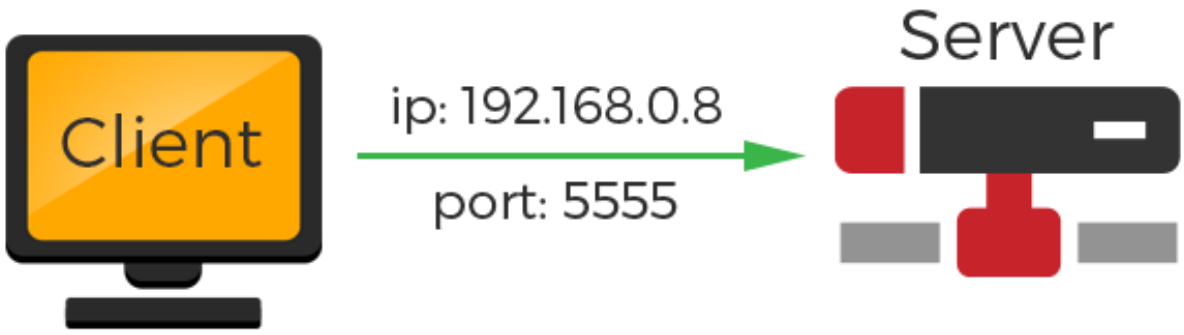


Object Oriented Programming: Internet Relay Chat

06.11.2016

—

Mainak Mondal
15BCA0005
Slot: A2+TA2
VIT University

Contents

Abstract

The inception of the biggest computer network, Internet, has revolutionised the world with seamless/effortless communication and data sharing.

Computer networks support an enormous number of applications and services such as access to the World Wide Web, digital video, digital audio, shared use of application and storage servers, printers, and fax machines, and use of email and instant messaging applications as well as many others. In most cases, application-specific communications protocols are layered (i.e. carried as payload) over other more general communications protocols.

C++ is a fairly recent Object Oriented Programming Language which exploits the internet to give users a better communication protocol through the introduction of sockets.

I will be using windows file sharing protocol and its addressing capabilities to build a windows-console based Chatting Client.

Windows Sockets API (WSA), which was later shortened to Winsock, is a technical specification that defines how Windows network software should access network services, especially TCP/IP. It defines a standard interface between a Windows TCP/IP client application (such as an FTP client or a web browser) and the underlying TCP/IP protocol stack.

It's not just making a software that counts, optimizing it is the real deal. A file sharing client for 1 to n clients is also available and uses the windows file sharing platform to do it.

Introduction

Internet Relay Chat (IRC) is an application layer protocol that facilitates communication in the form of text. The chat process works on a client/server networking model. IRC clients are computer programs that a user can install on their system. These clients communicate with chat servers to transfer messages to other clients. IRC is mainly designed for group communication in discussion forums, called channels, but also allows one-on-one communication via private messages as well as chat and data transfer, including file sharing.

Before internet relay chat, there was an application called Multiuser Talk(MUT), which was pretty useless because of one to one communication problem. The first IRC was created by Jarkko Oikarinen in August 1988. Jarkko intended to extend the BBS (Bulletin Board System) software he administered, to allow news in the Usenet style, real time discussions and similar BBS features. The first part he implemented was the chat part, which he did with borrowed parts written by his friends Jyrki Kuoppala and Jukka Pihl. The first IRC network was running on a single server named **"tolsun.oulu.fi.".** Oikarinen found inspiration in a chat system known as Bitnet Relay, which operated on the BITNET.

My inspiration to build a IRC is take features and build ideas from MUT and then implement it in the IRC, to better optimise it.

Concepts of Object Oriented Programming Used:

Objects:

Objects are the basic unit of OOP. They are instances of class, which have data members and uses various member functions to perform tasks.

Classes

It is similar to structures in C. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declare & defines what data variables the object will have and what operations can be performed on the class's object.

Encapsulation:

It can also be said data binding. Encapsulation is all about binding the data variables and functions together in class.

