

Luis Ruiz

CPE 400-1001

5001441817

Socket Programming Project

Server Program – Programed in C

- The server allows for multiple clients to connect to it. Note it will terminate once all clients have disconnected from it. However, there is a segment of code that will allow it to hang and wait for a connection if it is commented out; this snip will be highlighted below.
- Requires two arguments
 - ./executable portno

```
//CPE-400
```

```
//SECTION: 1001
```

```
//Luis Ruiz
```

```
//server code
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <sys/types.h>
```

```
#include <sys/time.h> //FD_SET,FD_CLR,FD_ZERO
```

```
#include <sys/socket.h>
```

```
#include <netinet/in.h>
```

```
#include <ctype.h>
```

```
#define MAX_CLIENT 30
```

```
#define TRUE 1
```

```
#define FALSE 0
```

```
///FUNCTIONS
```

```
void error(char* msg);
```

```
void palindrome(char* msg, int clientfd);
```

```
//MAIN
```

```
//-----
```

```
//-----
```

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

```
{
```

```
    int sockfd, portno;
```

```

int clientfd[MAX_CLIENT] = {0};
unsigned int clilen;
int new_clientfd;
fd_set readfds;

unsigned int count = 0;
int max_sd,sd,activity,valread;
unsigned int i;

char* buffer = (char*) calloc(500,sizeof(char));
struct sockaddr_in serv_addr, cli_addr;

if (argc < 2)
{
    fprintf(stderr, "\n%s\n", "ERROR: Please type in the following format");
    fprintf(stderr, "%s\n", "./executable portno.");
    error("\n");
}

//convert the port number to an integer
portno = atoi(argv[1]);

//start a new socket
sockfd = socket(AF_INET,SOCK_STREAM,0);
if(sockfd < 0)
    error("ERROR opening socket\n");

//zero out my address buffer
bzero((char *) &serv_addr, sizeof(serv_addr) );

serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(portno); //port number
serv_addr.sin_addr.s_addr = INADDR_ANY;

//To assign a local socket address.
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
    error("ERROR on binding\n");

if(listen(sockfd,MAX_CLIENT) < 0)
    error("Error listening to socket\n");

//get the lenght of the client address
clilen = sizeof(cli_addr);
puts("Waiting for connections ...");

```

```

//Loop that will test the clients sent string
while(TRUE)
{
    //clear the socket set
    FD_ZERO(&readfds);

    bzero(buffer,500);

    //add severs socket descriptor to set
    FD_SET(sockfd, &readfds);
    max_sd = sockfd;

    //add child sockets to set
    for ( i = 0 ; i < MAX_CLIENT ; i++)
    {
        //socket descriptor
        sd = clientfd[i];

        //if valid socket descriptor then add to read list
        if(sd > 0)
            FD_SET(sd,&readfds);

        //highest file descriptor number, need it for the select function
        if(sd > max_sd)
            max_sd = sd;
    }

    //checks for which client is active
    activity = select(max_sd+1, &readfds , NULL , NULL , NULL);

    //If something happened on the master socket ,
    //then its an incoming connection
    if (FD_ISSET(sockfd, &readfds))
    {
        new_clientfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
        if(new_clientfd < 0)
        {
            error("ERROR on accept\n");
        }

        //add new socket to array of sockets
        for (i = 0; i < MAX_CLIENT; i++)
        {
            //if position is empty
            if( clientfd[i] == 0 )
            {

```

```

        ++count;
        clientfd[i] = new_clientfd;
        printf("Adding to list of sockets as %d\n", i);

        break;
    }
}

//else operate on some other socket
for (i = 0; i < MAX_CLIENT; i++)
{
    //reset my buffer
    bzero(buffer,500);

    //grab the socket descriptor
    sd = clientfd[i];

    //check if that descriptor is in my montior
    if(FD_ISSET(sd ,&readfds))
    {
        if(read(clientfd[i],buffer,500) < 0)
            error("ERROR reading from socket\n");

        //The user simply pushed enter or ctrl+c to exit the connnection
        //on the other end
        if(strlen(buffer) <= 1)
        {
            fprintf(stderr, "Closing connection too Client %d\n", sd);
            close(sd);
            clientfd[i] = 0;
            --count;
            break;
        }
        //the socket has sent a valid string
        else
        {
            fprintf(stderr, "\nHere is a message from Client %d:
%s\n",sd,buffer);

            palindrome(buffer,sd);
        }
    }
}

//If we wish to have the server never close it's connections
//then can simply comment out this portion of the code

