

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

<리눅스 네트워크 프로그래밍>
2018.03.30 - 27 일차

강사 - 이상훈
gcccompil3r@gmail.com

학생 - 안상재
sangjae2015@naver.com

* 소스코드 분석

1) 인터넷 표준 점 표기법 형태 → 네트워크 바이트 순서 이진형태로 변환 (호스트에서 스위치로 전송)

- int inet_aton(const char *cp, struct in_addr *inp) : Internet-host-address 의 cp 를 IPv4-number-and-dots-notation 형태에서 이진형태로(network-byte-order) 변환해서, inp 가 가리키는 구조체 안에 저장한다.

(IPv4-number-and-dots-notation 형태 → 네트워크 이진형태)

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

typedef struct sockaddr_in    si;

void err_handler(char *msg)
{
    write(2, msg, strlen(msg));
    exit(1);
}

int main(int argc, char **argv)
{
    char *addr = "127.124.73.31";
    si addr_inet;

    if(!inet_aton(addr,&addr_inet.sin_addr)) // 인터넷 표준 점 표기법 형태에서 네트워크 이진 형태로 변환
        err_handler("Conversion Error!"); // inet_aton 의 반환값이 -1(에러)이면 err_handler()로 이동
    else
        printf("Network Ordered Integer Addr : %#x\n", addr_inet.sin_addr.s_addr);

    return 0;
}
```

2) host 바이트 순서 → 네트워크 바이트 순서로 변환 , 네트워크 이진 형태 → 인터넷 표준 점 표기법 형태로 변환(스위치에서 호스트로 전송)

- uint32_t htonl(uint32_t hostlong) : unsigned integer hostlong 변수를 host byte order 에서 network byte order 로 변환.

- char *inet_ntoa(struct in_addr in) : network-byte-order 로 되어있는 Internet-host-address 를 IPv4 dotted-decimal-notation 으로 변환해줌. (네트워크 이진형태 → IPv4-dotted-decimal-notation 형태)

```
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>

typedef struct sockaddr_in    si;

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

```

{
    si addr1, addr2;
    char *str;
    char str_arr[32] = {0};

    addr1.sin_addr.s_addr = htonl(0x10203040);    // host 바이트 순서에서 network 바이트 순서로 변환함
    addr2.sin_addr.s_addr = htonl(0x12345678);

    str = inet_ntoa(addr1.sin_addr); // 네트워크 바이트 순서의 이진형태에서 인터넷 표준 점 표기법 형태로 변환
    strcpy(str_arr, str);
    printf("Not 1 : %s\n", str);

    inet_ntoa(addr2.sin_addr); // 컴파일러가 똑똑해져서 str 를 안써도 알아서 str 에 저장해줌
    printf("Not 2 : %s\n", str);
    printf("Not 3 : %s\n", str_arr);

    return 0;
}

```

3) 메아리 통신

* 알고리즘 순서

- 클라이언트 프로세스에서 소켓파일에 메시지를 write 함.
- 서버 프로세스에서 소켓파일의 메시지를 read 함.
- 서버 프로세스에서 소켓파일로부터 read 한 메시지를 다시 소켓파일에 write 함.
- 클라이언트 프로세스에서 소켓파일의 메시지를 read 함.
- 클라이언트 프로세스에서 'q' 또는 'Q' 를 입력하면 소켓파일을 close 하고 종료됨.
- 클라이언트 프로세스가 종료되면, 서버 프로세스도 클라이언트용 소켓파일을 close 하고 다른 클라이언트의 접속을 기다림.

* 중요 개념

- 서버 프로세스의 bind() 시스템 콜은 소켓파일에 서버 프로세스의 ip 주소를 할당함.
- 서버 프로세스의 listen() 시스템 콜은 접속 가능한 클라이언트의 갯수를 셋팅함.
- 서버 프로세스의 accept() 시스템 콜은 클라이언트 프로세스의 connect()를 기다리는 blocking 함수임.
- 클라이언트 프로세스의 connect() 시스템 콜은 소켓파일에 목적지 ip 주소, 포트번호를 등록함.

<서버 프로세스>

```

#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    1024

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

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

    char msg[BUF_SIZE];

    si serv_addr, clnt_addr;
    socklen_t clnt_addr_size;

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

    serv_sock = socket(PF_INET, SOCK_STREAM, 0); // 통신을 위한 종말점을 만들고 fd 를 반환함

    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); /* INADDR_ANY 는 어느 ip 주소를 통해 접속하
                                                    더라도 모두 서비스를 허용함*/
    serv_addr.sin_port = htons(atoi(argv[1]));

    if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1) // 소켓파일에 자신의 수신가능한 ip 주소,
                                                                    포트번호 등록함.
        err_handler("bind() error");

    if(listen(serv_sock, 5) == -1) // 허용 가능한 client 숫자 지정
        err_handler("listen() error");

    clnt_addr_size = sizeof(clnt_addr);

    for(i=0; i<5; i++)
    {
        clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size); /* 클라이언
                                                                                          트의 연결을 받아들임*/

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