Inheritance

Inheritance is a way to reuse once written code again and again. The class which is inherited is called base calls & the class which inherits is called derived class. So when, a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

Polymorphism

It is a feature, which lets us create functions with same name but different arguments, which will perform differently. That is function with same name, functioning in different way. Or, it also allows us to redefine a function to provide its new definition. You will learn how to do this in details soon in coming lessons.

Module Breakup

1. Reader Module
   a. Pathfinder Class
   b. Reader Class(Loop)
   c. Start Setup.

2. Client Module (Writer)
   a. Pathfinder Class
   b. Client Class(Loop)
   c. Start Setup

3. Server Module
   a. Server Side Class
   b. Pathfinder Class
   c. Server Setup
   d. Server Loop.

4. File Sharing Module
   a. Sender Function
   b. Receiver Function

Reader Module

Pathfinder Class:

The class to store protected and sensitive shared location of the server, the Server Location Path. This class is approximately 5 lines but is the original configuration of the server shared location. To change the server, the configuration of this class should be updated to the new server location.

Reader Class:

The class has two parts an updating function, and a setup function. The main function or the setup function is more of a build up to the updating function. The setup is very important as it makes the console to get chat updates from the updating function. The loop function or the updating function is the heart of the reader class as it constantly updates the console with the incoming messages.

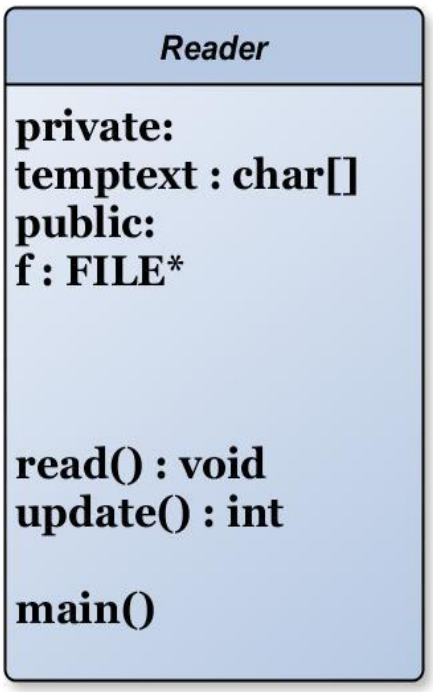
Start Setup:

A simple setup function, a replacement to the batch file to initiate the Reader Module and the Client Module.

How it works?

The Reader module looks for the chat log and updates itself and asks for a confirmation to go real time, once confirmed, it will go real time and update the console constantly until a admin kill code is received.

Class Diagram:



Code:

```
#include<iostream>
#include<string.h>
#include <stdlib.h>
#include<stdio.h>
#include<windows.h>
#include"pathfinder.cpp"
using namespace std;
class reader:pathfinder
{
	private:
		char temptext[500];

	public:
		FILE *f=fopen(path,"r");
		void read()
		{

			while (!feof(f))
			{
				fgets(temptext,500,f);
				cout<<"\n|\t"<<temptext;
```

```
                }

        }

        int update()
        {

                fgets(temptext,500,f);
                if(strcmp(temptext,"-999")==0){
                        return 0;
                }
                ;
                cout<<"\n|\t"<<temptext;

                return 1;
        }
};
main()
{
        reader r;
        if(r.f==NULL){
                exit(1);
        }
        r.read();
        cout<<"\n\n\t\t Enter Realtime? (y/n)\n\t\t $>";
        char c;
        cin>>c;
        if(c=='y'||c=='Y')
        {
                system("cls");
                while(true)
                {
                        Sleep(1000);
                        int flag=r.update();
                        if(flag==0){
                                cin.ignore();
                                system("cls");
                                cout<<"\n\n\t\t\tConnection Terminated";
                                break;
                        }
           }
        }

}
```

Client Module

Pathfinder Class:

The class to store protected and sensitive shared location of the server, the Server Location Path. This class is approximately 5 lines but is the original configuration of the server shared location. To change the server, the configuration of this class should be updated to the new server location.

