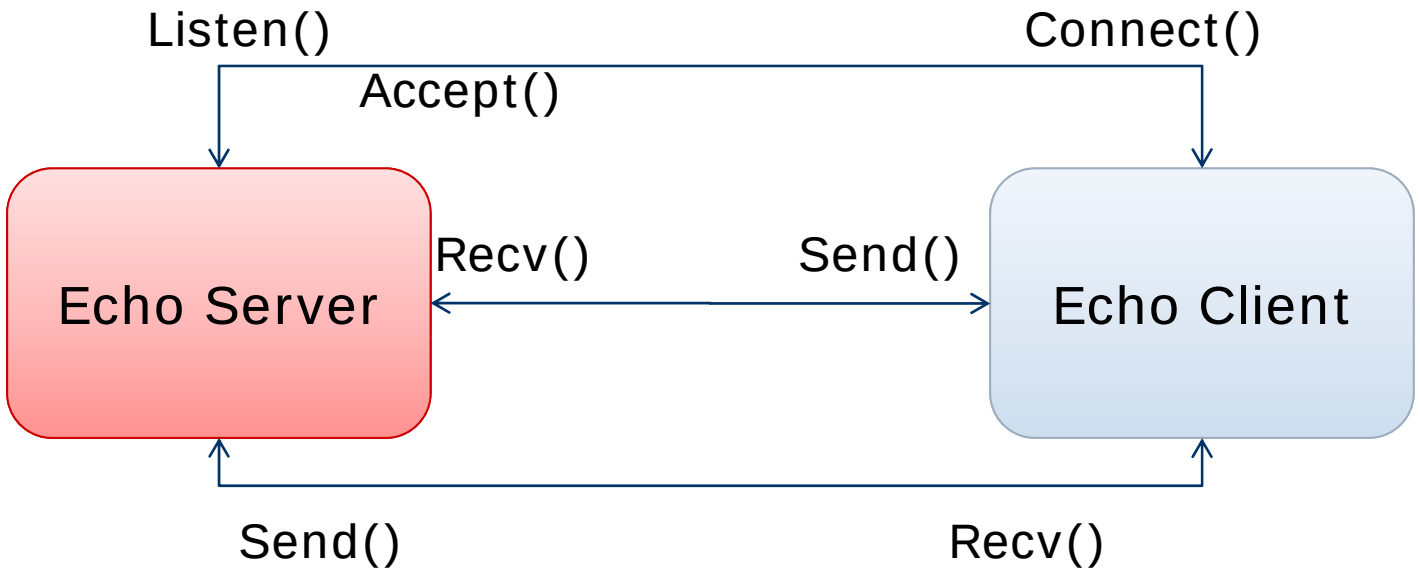
리눅스 프로그래밍

Echo Server & Client Programming

Mcalab
작성자 : 양평우
정용희

Socket

- Echo Server - Client



Echo Server Code

- echos.c

```c
#include <stdio.h> //printf(), fprintf()
#include <sys/socket.h> //socket(), bind(), connect()
#include <arpa/inet.h> //sockaddr_in, inet_ntoa()
#include <stdlib.h> //atoi()
#include <string.h> //memset()
#include <unistd.h> //close()

#define msgbuffsize 100 //받은 메시지를 담을 버퍼의 크기

int main(int argc, char * argv[])
{
    int serv_sock,cnt_sock; //에코서버는 자신의 소켓과 클라이언드의 소켓이 필요합니다.
    //그래서 두개의 소켓 식별자를 선언

    struct sockaddr_in serv_addr, cnt_addr; //마찬가지로 서버와 클라이언드의 주소 구조체
    unsigned short serv_port; //서버의 포드를 담을 변수
    unsigned int cnt_addr_len; //클라이언드의 주소길이를 담을 변수
    char recv_msg[msgbuffsize]; //받은 메시지를 담을 버퍼
    int recv_msgsize; //받은 메시지의 길이를 담을 변수
```

Echo Server Code(2)