```

```

        //this will allow it to hang and wait for any connections
        if(count == 0)
        {
            FD_ZERO(&readfds);
            printf("\n%s\n", "No More Connection are available!!!");
            break;
        }
    }
}

```

exit:

```

    for(i = 0; i < MAX_CLIENT ;i++)
        if(clientfd[i]!=0)
            close(clientfd[i]);

    return 0;
}
//-----
//-----
//FUNCTION BODIES

```

```

void error (char* msg)
{
    fprintf(stderr, "%s\n", msg);
    exit(1);
}

```

```

void palindrome(char* msg,int clientfd)
{
    //temp will just hold a copy of the msg passed in
    //token will be used to tokenize the string
    //str will be the concatenation of the token strings
    char* temp = (char*)calloc(500,sizeof(char));
    char* token;
    char* str = calloc(500,sizeof(char));
    strcpy(temp,msg);

    //loop that is used to concatenate
    token = strtok(temp, " \n");
    while(token!=NULL)
    {
        strcat(str,token);
        token = strtok(NULL, " \n");
    }
}

```

```

//remove the linefeed
msg = strtok(msg, "\n");

//messages
char* isPalindrome = (char*)calloc(500,sizeof(char));
char* notP = ": is Not a Palindrome!!\n";
char* isP = ": is a Palindrome!!\n";

// Start from leftmost and rightmost corners of str
int l = 0;
int h = strlen(str)-1;

// Keep comparing characters while they are same
while (h > l)
{
    if(isalpha(str[l]) > 0)
        tolower(str[l]);

    if(isalpha(str[h]) > 0)
        tolower(str[h]);

    if (str[l++] != str[h--])
    {
        //send the message to our client
        strcpy(isPalindrome,msg);
        strcat(isPalindrome,notP);
        if(write(clientfd,isPalindrome,500) < 0)
            error("ERROR:writing to client file descriptor\n");
        return;
    }
}

//send the message to our client
strcpy(isPalindrome,msg);
strcat(isPalindrome,isP);
if(write(clientfd,isPalindrome,500) < 0)
    error("ERROR:writing to client file descriptor\n");
}

```

Client Program – Programed in C

- Simple client code that send a string to a server and receives an answer, when you compile the program it requires 3 arguments:
 - ./executable X portno.
 - X is the machine name and if on the same machine put *localhost*

```
//CPE-400
//SECTION: 1001
//Luis Ruiz
//Client code
```

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
```

```
#define TRUE 1
#define FALSE 0
```

```
//display error messages
void error(char *msg)
{
    perror(msg);
    exit(0);
}
```

```
void getIp(struct sockaddr_in serv_addr, struct sockaddr_in cli_addr)
{
    //get the server ip and convert it to a readable string
    struct sockaddr_in* IPV4_client = (struct sockaddr_in*)&cli_addr;

    //string that will hold the servers ip
    char client[INET_ADDRSTRLEN];
    inet_ntop( AF_INET, &IPV4_client, client, INET_ADDRSTRLEN );

    //get the client ip and convert it to a readable string
    struct sockaddr_in* IPV4_server = (struct sockaddr_in*)&serv_addr;
    struct in_addr server_addr = IPV4_server->sin_addr;

    //string that will hold the clients ip
    char server_ip[INET_ADDRSTRLEN];
    inet_ntop( AF_INET, &server_addr, server_ip, INET_ADDRSTRLEN );

    //print to stdout the two ips
    fprintf(stderr, "\nServer:%s\nClient:%s\n", server_ip, client);
}
```

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

```

{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr, cli_addr;
    struct hostent *server;
    char buffer[500];

    //check if hostname of server and port number is provided
    if (argc < 3)
    {
        fprintf(stderr, "%s\n", "ERROR:");
        fprintf(stderr, "usage %s hostname port\n", argv[0]);
        exit(0);
    }

    //get desired port connection
    portno = atoi(argv[2]);

    //creating our socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");

    //get the host name
    server = gethostbyname(argv[1]);
    if (server == NULL)
    {
        fprintf(stderr, "ERROR, no such host\n");
        exit(0);
    }

    //zero out the address first
    bzero((char *) &serv_addr, sizeof(serv_addr));