Client Class:

This class is writer class, which accesses the server path from the pathfinder module and opens the log for writing messages. It asks for the username and appends it to every message to identify each user and their messages.

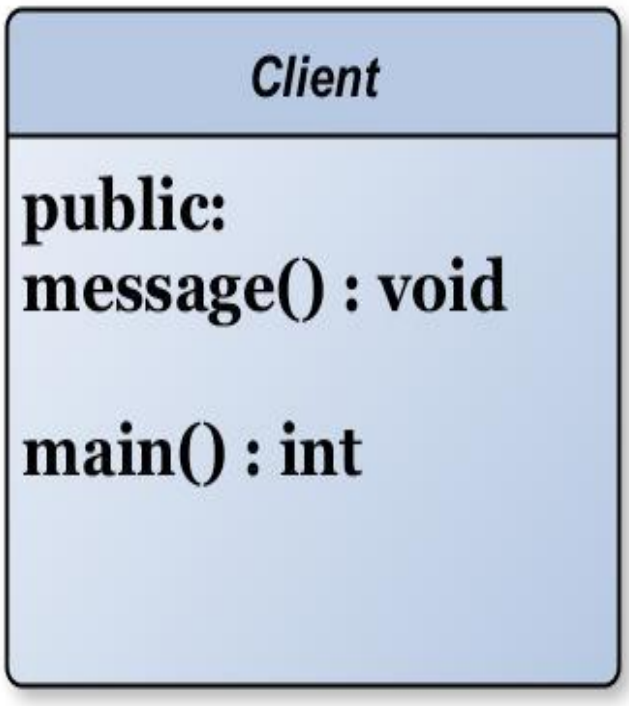
Start Setup:

A simple setup function, a replacement to the batch file to initiate the Reader Module and the Client Module.

How it works?

The Client module firstly asks for a username and keeps it, and then constantly waits for a message and once received it appends the username to the message and writes it to the log and then saves the file and then waits for the next message, this is a constant loop.

Class Diagram:



Code:

```
#include<iostream>
using namespace std;
#include<string>
#include <stdlib.h>
#include<fstream>
#include<windows.h>
#include"pathfinder.cpp"
class client:public pathfinder
{
	public:
		void message(char name[])
			{

				string message;
				cout<<"\n\n\t\t  $>";
				getline(cin,message);
				message = name+message+"\n";


				ofstream out(path,ofstream::app);
				out<<message;
				out.close();
			}

};
int main()
	{
		client c;
		char name[50];
		cout<<"\n\n\n\t\t\tEnter Nickname $>";
		cin>>name;
		strcat(name," : ");
		while(true)
		{
			system("cls");
			c.message(name);

		}
		return 0;
	}
```

Server Module

Pathfinder Class:

The class to store protected and sensitive shared location of the server, the Server Location Path. This class is approximately 5 lines but is the original configuration of the server shared location. To change the server, the configuration of this class should be updated to the new server location.

Server Side Class:

This class contains a series of functions including a menu to select each of the functions. The create() function creates the file log and sends the first message in the name of Admin. The message() function is there to send a special admin messages, to honour the original Bulletin Board System. The close() function closes the log by writing a kill code (independent), thus killing every client connected.

Server Setup:

A setup or initiation function which creates an object for the Server Side class and initiates the server loop and gets destroyed after the exit() function.

Server Loop:

