TI DSP, MCU 및
Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
27 일차 (2018. 03. 30)

인터넷 주소 변환 함수

네트워크의 기준은 big endian 이다.

※ inet_addr()

이 함수는 Dotte-Decimal Notation (10 진수 표현방식)을 빅엔디안 32 비트 값으로 변환시키는 함수이다.

unsigned long inet_addr(const char *string);

함수의 원형인데 함수 파라미터 값에 IP 주소 문자열의 시작주소를 넣어주면 알아서 big endian 32 비트 unsigned long 형의 값으로 만든다. 성공하면 big endian 형식 32 비트 값을, 실패하면 -1 (INADDR_NONE)을 리턴한다.

※ inet_aton()

기본적으로 주소 문자열을 빅엔디안 32 비트 값으로 변환 시켜주는건 inet_addr()와 같지만 다른 함수다. inet_addr()는 변환된 값을 sockaddr_in 구조체의 변수 안에 또 선언 된 in_addr 구조체에 값을 대입해주어야 한다. 하지만 inet_aton()는 자동으로 값을 대입시킨다. 따라서 따로 대입해야 할 필요가 없다.

int inet_aton(const char *string, struct in_addr *addr);

첫번째 파라미터 값은 주소 문자열이 있는 포인터변수가 들어가고 두번째 파라미터 값은 구조체 변수 중 주소값이 들어가는 변수의 주소값을 넣는다. 함수를 성공시키면 0 이 아닌값, 실패하면 0 이 리턴된다.

※ inet_ntoa()

네트워크 바이트 순서의 32 비트 값을 Dotted-Decimal Notation 으로 변환시켜주는 함수이다.

※ inet_aton()와 반대되는 개념이라고 생각하면 된다.

char * inet_ntoa(struct in_addr addr);

파라미터값은 구조체의 변수 중 주소 문자열이 있는 변수를 넣는다. 이 함수가 성공했을 시 해당 문자열의 포인터를 리턴하고, 실패하면 -1 을 리턴한다.

inet_aton()

```
#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))
        err_handler("Conversion Error!");
    else
```

```
    printf("Network Ordered Integer Addr:%#x₩n", addr_inet.sin_addr.s_addr);

  return 0;     }
```

결과

```
Network Ordered Integer Addr:0x1f497c7f
```

inet_ntoa()

```
#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);
    addr2.sin_addr.s_addr = htonl(0x12345678);

    str = inet_ntoa(addr1.sin_addr);
    strcpy(str_arr, str);
    printf("Not 1 :%s₩n", str);

    inet_ntoa(addr2.sin_addr); //원래는 'str ='이 붙어야 한다. 컴파일러의 최적화로 가능한 코드이다.
    printf("Not 3: %s₩n", str);
    printf("Not 4: %s₩n", str_arr);

    return 0;     }
```

결과

```
Not 1 :16.32.48.64
Not 3: 18.52.86.120
Not 4: 16.32.48.64
```

echo_server.c

```
#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("sue :%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));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(atoi(argv[1]));  //여기까지 기존 예제와 같다. 형식이다.

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

    if(listen(serv_sock, 5) == -1) // 5 명 받을 수 있다.
     err_handler("listen() err");

    clnt_addr_size = sizeof(clnt_addr);
```

```
   for(i=0; i<5; i++)
   { clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size);
//accept 가 하는 것? 승인 > 클라이언트 전용 소켓
     if(clnt_sock == -1)
       err_handler("accept() err");
     else
       printf("Connected Clinet %d₩n", i =1);

     while((str_len = read(clnt_sock, msg, BUF_SIZE)) !=0)
       write(clnt_sock, msg, str_len);
// 에코 해주고 있다. 끝날 일이 없다. read, write 반복 다른 사람 쓰는 것이 읽히지 않는다. read 가
블록이라서 언제 마무리가 되는가? 클라이언트를 전부 끄게 되면 뭔가 주르륵 나온다 묶여 있었기 때문이다.
묶인 것이 풀리면서 다른 애들이 처리 되면서 한 번에 나오는 것이다. 이 부분에 문제를 해결하려면 리드
라이트에 논블로킹을 준다.

     close(clnt_sock);
   }
   close(serv_sock);

  return 0;        }
```

echo.client.c

```
#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;
```

```
 int 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);

 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])); //패턴이라서 앞부분은 같다.

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

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

 for(;;) //차이가 나는 곳은 여기 무한루프 돌면서 인풋 메세지와 메세지를 받고 있다
 { fputs("Input msg(q to quit): ", stdout);          //write 랑 같다. 메시지를 입력 받는다.
    fgets(msg, BUF_SIZE, stdin);                    //read 랑 같다

    if(!strcmp(msg, "q₩n") || !strcmp(msg, "Q₩n"))       //메시지를 q, Q 로 입력했을 경우, 끝이 난다.
      break;

    write(sock, msg, strlen(msg));          //fgets 에서 받은 것을 sock 에 넣는다.
    str_len = read(sock, msg, BUF_SIZE -1);

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

   msg[str_len] = 0;

   printf("msg form serv : %s₩n", msg); //서버에서 받은 msg 가 나온다. 구현은 read, write 로 해도 같다.
```

```
  }
  close(sock);

 return 0;        } //정상적으로 하려면 논블로킹을 하거나 포크를 추가 한다.
```

결과 통신은 아직이고 에코만 된다. 서버 말고 클라이언트 쪽에서만 입력한 값이 다시 되돌아 온다,

```
Connected.....
Input msg(q to quit): dfaf
msg form serv : dfaf

Input msg(q to quit): dfdf
msg form serv : dfdf

Input msg(q to quit):
```

Q, q 를 입력하기 전까지 클라이언트는 위와 같이 작동한다.

op_server.c // 계산은 서버가 한다.

```
#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         //Operation Size

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++) //위에 이미 0 을 받았기 때문에 1 부터 시작하는 것
                        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));
        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);

    for(i=0;i<5;i++) // 여러명이 접속 할 수 있으니까 접속에 대비하는 것
    {       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) //recv_len 0 끊기는 상황 방지하기 위해서
        { recv_cnt = read(clnt_sock, &opinfo[recv_len], BUF_SIZE-1);//읽은 바이트 수가 리턴
         recv_len += recv_cnt;                          }

    result = calculate(opnd_cnt, (int*)opinfo, opinfo[recv_len-1]);
    // 배열의 시작은 0 부터이니까 recv_len -1 이 되는 것
        write(clnt_sock, (char*)&result, sizeof(result));

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

op_client.c

```
#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);
        fputc('₩n', stderr);
        exit(1);  }
```

```
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);

        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.....");

        fputs("Operand Cnt: ", stdout);
        scanf("%d", &opnd_cnt); //opnd_cnt 에 값을 입력해줌
        opmsg[0]= (char)opnd_cnt;      //위에 받은 값을 char 형으로  변환하여 저장함

        for(i=0;i<opnd_cnt;i++)
        {       printf("Operand %d:", i+1);
                scanf("%d", (int*)&opmsg[i*OPSZ+1]);            }

        fgetc(stdin);      //read 에서 1 바이트만 읽음
        fputs("Operator: ", stdout);
        scanf("%c", &opmsg[opnd_cnt * OPSZ+ 1]);
        write(sock, opmsg, opnd_cnt*OPSZ+2);
        read(sock, &result, RLT_SIZE);

        printf("Operation result: %d₩n", result);
        close(sock);
        return 0;          }
```