```

```

        printf("Connected Client %d\n", i+1);
        while((str_len = read(clnt_sock, msg, BUF_SIZE)) != 0) /* read()는 blocking 함수임
            write(clnt_sock, msg, str_len);

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

```

////////////////////////////////////

<클라이언트 프로세스>

```
#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 1024
```

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

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

```
int sock, str_len;
si serv_addr;
char msg[32];
char *m = "Input Message(q to quit) : ";
```

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

```
sock = socket(PF_INET, SOCK_STREAM, 0); // 통신을 위한 종말점을 만들고 fd 를 반환함
```

```
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]));
```

록함

```
if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1) // 소켓파일에 목적 ip 주소, 포트번호를 등
    err_handler("connect() error");
else
    puts("Connected .....");

for(;;)
{
    fputs("Input msg(q to quit) : ", stdout);    //fputs == write
    fgets(msg, BUF_SIZE, stdin);                // fgets == read

    if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
        break;

    write(sock, msg, strlen(msg));
    str_len = read(sock, msg, BUF_SIZE - 1);

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

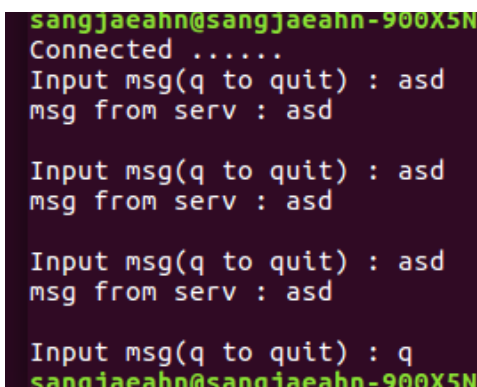
    msg[str_len] = 0;

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

return 0;
}
```

3-1) 결과 분석

- 클라이언트에서 메시지를 입력하면 서버에서 그 메시지를 읽고 다시 보내준다. 클라이언트에서 다시 그 메시지를 읽고 화면에 출력한다. 클라이언트에서 q 를 입력하면 종료되고, 서버에서는 다른 클라이언트의 접속을 기다린다. 접속가능한 클라이언트의 갯수는 5 개이다.

A terminal window with a dark background and green text. The prompt is 'sangjaeahn@sangjaeahn-900X5N'. The output shows 'Connected', followed by three times 'Input msg(q to quit) : asd' and 'msg from serv : asd'. Finally, 'Input msg(q to quit) : q' is entered, and the prompt 'sangjaeahn@sangjaeahn-900X5N' appears again.

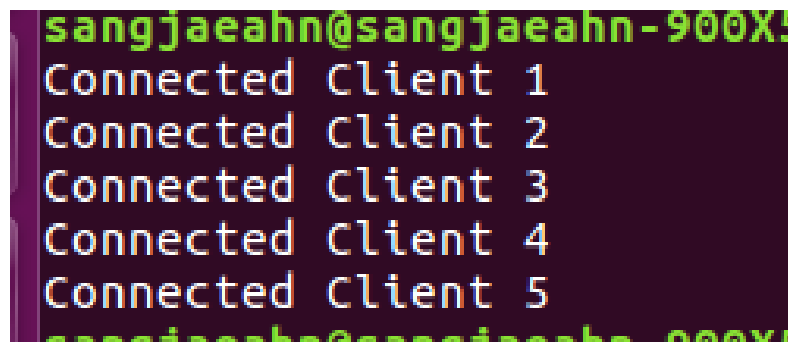
```
sangjaeahn@sangjaeahn-900X5N
Connected .....
Input msg(q to quit) : asd
msg from serv : asd

Input msg(q to quit) : asd
msg from serv : asd

Input msg(q to quit) : asd
msg from serv : asd

Input msg(q to quit) : q
sangjaeahn@sangjaeahn-900X5N
```

그림 1. 클라이언트 프로세스의
출력 화면

A terminal window with a dark background and green text. The prompt is 'sangjaeahn@sangjaeahn-900X5N'. The output shows 'Connected Client 1', 'Connected Client 2', 'Connected Client 3', 'Connected Client 4', and 'Connected Client 5'. The prompt 'sangjaeahn@sangjaeahn-900X5N' is visible at the bottom.

```
sangjaeahn@sangjaeahn-900X5N
Connected Client 1
Connected Client 2
Connected Client 3
Connected Client 4
Connected Client 5
sangjaeahn@sangjaeahn-900X5N
```