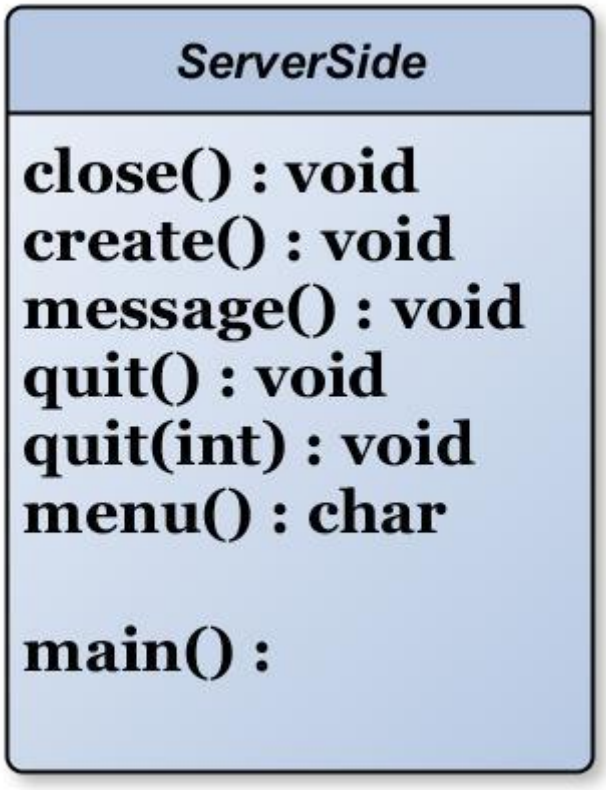
The server loop is a condition which constantly shows the menu to the administrator and waits for an input and then accordingly invokes the server side functions.

How it works?

The server setup creates the object for the ServerSide Class and starts the loop which waits for an input from the selection menu and performs accordingly.

***Kill Code: -999**

Class Diagram:



Code:

```
#include<iostream>
using namespace std;
#include<string.h>
#include <stdlib.h>
#include<stdio.h>
#include<windows.h>
#include"pathfinder.cpp"
class ServerSide:pathfinder
{

        public:
                void close()
                {
                        FILE *f=fopen(path,"a+");
                        fprintf(f,"\n-999");
                        fclose(f);
                }
                void create()
```

```
{
	FILE *f=fopen(path,"w");
	fprintf(f,"\t\t\tConnection Succesfull\n");
	fprintf(f,"Admin says Hello\n");
	fclose(f);


}
void message()
{
	FILE *f=fopen(path,"a+");
	char message[100];
	cout<<"\n\n\t\t Enter Message $>";
	cin>>message;
	fprintf(f,"Administrator : %s",message);
	fclose(f);
}
void quit()
{
	FILE *f=fopen(path,"w");
	fprintf(f,"\n-999");
	fclose(f);

}
char menu()
{
	cout<<"\n\n\n\t\t\t 1. Start.";
	cout<<"\n\t\t\t 2. Administrator message";
	cout<<"\n\t\t\t 3. Close server";
	cout<<"\n\t\t\t X. Exit";
	cout<<"\n\n\t\t\t Enter Choice $>";
	char choice;
	cin>>choice;
	return choice;
}
void quit(int x)
{
		system("cls");
		cout<<"\n\n\n\n\t\t\t --|Exiting|--";
		Sleep(3000);
		exit(1);
}

};
```

File Sharing Module

A fairly recent addition to the Internet Relay Chat which is dedicated to share files between one server and n client configuration. This is an independent module which is not yet integrated with the original IRC software.

Sender Function:

As the name suggests, it is a function which uploads files to the server, more specifically it created a log file with the filename and copies the target file to the shared Server Location. This function by default looks for the file to be sent in the application root folder.

Receiver Function:

A function which can be used by n number of clients to receive the file that is present in the server. This function by default saves the received file in the application root folder.

How it works?

Out of testing, this Application independently copies the file to be sent from the target location to the server shared location and logs the filename for the client's record.

The client can then access the log file and copy the file from the shared location the application root.

