Computer Networking

Assignment 3

Socket Programming

Class: F1403028

Name: Yi Jiang

Student ID: 5140219152

Table of Contents

1 Abstract

This project implements a simple file share application and it is about socket programming. It uses TCK Socket APIs to do the tasks. The peers could upload/download files to/from other peers. The C/S model should be implemented anyway and so are some basic functions, while the P2P model is optional.

2 C/S Model

In this part, the C/S model of this simple file share application is implemented. The server listens to a certain port, 2680 in the following, while the client initiates a TCP connection to the server to download a file. Then the server responds with the file. When 'Esc + Enter' is pressed, the connection would be torn down and otherwise, the client can repeatedly request file from the server.

2.1 Main Functions

1）Server listens to a given port: 2680;

2）Client initiates a TCP connection to the server with IP address 192.168.1.114 (the IP address of my PC) and port number 2680;

3）Client can send a request to the server to download a file/text. The filename is entered by the user;

4）Server respond client with the file/text;

5）Client saves the file locally;

6）Steps from 3 to 5 is repeated until 'Esc+Enter' is pressed, and until then client will tear the TCP connection.

2.2 How to Run

As I didn't write a GUI, a Visual Studio 2013 is needed to run the program.

First, open the server.sln and run the server program. The server program displays "Listening to Client…" if run correctly.

Then open the client.sln and run the client program. You may need to alter the ip address in the project property according to your PC if it is running under debug mode. After the client automatically connected to the given ip address and port number, "Please input filename you request on the server: " is displayed. And here you input the filename you request from the server and press "Enter".

If the file transmission is successful, the requested file can be found in client's folder and they are still connected for other file transfer unless "Esc+Enter" is pressed.

For any questions, you are welcome to contact elainejy@outlook.com.

2.3 Troubleshooting

2.3.1 Request Specific File from the Server

The sample code is just a basic example of socket connection and it basically just prints out the number of times the server has been visited. Thus, as you are required to pull a specific file from the server entered by the user, the filename should be sent to the server. To save the filename from the client side, a regular expression is used to extract the filename. Then it is put into a send buffer to send to the server side. After that, the sent filename is also saved in a buffer on the server side and the specific file can be sent.

2.3.2 Exit

To continually request files from the server side, a while function is used. Also to exit the program when pressing “Esc”, getchar() function cannot be used or the filename may be truncated. Thus I still use the regular expression above to check whether the first character input is “Esc”. If it is, the key will be 27 and you exit the program.

2.3.3 Debug

Some redundant printf and system(“pause”) is added in the program to debug. Also, the ip address of the PC may change, attention should be paid to that as the ip address attribute is pre set in the program manager.

2.3.4 File not Existed

If the requested file doesn’t exist on the server side, the client will still create a file named that but nothing is written into it and it is also displayed that it doesn’t exist.

2.4 Source Code

2.4.1 Client

```
/*
Client: connect the server, upload/download files
*/

#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys\stat.h>
```

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

#pragma comment(lib, "ws2_32.lib")
#define MAX 100                 //max buffer size
#define PORT 2680               //default TCP port number
#define FILE_NAME_MAX 512     //file name max size
#define BUFFER_SIZE 1024

int main(int argc, char * argv[]){
        struct sockaddr_in servaddr;       //struct to hold server's address
        int sockfd, n;          //socker descriptor
        char buf[MAX + 1];
        unsigned long temp;

        //start the winsock lib
#ifdef WIN32
        WSADATA wsaData;
        WSAStartup(0x0101, &wsaData);
#endif

        //set servaddr value
        memset((char *)&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        if (argc < 2){
                printf("usage:client<ipaddress>\n");
                system("pause");
                exit(0);
        }

        servaddr.sin_port = PORT;
        temp = inet_addr((const char FAR *) argv[1]);
        memcpy(&servaddr.sin_addr, &temp, sizeof(servaddr.sin_addr));

        //create a socket
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0){
                fprintf(stderr, "socket creation error\n");
                system("pause");
                exit(1);
        }

