는 여러개의 쓰레드가 공유하는 데이터를 보호하기 위해서 사용되는 도구로써, 보호하고자 하는 데이터를 다루는 코드영역을 단지 한번에 하나의 쓰레드만 실행가능 하도록 하는 방법으로 공유되는 데이터를 보호한다. 이러한 코드영역(하나의 쓰레드만 점유가능한)을 critical section 이라고 하며, mutex 관련 API 를 이용해서 관리할수 있다.

pthread\_mutex\_init 는 mutex 객체를 초기화 시키기 위해서 사용한다. 첫번째 인자로 주어지는 mutex 객체 mutex 를 초기화시키며, 두번째 인자인 attr 를 이용해서 mutex 특성을 변경할수 있다. 기본 mutex 특성을 이용하기 원한다면 NULL 을 사용하면 된다.

### pthread\_mutex\_lock

```
ex)int pthread_mutex_lock(pthread_mutex_t *mutex);
```

pthread\_mutex\_lock 는 critical section 에 들어가기 위해서 mutex lock 을 요청한다. 만약 이미 다른 쓰레드에서 mutex lock 를 얻어서 사용하고 있다면 다른 쓰레드에서 mutex lock(뮤텍스 잠금) 을 해제할때까지(사용할수 있을때까지) 블럭 된다.

만약 다른 어떤 쓰레드에서도 mutex lock 을 사용하고 있지 않다면, 즉시 mutex lock 을 얻을수 있게 되

고 critical section 에 진입하게 된다. critical section 에서의 모든 작업을 마쳐서 사용하고 있는 mutex lock 이 더이상 필요 없다면 pthread\_mutex\_unlock 를 호출해서 mutex lock 를 되돌려준다.

### **pthread\_mutex\_unlock**

```
ex)int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

critical section 에서의 모든 작업을 마치고 mutex lock 을 돌려주기 위해서 사용한다.  
pthread\_mutex\_unlock 를 이용해서 mutex lock 를 되돌려주면 다른 스레드에서 mutex lock 를 얻을수 있는 상태가 된다