

Project Report

**Systems Programming Lab**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**302L-Systems Programming Lab**

**Department of Computer System Engineering**

**University of Engineering and Technology Peshawar**

Submitted to: **Engr. Abdullah Hamid**

Submitted by:

**Muhammad Saad - 21PWCSE1997**

**Muhammad Umar Jan - 21PWCSE2000**

**Muhammad Ilyas - 21PWCSE2055**

**DCSE, Batch 23, Section “B”**

**Project Title: File transfer through Socket**

**Introduction:**

This project involves a simple client-server application. Initially, the client establishes a socket connection with the server. Subsequently, the server awaits the client's input. The client tpye a directory name to the server, which, in turn, responds by providing a list of all files within that directory. If the client inputs "send" followed by a specific filename within the directory, the server proceeds to transmit the corresponding file to the client.

**Project Code: Server**

#include *<stdio.h>*

#include *<sys/types.h>*

#include *<sys/socket.h>*

#include *<netinet/in.h>*

#include *<string.h>*

#include *<unistd.h>*

#include *<stdlib.h>*

#include *<errno.h>*

#include *<dirent.h>*

#include *<fcntl.h>*

#include *<sys/stat.h>*

**static** char buff[1024];

int giveDirList(char \*folderName)

{

**struct** **dirent** \*entry;

DIR \*folder;

**struct** **stat** Stat;

strcat(buff, folderName);

strcat(buff, "/");

folder = opendir(buff);

int mkff = mkfifo("fifo", S\_IRWXU);

int fifofd = open("fifo", O\_RDWR, S\_IRWXU);

**while** ((entry = readdir(folder)) != NULL)

{

**if** (strcmp(entry->d\_name, ".") != 0 && strcmp(entry->d\_name, "..") != 0)

{

stat(entry->d\_name, &Stat);

**if** (S\_ISDIR(Stat.st\_mode))

write(fifofd, "D**\t**", 2);

**else**

write(fifofd, "F**\t**", 2);

}

**else**

write(fifofd, "**\t**", 1);

write(fifofd, entry->d\_name, strlen(entry->d\_name));

write(fifofd, "**\n**", 1);

}

**return** fifofd;

}

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

{

int sockfd, fd, portno, clientLen, bnd, lstn, rd, wr;

char bigBuff[624288], buffer[1024];

**struct** **sockaddr\_in** server, client;

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

**if** (sockfd < 0) {

perror("Error opening socket");

exit(EXIT\_FAILURE);

}

memset((void \*)&server, 0, **sizeof**(server));

portno = atoi(argv[1]);

server.sin\_port = htons(portno);

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = INADDR\_ANY;

bnd = bind(sockfd, (**struct** **sockaddr** \*)&server, **sizeof**(server));

lstn = listen(sockfd, 5);

clientLen = **sizeof**(client);

fd = accept(sockfd, (**struct** **sockaddr** \*)&client, &clientLen);

rd = read(fd, buffer, 10);

printf("%s**\n**", buffer);

**while** (1)

{

char copy[1024];

int fd1;

memset((void \*)buffer, 0, **sizeof**(buffer));

rd = read(fd, buffer, 1024);

buffer[rd] = '\0';

printf("%s**\n**", buffer);

**if** (strcmp("send", buffer) == 0)

{

strcpy(copy, buff);

write(fd, ">**\t**", 2);

rd = read(fd, buffer, 1024);

strcat(copy, buffer);

int fdc = open(copy, O\_RDWR, S\_IRWXU);

**if** (fdc < 0)

{

printf("%s", buffer);

exit(-1);

}

**while** (1)

{

rd = read(fdc, bigBuff, 624288);

**if** (rd == 0)

**break**;

write(fd, bigBuff, rd);

}

}

**else**

{

fd1 = giveDirList(buffer);

rd = read(fd1, buffer, 1024);

write(fd, buffer, rd);

}

}

}

**Explanation**

1. Socket Initialization:
   * The program initializes a socket using socket() with the AF\_INET (IPv4) address family and SOCK\_STREAM (TCP) socket type.
   * The server socket is bound to a specific port and IP address using bind().
2. Listening for Connections:
   * The listen() function is called to make the server socket ready to accept incoming client connections.
   * The accept() function waits for and accepts an incoming connection, creating a new socket fd for communication with the connected client.
3. Read Initial Client Message:
   * The server reads the first 10 characters from the client, presumably receiving some initial message.
4. Main Loop:
   * The program enters a loop where it continuously reads messages from the client and responds accordingly.
5. Handling "send" Command:
   * If the client sends the string "send", the server interprets this as a request to send a file.
   * The server reads the next line from the client, which is expected to be a filename.
   * It constructs the full path to the file using a combination of the directory path stored in buff and the filename received.
   * The server then opens the file using open() and proceeds to read its content in chunks (624288 bytes) using read() and sends the data to the client using write().
6. Handling Directory Listing Command:
   * If the client sends a command other than "send", the server assumes it's a directory name.
   * It calls the giveDirList function to generate a list of files and directories within the specified directory.
   * The list is written back to the client using write().
7. Directory Listing Function (giveDirList):
   * This function takes a folder name as input and opens the directory using opendir().
   * It creates a named pipe ("fifo") for inter-process communication and opens it using open().
   * For each file in the directory, it checks whether it's a directory or a regular file using stat() and writes information about it to the named pipe.