        //connect the socket to the specified server
```

```
    if (connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0){
        fprintf(stderr, "connect failed\n");
        system("pause");
        exit(1);
    }

    while (true){

        char file_name[FILE_NAME_MAX + 1];
        memset(file_name, 0, FILE_NAME_MAX + 1);
        printf("Please input filename you request on the server: ");
        scanf("%[^\n]%*c", &file_name);

        if (27 == file_name[0]){
            break;
        }

        char buffer[BUFFER_SIZE];
        memset(buffer, 0, BUFFER_SIZE);
        strncpy(buffer, file_name, strlen(file_name) > BUFFER_SIZE ? BUFFER_SIZE :
strlen(file_name));

        if (send(sockfd, buffer, BUFFER_SIZE, 0) < 0){
            printf("Send file name failed\n");
            system("pause");
            exit(1);
        }

        //open the file
        FILE *fp = fopen(file_name, "wb");
        if (NULL == fp){
            printf("File %s can not be opened\n", file_name);
            system("pause");
            exit(1);
        }
        else{
            memset(buffer, 0, BUFFER_SIZE);
            int length = 0;
            while ((length = recv(sockfd, buffer, BUFFER_SIZE, 0)) > 0){
                if (fwrite(buffer, sizeof(char), length, fp) < length){
                    printf("File %s write failed\n", file_name);
                    break;
                }
                memset(buffer, 0, BUFFER_SIZE);
```

```
                }

                printf("Received File %s from server success \n", file_name);
            }

            fclose(fp);
        }

        //close the socket
        closesocket(sockfd);
        system("pause");
        exit(0);
}
```

2.4.2 Server

```
/*
server: allocate a socket, respond to the client
*/

#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <io.h>
#include <conio.h>
#include <stdlib.h>

#pragma comment(lib, "ws2_32.lib")

#define MAX 100
#define QLEN 6          //size of request queue
#define PORT 2680
#define FILE_NAME_MAX_SIZE 512
#define BUFFER_SIZE 1024


int main(int argc, char * argv[]){
        struct sockaddr_in servaddr;       //hold server address
        struct sockaddr_in clntaddr;//hold client address
```

```
	int sockfd, addrlen, sockfd2;
	int visits = 0;
	char buf[MAX + 1];

	//start the winsock lib
#ifdef WIN32
	WSADATA wsaData;
	WSAStartup(0x0101, &wsaData);
#endif

	//set the servaddr value
	memset((char *)&servaddr, 0, sizeof(servaddr));
	servaddr.sin_family = AF_INET;
	servaddr.sin_port = PORT;
	servaddr.sin_addr.s_addr = INADDR_ANY;

	//create a socket
	sockfd = socket(PF_INET, SOCK_STREAM, 0);
	if (sockfd < 0){
		fprintf(stderr, "socket creation error\n");
		exit(1);
	}

	//bind servaddr to the socket
	if (bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0){
		fprintf(stderr, "connect failed\n");
		exit(1);
	}

	//listen to the socket, waiting for client
	if (listen(sockfd, QLEN) < 0){
		fprintf(stderr, "listen failed\n");
		exit(1);
	}

	//Main server loop: accept and handle requests
	while (1){
		printf("Listening To Client...\n");

		sockaddr_in client_addr;
		int client_addr_len = sizeof(client_addr);

		SOCKET m_New_Socket = accept(sockfd, (sockaddr *)&client_addr,
&client_addr_len);
```

```
        if (SOCKET_ERROR == m_New_Socket)
        {
            printf("Server Accept Failed: %d", WSAGetLastError());
            break;
        }

        char buffer[BUFFER_SIZE];
        memset(buffer, 0, BUFFER_SIZE);
        if (recv(m_New_Socket, buffer, BUFFER_SIZE, 0) < 0)
        {
            printf("Server Receive Data Failed!");
            break;
        }

        char file_name[FILE_NAME_MAX_SIZE + 1];
        memset(file_name, 0, FILE_NAME_MAX_SIZE + 1);
        strncpy_s(file_name, buffer, strlen(buffer)>FILE_NAME_MAX_SIZE ?
FILE_NAME_MAX_SIZE : strlen(buffer));
        printf("%s\n", file_name);

        FILE * fp = fopen(file_name, "rb");
        if (NULL == fp)
        {
            printf("File: %s Not Found\n", file_name);
        }
        else
        {
            memset(buffer, 0, BUFFER_SIZE);
            int length = 0;

            while ((length = fread(buffer, sizeof(char), BUFFER_SIZE, fp)) > 0)
            {
                if (send(m_New_Socket, buffer, length, 0) < 0)
                {
                    printf("Send File: %s Failed\n", file_name);
                    break;
                }
                memset(buffer, 0, BUFFER_SIZE);
            }

            fclose(fp);
            printf("File: %s Transfer Successful!\n", file_name);
        }
        closesocket(m_New_Socket);
```