Class Diagram:

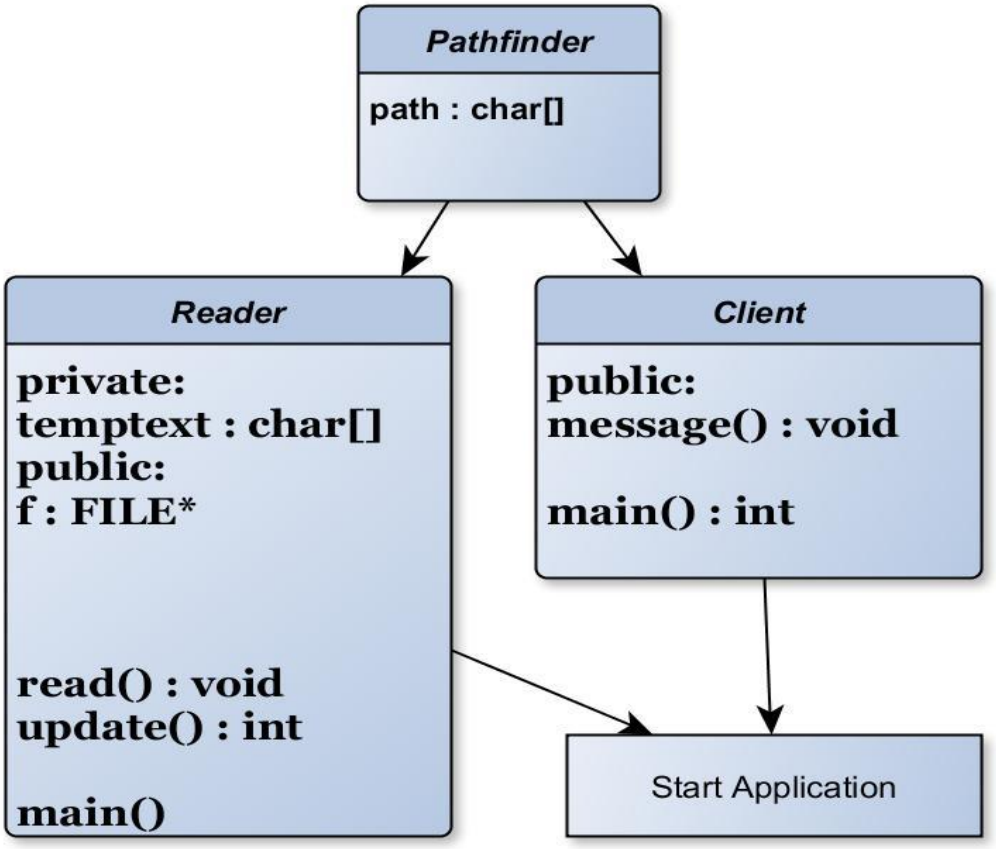
Code (Independent Application):

```
#include<iostream>
#include<string.h>
#include <stdlib.h>
#include<fstream>
#include<windows.h>
using namespace std;
char log[100] = "//MINI_CRASHXZ/Users/Mainak/Documents/Voodle/file/file.log";
class fileshare
{
public:
	void recieve()
	{
		ifstream x;
		x.open(log, ifstream::out);
		char filename[100];
		x>>filename;
		x.close();
		if (filename == NULL)
		{
			cout << "\n\n\n\n\t\t\tNo File Present!";
		}
		else
		{
			char command[500] = "copy ";

	strcat(command,"\\\\MINI_CRASHXZ\\Users\\Mainak\\Documents\\Voodle\\file\\");
			strcat(command,filename);
			strcat(command," ");
			strcat(command,filename);
			system(command);
		}
	}

	void send()
	{
		ofstream x;
		x.open(log,ofstream::in);
		char filename[100];
		cout << "\n\n\n\n\t\tEnter Filename $> ";
		cin >> filename;
		x<<filename;
		x.close();
		char command[200] = "copy ";
```

```
                strcat(command,filename);
                strcat(command," ");
                strcat(command,"\\\\MINI_CRASHXZ\\Users\\Mainak\\Documents\\Voodle\\file");
                strcat(command,"\\");
                strcat(command,filename);
                cout<<command;
                system(command);
        }
};
main()
{
        system("cls");
        cout << "\n\n\n\t\t\t1. Send\n\t\t\t2. Recieve\n\t\t\t$> ";
        fileshare sh;
        int choice;
        cin >> choice;
        switch (choice)
        {
        case 1:{
                        system("cls");
                        sh.send();
                        break;
        }
        case 2:{
                        system("cls");
                        sh.recieve();
                        break;
        }
        }
}
```