    //assign values to serv_addr variable
    serv_addr.sin_family = AF_INET;           //for IPv4 communication
        //copy h_addr to s_addr
    bcopy((char *)server->h_addr, (char *) &serv_addr.sin_addr.s_addr, server->h_length);

    //port number
    serv_addr.sin_port = htons(portno);

    getIp(serv_addr, cli_addr);

    //requesting connection to server
    if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR connecting");
}

```



```

while(TRUE)
{
    bzero(buffer,500);

    fprintf(stderr, "%s\n", "If you wish to exit, you can CTRL+C or simply press enter");
    fprintf(stderr, "%s", "Enter string you wish to transmit:" );

    //get string
    fgets(buffer,500,stdin);

    //if the buffer is empty end the client communication
    if(strlen(buffer) == 1 )
        return 0;

    //sending message to host server
    send(sockfd,buffer,sizeof(buffer),0);

    bzero(buffer,500);

    //recieving message from host server
    recv(sockfd,buffer,sizeof(buffer),0);
    printf("%s\n",buffer);

    bzero(buffer,500);
}

//close file descriptor
if(close(sockfd) < 0)
    error("ERROR: on closing");

return 0;
}

```

Images:

The image displays four terminal windows arranged in a 2x2 grid, all running on a machine with the username 'ruizl6' and directory '~/cpe400'. The top-left window shows the server program running, listening on port 8888. It receives messages from four clients: 'do geese see god', '123456', and two closing connections. The top-right window shows a client program connecting to localhost:8888, receiving the server's IP (127.0.0.1) and a client IP (192.191.176.104). It transmits '123456' and receives the response '123456: is Not a Palindrome!!'. The bottom-left window shows another client program connecting to localhost:8888, receiving the server's IP (127.0.0.1) and a client IP (176.109.233.27). It transmits 'do geese see god' and receives the response 'do geese see god: is a Palindrome!!'. The bottom-right window shows a third client program connecting to localhost:8888, receiving the server's IP (127.0.0.1) and a client IP (128.35.92.141). It transmits an empty string and receives the response 'Enter string you wish to transmit:'.

```
[ruizl6@cardiac cpe400]$ gcc -o server server.c
[ruizl6@cardiac cpe400]$ ./server 88888
Waiting for connections ...
Adding to list of sockets as 0
Adding to list of sockets as 1
Adding to list of sockets as 2

Here is a message from Client 4: do geese see god

Here is a message from Client 5: 123456

Closing connection too Client 6
Closing connection too Client 5
Closing connection too Client 4

No More Connection are available!!!
[ruizl6@cardiac cpe400]$
```

```
[ruizl6@cardiac cpe400]$ gcc -o client client.c
[ruizl6@cardiac cpe400]$ ./client localhost 88888

Server:127.0.0.1
Client:192.191.176.104
If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:123456
123456: is Not a Palindrome!!

If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:
[ruizl6@cardiac cpe400]$
```

```
[ruizl6@cardiac cpe400]$ gcc -o client client.c
[ruizl6@cardiac cpe400]$ ./client localhost 88888

Server:127.0.0.1
Client:176.109.233.27
If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:do geese see god
do geese see god: is a Palindrome!!

If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:
[ruizl6@cardiac cpe400]$
```

```
[ruizl6@cardiac cpe400]$ gcc -o client client.c
[ruizl6@cardiac cpe400]$ ./client localhost 88888

Server:127.0.0.1
Client:128.35.92.141
If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:
[ruizl6@cardiac cpe400]$
```

1:Figure display on server and 3 clients, all on the same machine

The image displays four terminal windows arranged in a 2x2 grid. The top-left window shows the server program running, listening on port 8888. It receives messages from three clients: 'different machine', '1001', and 'same machine'. The top-right window shows a client program connecting to localhost:8888, receiving the server's IP (127.0.0.1) and a client IP (160.227.181.171). It transmits 'same machine' and receives the response 'same machine: is Not a Palindrome!!'. The bottom-left window shows a client program connecting to localhost:8888, receiving the server's IP (127.0.0.1) and a client IP (112.64.200.194). It transmits 'test 2' and receives the response 'test 2: is Not a Palindrome!!'. The bottom-right window shows another client program connecting to localhost:8888, receiving the server's IP (127.0.0.1) and a client IP (16.205.153.78). It transmits '1001' and receives the response '1001: is a Palindrome!!'.