```
    }
}
```

3 P2P Model

In this part, the P2P model of this simple file share application is implemented. Each peer can implement both client and server thread. Each of them can download file from others. The code is mainly similar to the C/S model, except that the peer can choose to be the server or client in the program

3.1 Main Functions

1）Peer can choose to be either client or server;

2）Server listens to a given port: 2680;

3）Client initiates a TCP connection to the server with IP address 192.168.1.114 (the IP address of my PC) and port number 2680;

4）Client can send a request to the server to download a file/text. The filename is entered by the user;

5）Server respond client with the file/text;

6）Client saves the file locally;

7）Steps from 4 to 6 is repeated until ‘Esc+Enter’ is pressed, and until then client will tear the TCP connection.

3.2 How to Run

As I still didn’t write a GUI, a Visual Studio 2013 is needed to run the program. First, open the peer1.sln and run the peer program. The peer program displays “Choose mode: client or server?” if run correctly.

Then input client or server to choose between the modes. It is recommended

to choose server mode first. After that, open another peer program, say peer2.sln. Run it and choose the client mode. Then follow the instructions and do the file transmission. It is very similar to the previous C/S model.

If the file transmission is successful, the requested file can be found in the folder of peer of client mode and they are still connected for other file transfer unless “Esc+Enter” is pressed.

For any questions, you are welcome to contact elainejy@outlook.com.

3.3 Troubleshooting

3.3.1 Choose Between the Client/Server Modes

The client and server side are implemented as two functions of peer. The main function chooses between them to run either client or server.

3.4 Source Code

3.4.1 Peer

```
/*
Peer: choose to be client or server
*/

#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <io.h>
#include <conio.h>
#include <stdlib.h>

#pragma comment(lib, "ws2_32.lib")
```

```
#define MAX 100                 //max buffer size
#define PORT 2680         //default TCP port number
#define FILE_NAME_MAX 512     //file name max size
#define BUFFER_SIZE 1024
#define QLEN 6         //size of request queue

int client(int argc, char * argv[]){
      struct sockaddr_in servaddr;   //struct to hold server's address
      int sockfd, n;       //socker descriptor
      char buf[MAX + 1];
      unsigned long temp;

      //start the winsock lib
#ifdef WIN32
      WSADATA wsaData;
      WSAStartup(0x0101, &wsaData);
#endif

      //set servaddr value
      memset((char *)&servaddr, 0, sizeof(servaddr));
      servaddr.sin_family = AF_INET;
      if (argc < 2){
            printf("usage:client<ipaddress>\n");
            system("pause");
            exit(0);
      }

      servaddr.sin_port = PORT;
      temp = inet_addr((const char FAR *) argv[1]);
      memcpy(&servaddr.sin_addr, &temp, sizeof(servaddr.sin_addr));

      //create a socket
      sockfd = socket(AF_INET, SOCK_STREAM, 0);
      if (sockfd < 0){
            fprintf(stderr, "socket creation error\n");
            system("pause");
            exit(1);
      }

      //connect the socket to the specified server
      if (connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0){
            fprintf(stderr, "connect failed\n");
            system("pause");
```

```
        exit(1);
    }

    while (true){

        char file_name[FILE_NAME_MAX + 1];
        memset(file_name, 0, FILE_NAME_MAX + 1);
        printf("Please input filename you request on the server: ");
        scanf("%[^\n]%*c", &file_name);

        if (27 == file_name[0]){
            break;
        }

        char buffer[BUFFER_SIZE];
        memset(buffer, 0, BUFFER_SIZE);
        strncpy(buffer, file_name, strlen(file_name) > BUFFER_SIZE ?
BUFFER_SIZE : strlen(file_name));

        if (send(sockfd, buffer, BUFFER_SIZE, 0) < 0){
            printf("Send file name failed\n");
            system("pause");
            exit(1);
        }