- echos.c(2)

```
if(argc != 2)
{
    fprintf(stderr,"How to : %s <server port>\n",argv[0]);
    exit(1);
}
/* 두개의 인자를 넣지 않으면 즉, [파일명][포드번호] 이 형식을 안쓰면 에러처리와 사용법
설명 후 종료*/
serv_port = atoi(argv[1]);

//두번째 아규먼드가 포드번호이고 그 문자열을 정수화시켜서 정수를 저장
serv_sock = socket(PF_INET,SOCK_STREAM,0); //서버 소켓 생성

if(serv_sock < 0)
{
    fprintf(stderr,"socket() failed\n");
    exit(1);
}
//소켓생성에 실패하면 -1을 반환하기 때문에 0보다 작으면 실패이고 에러처리 후 종료
```

Echo Server Code(3)

- echos.c(3)

```
memset(&serv_addr,0,sizeof(serv_addr));
//서버의 주소를 담을 구조체 변수를 0으로 깨끗하게 초기화 시킴
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(serv_port);

if(bind(serv_sock,(struct sockaddr*)&serv_addr,sizeof(serv_addr)) < 0)
{
    fprintf(stderr,"bind() failed\n");
    exit(1);
}

if(listen(serv_sock,3) < 0)
{
    fprintf(stderr,"listen() failed\n");
    exit(1);
}
```

bind()함수는 지정된 소켓에 대해 지역 주소 및 포트번호를 채워줍니다.

listen()함수는 TCP연결이 클라이언트로부터 들어오는 연결들을 허용하도록 합니다

Echo Server Code(4)

- echos.c(4)

```
while(1)
{ //들어오는 연결을 반복적으로 처리
    cnt_addr_len = sizeof(cnt_addr); //클라이언트 주소 구조체의 길이를 변수에 저장
    cnt_sock = accept(serv_sock,(struct sockaddr*)&cnt_addr,&cnt_addr_len);

    if(cnt_sock < 0)
    {
        fprintf(stderr,"accept() failed\n");
        exit(1);
    }

    //소켓 생성 실패시 에러처리
    printf("client ip : %s\n",inet_ntoa(cnt_addr.sin_addr));

    //어디에서 접근했는지 단순 클라이언트 IP출력
    recv_msgsize = recv(cnt_sock,recv_msg,msgbuffsize,0);

    /*recv()함수는 전달받은 데이터의 바이트수를 반환해줍니다. 실패하면 -1을 반환
    하죠.. 받은 메시지 크기를 받을 recv_msgsize에 그 크기를 전달합니다.
    그리고 recv_msg라는 버퍼에 그 데이터를 넣습니다. */
    if(recv_msgsize < 0)
    {
        fprintf(stderr,"recv() failed\n");
    }
```

서버는 accept()함수를 호출하는데 이 함수는 듣고 있는 소켓의 포트번호로 들어오는 연결 요구가 있을 때까지 블록됩니다.

Echo Server Code(5)

- echos.c(5)

```
        //recv()함수가 -1을 반환하면 실패했다는 뜻이고 그때 동작시킬 에러처리
        while(recv_msgsize > 0)
        {
            if(send(cnt_sock,recv_msg,recv_msgsize,0) != recv_msgsize)
            {
                fprintf(stderr,"send() failed\n");
            }

            /* send()함수 역시 성공하면 보낸 데이터의 바이트 수를 반환시킵니다.
            전달받은 데이터의 크기와 보낸 데이터의 크기가 다르다면 에코역할을
            못한것이고 그때 에러처리를 해줍니다. */
            recv_msgsize = recv(cnt_sock,recv_msg,msgbuffsize,0);

            //상대가 전달할 데이터가 더 있는지 확인합니다.
            if(recv_msgsize < 0)
            {
                fprintf(stderr,"recv() failed\n");
            }
            close(cnt_sock);

            /* 할 일을 다한 클라이언트 소켓은 해지시킵니다. while문이 끝나면
            다시 전의 while의 첫번째로 돌아가서 대기를 합니다. */
        }
    }
}
```

Echo Client Code

- echoc.c

```c
#include <stdio.h> //printf(), fprintf()
#include <sys/socket.h> //socket(), connect(), send(), recv()
#include <arpa/inet.h> //sockaddr_in , inet_addr()
#include <stdlib.h> //atoi()
#include <string.h> //memset()
#include <unistd.h> //close()

#define msgsize 100 //recv 버퍼 크기 설정

int main(int argc,char* argv[])
{
    int cnt_sock; //사용할 소켓 변수
    struct sockaddr_in serv_addr; //서버의 주소를 담을 구조체 변수
    unsigned short serv_port; //서버의 포트를 담을 변수
    char *serv_ip; //서버의 ip를 문자로 입력받을 포인터 변수
    char *msg; //전달할 메시지를 문자로 받을 포인터 변수
    char msgbuff[msgsize]; //전달 받은 메시지를 받을 버퍼
    unsigned int msglen; //메시지의 길이를 담을 변수
    int recvbyte, total_recvbyte=0; //전달받은 메시지의 길이를 담을 변수

    //총 받은 메시지의 길이를 담을 변수
```