**Client**

#include *<stdio.h>*

#include *<sys/types.h>*

#include *<sys/socket.h>*

#include *<netinet/in.h>*

#include *<netdb.h>*

#include *<stdlib.h>*

#include *<string.h>*

#include *<unistd.h>*

#include *<sys/stat.h>*

#include *<fcntl.h>*

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

{

int sockfd, portno, rd, wr, cn, prog=0;

**struct** **sockaddr\_in** serverAddress;

char buffer[1024], bigBuff[624288];

portno = atoi(argv[2]);

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

memset((void \*)&serverAddress, 0, **sizeof**(serverAddress));

memset((void \*)&serverAddress, 0, **sizeof**(serverAddress));

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(portno);

serverAddress.sin\_addr.s\_addr = inet\_addr(argv[1]);

cn = connect(sockfd, (**struct** **sockaddr** \*)&serverAddress, **sizeof**(serverAddress));

**if** (cn >= 0)

printf("Connected!!**\n**");

write(sockfd, "connected", 10);

**while** (1)

{

scanf("%s", buffer);

wr = write(sockfd, buffer, 1024);

rd = read(sockfd, buffer, 1024);

buffer[rd] = '\0';

**if** (buffer[0] == '>')

{

memset((void \*)buffer, 0, **sizeof**(buffer));

write(1, "**\n**What file do you what to send?**\t**", 32);

scanf("%s", buffer);

write(sockfd, buffer, **sizeof**(buffer));

int fdc = open(buffer, O\_WRONLY | O\_CREAT, 0777);

**if** (fdc < 0)

exit(-1);

**while** (1)

{

printf("%dkb done**\n**");

rd = read(sockfd, bigBuff, 624288);

prog+=(rd/1024);

**if** (rd == 0)

**break**;

write(fdc, bigBuff, rd);

}

}

**else**

write(1, buffer, rd);

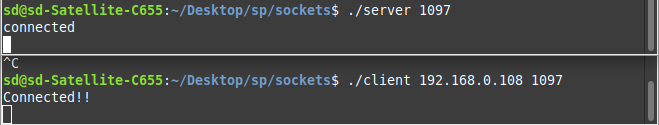
}

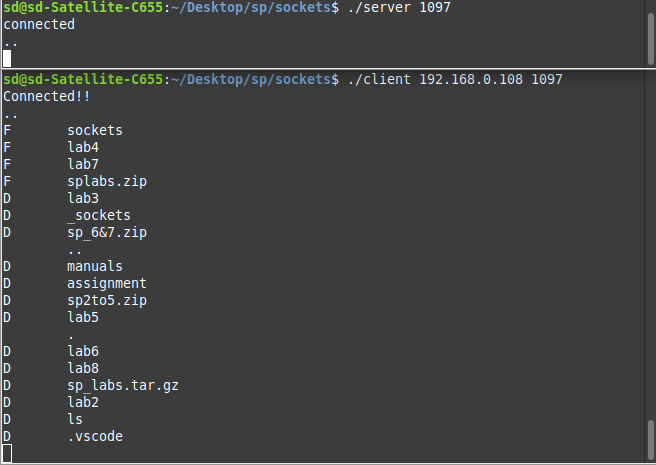
}

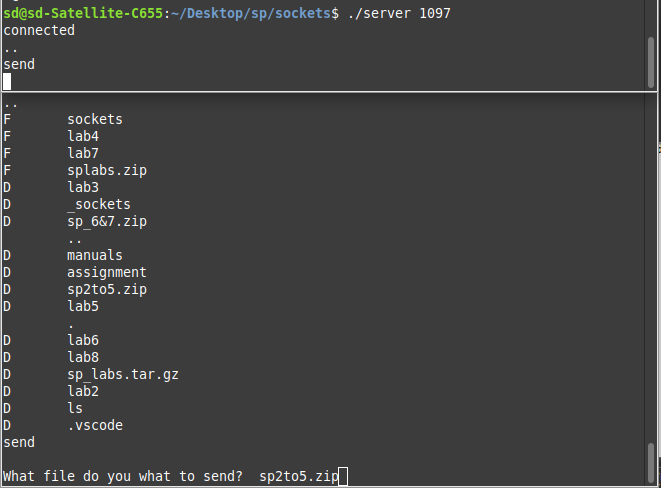
**Explanation**

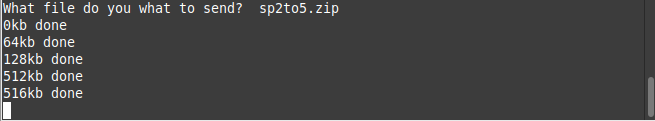
1. Socket Initialization:
   * The program initializes a socket using socket() with the AF\_INET (IPv4) address family and SOCK\_STREAM (TCP) socket type.
2. Server Connection:
   * It establishes a connection to the server using connect(). The server's IP address is provided as a command-line argument (argv[1]), and the port number is provided as argv[2].
3. Sending Initial Message:
   * The client sends the string "connected" to the server with write().
4. Main Loop:
   * The program enters a loop where it continuously reads input from the user and sends it to the server.
5. Handling "send" Command:
   * If the user inputs "send," the client prompts the user for a filename and sends it to the server using write().
6. Receiving File Data:
   * If the server responds with a line starting with ">", the client assumes that the server is ready to send a file.
   * The client prompts the user for the filename and sends it to the server using write().
   * It opens the specified file for writing (O\_WRONLY | O\_CREAT) and begins receiving file data from the server in chunks of 624,288 bytes (about 609 KB).
   * The progress of file download is printed periodically in kilobytes.
7. Handling Regular Messages:
   * If the server does not indicate a file transfer (no ">" at the beginning of the line), the client assumes it's a regular message and writes it to the standard output.

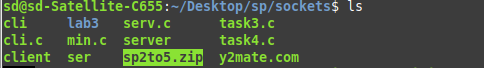
**Output**

**Connecting to the server**

**Server receiving directory name**

**Client requesting for a filer**

**File sent from server to client**



**File download in the same directory as client process**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**