        //open the file
        FILE *fp = fopen(file_name, "wb");
        if (NULL == fp){
            printf("File %s can not be opened\n", file_name);
            system("pause");
            exit(1);
        }
        else{
            memset(buffer, 0, BUFFER_SIZE);
            int length = 0;
            while ((length = recv(sockfd, buffer, BUFFER_SIZE, 0)) > 0){
                if (fwrite(buffer, sizeof(char), length, fp) < length){
                    printf("File %s write failed\n", file_name);
                    break;
                }
                memset(buffer, 0, BUFFER_SIZE);
            }

            printf("Received File %s from server success \n", file_name);
```

```
            }

            fclose(fp);

        }

        //close the socket
        closesocket(sockfd);
        system("pause");
        exit(0);
}

int server(int argc, char * argv[]){
        struct sockaddr_in servaddr;   //hold server address
        struct sockaddr_in clntaddr;    //hold client address
        int sockfd, addrlen, sockfd2;
        int visits = 0;
        char buf[MAX + 1];

        //start the winsock lib
#ifdef WIN32
        WSADATA wsaData;
        WSAStartup(0x0101, &wsaData);
#endif

        //set the servaddr value
        memset((char *)&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = PORT;
        servaddr.sin_addr.s_addr = INADDR_ANY;

        //create a socket
        sockfd = socket(PF_INET, SOCK_STREAM, 0);
        if (sockfd < 0){
            fprintf(stderr, "socket creation error\n");
            exit(1);
        }

        //bind servaddr to the socket
        if (bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0){
            fprintf(stderr, "connect failed\n");
            exit(1);
        }
```

```
    //listen to the socket, waiting for client
    if (listen(sockfd, QLEN) < 0){
        fprintf(stderr, "listen failed\n");
        exit(1);
    }

    //Main server loop: accept and handle requests
    while (1){
        printf("Listening To Client...\n");

        sockaddr_in client_addr;
        int client_addr_len = sizeof(client_addr);

        SOCKET m_New_Socket = accept(sockfd, (sockaddr *)&client_addr,
&client_addr_len);
        if (SOCKET_ERROR == m_New_Socket)
        {
            printf("Server Accept Failed: %d", WSAGetLastError());
            break;
        }

        char buffer[BUFFER_SIZE];
        memset(buffer, 0, BUFFER_SIZE);
        if (recv(m_New_Socket, buffer, BUFFER_SIZE, 0) < 0)
        {
            printf("Server Receive Data Failed!");
            break;
        }

        char file_name[FILE_NAME_MAX + 1];
        memset(file_name, 0, FILE_NAME_MAX + 1);
        strncpy_s(file_name, buffer, strlen(buffer)>FILE_NAME_MAX ?
FILE_NAME_MAX : strlen(buffer));
        printf("%s\n", file_name);

        FILE * fp = fopen(file_name, "rb");
        if (NULL == fp)
        {
            printf("File: %s Not Found\n", file_name);
        }
        else
        {
            memset(buffer, 0, BUFFER_SIZE);
            int length = 0;
```

```
            while ((length = fread(buffer, sizeof(char), BUFFER_SIZE, fp)) > 0)
            {
                if (send(m_New_Socket, buffer, length, 0) < 0)
                {
                    printf("Send File: %s Failed\n", file_name);
                    break;
                }
                memset(buffer, 0, BUFFER_SIZE);
            }

            fclose(fp);
            printf("File: %s Transfer Successful!\n", file_name);
        }
        closesocket(m_New_Socket);
    }

}

int main(int argc, char * argv[]){
    char mode[100];
    printf("Choose mode: client or server? ");
    scanf("%s", &mode);
    if (strcmp(mode, "client")==0){
        client(argc, argv);
    }
    else{
        server(argc, argv);
    }
    return 0;
}
```

4 Conclusion

In this assignment, a simple file share application using TCP Socket APIs is implemented. The peers can upload/download files to/from other peers. Both C/S model and P2P model are implemented.

The main functions are very similar. Server listens to port 2680. Client initiates a TCP connection to server with default port number 2680. Client send a

request to download a specific file/text. Server responds to the client with the requested file/text. Client save the file to its local directory. The procedure is repeated until “Esc” is pressed. And the peer model combines the client and server together.

The main flaw is that no GUI is implemented. As for me, it’s hard to implement GUIs in C/C++ and time is also limited. Maybe for further improvement, I will do the task.

Besides, I’m not quite sure about the P2P model. As from my understanding, there should a centralized server to do the scheduling of file distribution. Anyway, it is implemented.

After finishing this assignment, I’ve got a better understanding of socket programming and some other parts of the course, which should be very useful.