Echo Client Code(2)

- echoc.c(2)

```
if(argc!=4)
{ //사용법이 어긋나면 에러처리

    fprintf(stderr,"how to : %s server_ip server_port msg\n",argv[0]);
    exit(1);
}

serv_ip = argv[1]; //두번째 아규먼드는 입력한 서버 ip이고 serv_ip에 그 문자를 입력
msg = argv[3]; //네번째 아규먼드는 입력한 문자열이고 msg에 그 문자열 입력
serv_port = atoi(argv[2]); /*세번째 아규먼드는 입력한 서버 포드번호이고 serv_port
에 그 문자열을 입력하는데 atoi()함수 (숫자만으로 이루어진 문자열을 정수로 바꾸는 함수)를
이용해 문자열을 정수로 바꿔 정수를 입력*/

cnt_sock = socket(PF_INET,SOCK_STREAM,0); // 소켓 생성

if(cnt_sock < 0)
{ //socket()함수가 실패하면 -1을 반환하기 때문에 0보다 작으면 실패

    fprintf(stderr,"socket() failed\n");
    exit(1);
}
```

Echo Client Code(3)

- echoc.c(3)

```
memset(&serv_addr,0,sizeof(struct sockaddr_in));
/*주소가 적절히 채워지고 나머지는 0으로 채워져야 하기 때문에 0으로 초기화 시킴*/

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(serv_ip);
serv_addr.sin_port = htons(serv_port);

//connect()함수를 사용해 접속하고 실패하면 에러처리
if(connect(cnt_sock,(struct sockaddr*) &serv_addr,sizeof(serv_addr)) < 0)
{
    fprintf(stderr,"connect() failed\n");
    exit(1);
}
msglen = strlen(msg); //입력 메시지의 길이를 추출

//send()함수를 이용해 메시지를 전달하고 실패하면 에러처리
if(send(cnt_sock,msg,msglen,0)!=msglen)
{
    fprintf(stderr,"send() failed\n");
    exit(1);
}
printf("received : ");
```

Echo Client Code(4)

- echoc.c(4)

```
    while(total_recvbyte < msglen)
    {

    /*recv()함수를 이용해 상대가 보낸 데이터를 설정한 버퍼에 받고 받은 데이터의
    길이를 반환하는 특징을 이용해 recvbyte에 받은 데이터의 길이를 받음
    실패하면 에러처리를 합니다. */
        if((recvbyte = recv(cnt_sock,msgbuff,msgsize-1,0))<=0)
        {
            fprintf(stderr,"recv() failed\n");
            exit(1);
        }

        total_recvbyte += recvbyte; //전체 받은 바이드수를 알아야 하기때문에..
        msgbuff[recvbyte] = '\0'; //마지막은 NULL
        printf("%s",msgbuff); //전달받은 메시지 출력

    }
    printf("\n");
    close(cnt_sock); //소켓 해지
    exit(1); //프로그램 종료
}
```

소스 코드 실행

- Echo Server – Client 소스코드 컴파일
  - echos.c, echoc.c 파일을 각각 gcc 컴파일러를 통해 실행파일 생성
- 또 하나의 터미널창(쉘)을 실행하여 각각 Server와 Client실행
  - Server : ./실행파일 (포트)

```
root@ubuntu:/home/yongza/study/echo# ./echos.exe 8000
client ip : 192.168.174.129
client ip : 192.168.174.129
```

  - Cleint : ./실행파일 (IP 주소) (포트) (메시지)

```
root@ubuntu:/home/yongza/study/echo# ./echoc.exe 192.168.174.129 8000 heo
received : heo
root@ubuntu:/home/yongza/study/echo# ./echoc.exe 192.168.174.129 8000 hello
received : hello
root@ubuntu:/home/yongza/study/echo#
```