lect4: Review on socket programming

1. Communication using a file

p1:

x=open("/aa/bb", O\_WRONLY, 00777); // open a file

write(x, "korea", 5); // write some data

close(x); // close it

p2:

x=open("/aa/bb", O\_RDONLY, 00777); // open the same file

y=read(x, buf, 50); // read max 50 bytes from file x into buf

buf[y]=0; // make it a string

printf("the data was %s\n", buf); // check the contents

close(x);

2. Communication using a socket

p1(client):

x=socket(PF\_INET, SOCK\_STREAM, 0);

connect(x, &serv\_addr, sizeof(serv\_addr)); // open a socket

write(x, "hi", 2); // write some data

.........

p2(server):

x1=socket(PF\_INET, SOCK\_STREAM, 0);

bind(x1, &serv\_addr, sizeof(serv\_addr));

listen(x1, 5);

x2=accept(x1, &cli\_addr, &xx); // open a socket

y= read(x2, buf, 2); // read

..........

3. Example

client:

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define SERV\_TCP\_PORT 9924

#define SERV\_ADDR "165.246.38.157"

main(){

   int x, y;

   struct sockaddr\_in  serv\_addr;

   char buf[100];

   printf("Hi, I am the client\n");

   bzero((char \*) &serv\_addr, sizeof(serv\_addr));

   serv\_addr.sin\_family = PF\_INET;

   serv\_addr.sin\_addr.s\_addr = inet\_addr(SERV\_ADDR);

   serv\_addr.sin\_port = htons(SERV\_TCP\_PORT);

   /\* open a tcp socket\*/

   if ( (x =socket(PF\_INET, SOCK\_STREAM,0)) < 0){

      perror("socket creation error\n");

      exit(1);

   }

   printf(" socket opened successfully. socket num is %d\n", x);

   /\* connect to  the server \*/

   if (connect(x, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr))<0){

      perror("can't connect to the server\n");

      exit(1);

   }

    /\* send msg to the server \*/

    printf("now i am connected to the erver. enter a string to send\n");

scanf("%s", buf);

    write(x,buf,strlen(buf));

// read from the server

    printf("now let's read from the server\n");

    y=read(x,buf,50);

    buf[y]=0;

    printf("what echoed from the server is %s\n",buf);

close(x); // disconnect the connection

}

server:

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define SERV\_TCP\_PORT 9924

#define SERV\_ADDR "165.246.38.157"

main(){

        int s1,s2,x, y;

        struct sockaddr\_in serv\_addr, cli\_addr;

        char buf[100];

        size\_t xx;

        printf("Hi, I am the server\n");

        bzero((char \*)&serv\_addr, sizeof(serv\_addr));

        serv\_addr.sin\_family=PF\_INET;

        serv\_addr.sin\_addr.s\_addr=inet\_addr(SERV\_ADDR);

        serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);

// open a tcp socket

        if((s1=socket(PF\_INET, SOCK\_STREAM, 0))<0){

                perror("socket creation error\n");

                exit(1);

        }

        printf("socket created successfully. socket num is %d\n", s1);

// bind ip

        x=bind(s1, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr));

        if (x < 0){

           perror("binding failed\n");

           exit(1);

        }

        printf("binding passed\n");

        listen(s1, 5);

        xx = sizeof(cli\_addr);

        s2 = accept(s1,(struct sockaddr \*)&cli\_addr,&xx);

        printf("we passed accept. new socket num is %d\n", s2);

// read msg from client

printf(“now reading from client\n”);

        y=read(s2,buf,50);

        buf[y]=0;

        printf("we got %s from cli\n",buf);

// send msg to the client

        printf("what do you want to send to cli? enter your string\n");

        scanf("%s", buf);

        write(s2,buf,strlen(buf));

        close(s2); // disconnect the connection

close(s1); // close the original socket

}

4. protocol

1) well-known port

Internet service programs are waiting on well-known ports for service. The client can talk to these service programs by opening a socket on a well-known port and following the corresponding protocol.

21: ftp, 23: telnet, 25: smtp, 80: http, ........................

2) SMTP (Simple Mail Transfer Protocol) (rfc 2821) : for sending mail (port 25)

kchang(mail.inha.ac.kr) ==> kchang(mail.inha.ac.kr)

S: EHLO 165.246.38.219

R: 250-portal.inha.ac.kr Hello

   250-TURN

   .............

   250 OK

S: MAIL FROM:<kchang@inha.ac.kr>

R: 250 ............... OK

S: RCPT TO:<kchang@mail.inha.ac.kr>

R: 250 ................

S: DATA

R: 354 Start mail input; end with <CRLF>.<CRLF>

S: Blah blah blah...

S: <CRLF>.<CRLF>

R: 250 ..........

S: QUIT

R: Connection closed

3) POP3 (rfc 1939) : for retrieving mail (port 110)

S: USER \*\*\*\*

R: +OK

S: PASS \*\*\*\*

R: +OK

S: stat

R: +OK 1 539

S: list

R: +OK 1 539

S: retr 1

R: +OK

   mail here .........

S: quit

4) http client/server

http client(web browser):

      ..........

      #define SERV\_TCP\_PORT 80

      ..........

      connect(s, ........);

      write(s, "GET / HTTP/1.0\r\n\r\n", 18);  // 1st msg in http protocol

      read(s, buf, ..);

      .............

http server(web server):

     ...............

     s1=socket(...);

     bind(s1,...);  // bind on port 80

     listen(s1,...);

     for(;;){

       s2=accept(s1, .....);

       x=fork();

       if (x==0){

            close(s1);

            read(s2, in\_buf, n); // read http request

            build\_out\_buf(in\_buf, out\_buf);

                                //find the request html file and return with

// a proper header.

            write(s2, out\_buf, ....);

            close(s2);

            exit(0);

        } else cose(s2); /

     }

5. Homework

1) Copy serv.c.

- Open putty terminal and connect to 165.246.38.151 or 165.246.38.157

- Copy serv.c

$ cp ../../linuxer1/serv.c . (or "cp ../../linuxer3/serv.c ." in 157)

$ ls

serv.c

- Compile

$ gcc -o serv serv.c

2) Copy cli.c

- Open another putty terminal and connect to 165.246.38.151 or 165.246.38.157

- Copy cli.c

$ cp ../../linuxer1/cli.c . (or "cp ../../linuxer3/cli.c ." in 157)

$ ls

cli.c

- Compile

$ gcc -o cli cli.c

3) Adjust port number both server and client. Recompile both and run the server first and run the client next. The client should talk first and then the server.