그림 2. 서버 프로세스의 출력화면

4) TCP 통신을 이용한 사칙연산

<서버 프로세스>

```
#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 1024
#define OPSZ 4

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

int calculate(int opnum, int *opnds, char op)
{
    int result = opnds[0], i;

    switch(op)
    {
        case '+':
            for(i=1; i<opnum; i++) // 이미 0 번을 받았으므로 1 번부터 시작
                result += opnds[i];
            break;
        case '-':
            for(i=1; i<opnum; i++)
                result -= opnds[i];
            break;
        case '*':
            for(i=1; i<opnum; i++)
                result *= opnds[i];
            break;
    }

    return result;
}

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock;
    char opinfo[BUF_SIZE];

    int result, opnd_cnt, i;
    int recv_cnt, recv_len;

    si serv_addr, clnt_addr;
```

```

socklen_t clnt_addr_size;

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

serv_sock = socket(PF_INET, SOCK_STREAM,0);

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

memset(&serv_addr, 0, sizeof(serv_addr)); // 네트워크 관련 구조체안의 값을 0 으로 초기화시킴
serv_addr.sin_family = AF_INET;           /* 서버 프로세스의 네트워크 관련 구조체의 멤버값 셋팅 */
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) /* 서버 프로세스의 소켓파일에 통신가능
                                                                한 ip 주소, 포트번호 등록*/
    err_handler("bind() error");

if(listen(serv_sock, 5) == -1) // 접속 가능한 클라이언트 프로세스의 갯수를 5 개로 셋팅함
    err_handler("listen() error");

clnt_addr_size = sizeof(clnt_addr);

for(i=0;i<5;i++) // 최대 5 개의 클라이언트 프로세스에 대한 반복문
{
    opnd_cnt = 0;
    clnt_sock = accept(serv_sock, (sap)&clnt_addr, &clnt_addr_size);
    read(clnt_sock, &opnd_cnt, 1);

    recv_len = 0;

    while((opnd_cnt * OPSZ + 1) > recv_len) // OPSZ 는 operand 사이
    {
        recv_cnt = read(clnt_sock, &opinfo[recv_len], BUF_SIZE - 1); // read 의 반환값은
                                                                    읽은 바이트 수*/
        recv_len += recv_cnt;
    }

    result = calculate(opnd_cnt, (int *)opinfo, opinfo[recv_len - 1]); // opnd_cnt 는 배열안의 전
                                                                    체 데이터 갯수*/
    write(clnt_sock, (char *)&result, sizeof(result));

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

```

////////////////////////////////////

<클라이언트 프로세스>

```
#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 1024
#define RLT_SIZE 4
#define OPSZ 4
```

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

/*

1. 소켓 파일 연다.
2. sockaddr_in 구조체 초기값 설정
3. 소켓파일과 sockaddr_in 구조체의 주소 연결

*/

```
int main(int argc, char **argv)
{
    int i, sock, result, opnd_cnt;
    char opmsg[BUF_SIZE] = {0};
    si serv_addr;

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

    sock = socket(PF_INET, SOCK_STREAM, 0); /* PF_INET : IPv4 인터넷 프로토콜 체계,
                                             SOCK_STREAM : TCP 통신 */

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

    memset(&serv_addr, 0, sizeof(serv_addr)); // serv_addr 을 serv_addr 의 사이즈만큼 0 으로 초기화
    serv_addr.sin_family = AF_INET;           // IPV4 주소체계
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]); // 주소
    serv_addr.sin_port = htons(atoi(argv[2])); // 포트번호

    if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1) // 소켓파일과 serv_addr 주소를 연결함
        err_handler("connect () error");
    else
```

```

        puts("connected.....");

fputs("Operand Cnt : ", stdout);
scanf("%d", &opnd_cnt); // 피연산자 갯수 입력

opmsg[0] = (char)opnd_cnt;

for(i=0;i<opnd_cnt;i++)
{
    printf("Operand %d: ", i+1);
    scanf("%d", (int *)&opmsg[i*OPSZ+1]); /* "1 + 2 = 3" 의 스타일로 만들기 위해서 배열형태를
                                                맞춤, 피연산자 입력 */
}

fgetc(stdin);
fputs("Operator: ", stdout);
scanf("%c", &opmsg[opnd_cnt*OPSZ + 1]); // 사용자가 원하는 연산자 입력
write(sock, opmsg, opnd_cnt*OPSZ + 2); // 입력받은 연산자까지 첨부해서 배열의 모든 데이터를 write 함
read(sock, &result, RLT_SIZE); // 서버 프로세스에서 소켓파일에서 write 한 메시지를 read 함

printf("Operation result : %d\n", result);
close(sock);

return 0;
}

```

4-1) 결과 분석

- 5 번째 클라이언트가 접속할 때는 정상적으로 허용이 되다가, 6 번째 클라이언트가 접속하는 순간 접속이 허용되지 않는다.

```

Operation result : 6
sangjaeahn@sangjaeahn-900X5N:~/code/linux/network/3.30$ ./clnt 127.0.0.1 7777
connected.....
Operand Cnt : 1
Operand 1: 1
Operator: +
Operation result : 1
sangjaeahn@sangjaeahn-900X5N:~/code/linux/network/3.30$ ./clnt 127.0.0.1 7777
connected.....
Operand Cnt : 2
Operand 1: 3
Operand 2: 1
Operator: -
Operation result : 2
sangjaeahn@sangjaeahn-900X5N:~/code/linux/network/3.30$ ./clnt 127.0.0.1 7777
connected.....
Operand Cnt : 2
Operand 1: 2
Operand 2: 2
Operator: *
Operation result : 4
sangjaeahn@sangjaeahn-900X5N:~/code/linux/network/3.30$ ./clnt 127.0.0.1 7777
connect () error
sangjaeahn@sangjaeahn-900X5N:~/code/linux/network/3.30$

```