```
[ruizl6@cardiac cpe400]$ ./server 88888
Waiting for connections ...
Adding to list of sockets as 0
Adding to list of sockets as 1
Adding to list of sockets as 2

Here is a message from Client 5: different machine

Here is a message from Client 4: 1001

Here is a message from Client 6: same machine

Closing connection too Client 4
Closing connection too Client 5
Closing connection too Client 6

No More Connection are available!!!
[ruizl6@cardiac cpe400]$
```

```
[ruizl6@cardiac cpe400]$ ./client localhost 88888

Server:127.0.0.1
Client:160.227.181.171
If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:same machine
same machine: is Not a Palindrome!!

If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:
[ruizl6@cardiac cpe400]$
```

```
[ruizl6@cardiac cpe400]$ ./client localhost 88888

Server:127.0.0.1
Client:112.64.200.194
If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:test 2
test 2: is Not a Palindrome!!

If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:diffrent machine
diffrent machine: is Not a Palindrome!!

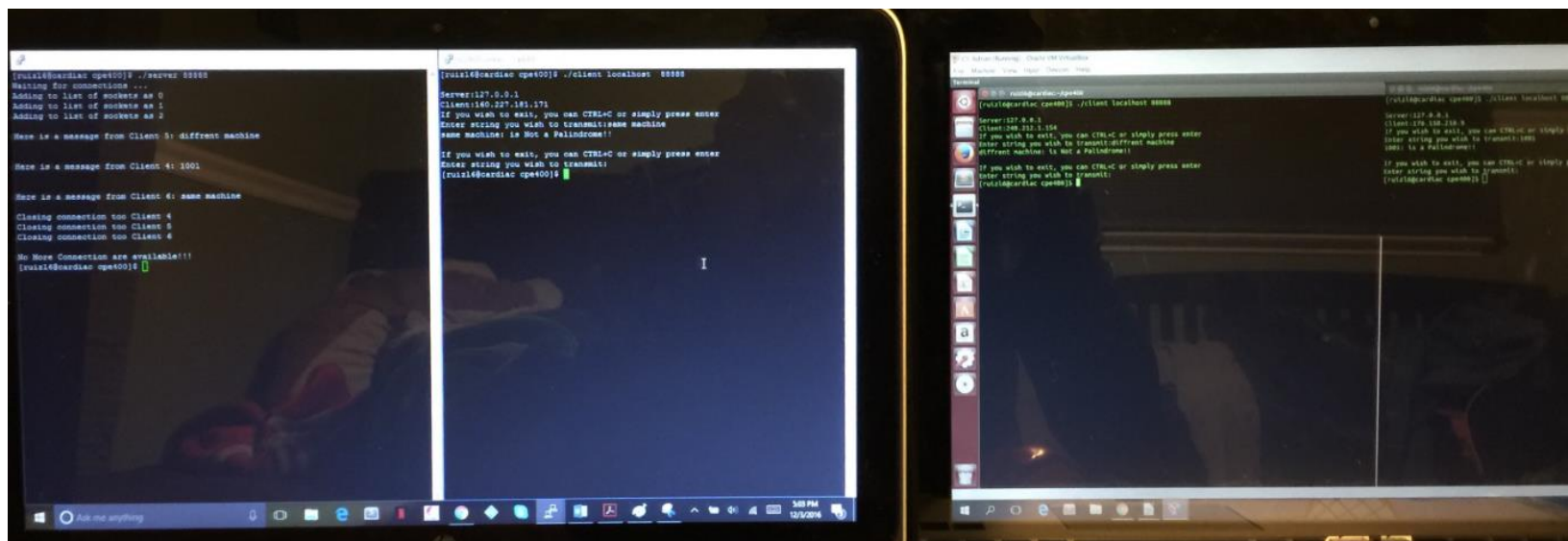
If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:
[ruizl6@cardiac cpe400]$
```

```
[ruizl6@cardiac cpe400]$ ./client localhost 88888

Server:127.0.0.1
Client:16.205.153.78
If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:1001
1001: is a Palindrome!!

If you wish to exit, you can CTRL+C or simply press enter
Enter string you wish to transmit:
[ruizl6@cardiac cpe400]$
```

2:Bottom image are 2 clients from different machines and the top are images are a server and client from my machine



3:An image of the two machines