Client Side

Client + Reader Module(Integration):

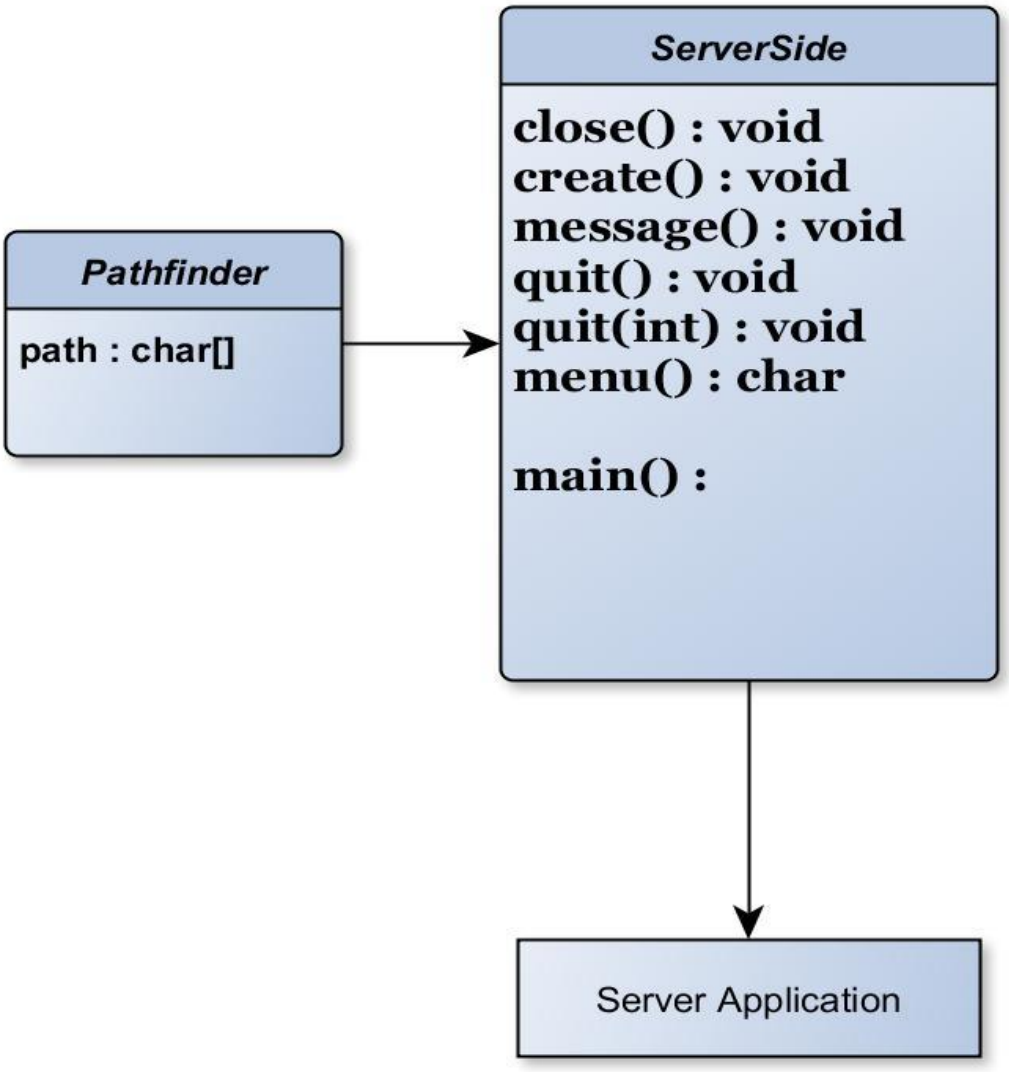


Start Application Code:

```
#include<windows.h>
main()
{
	system("start Client.exe");
	system("start Reader.exe");
}
```

Server Side

Server Side Design:

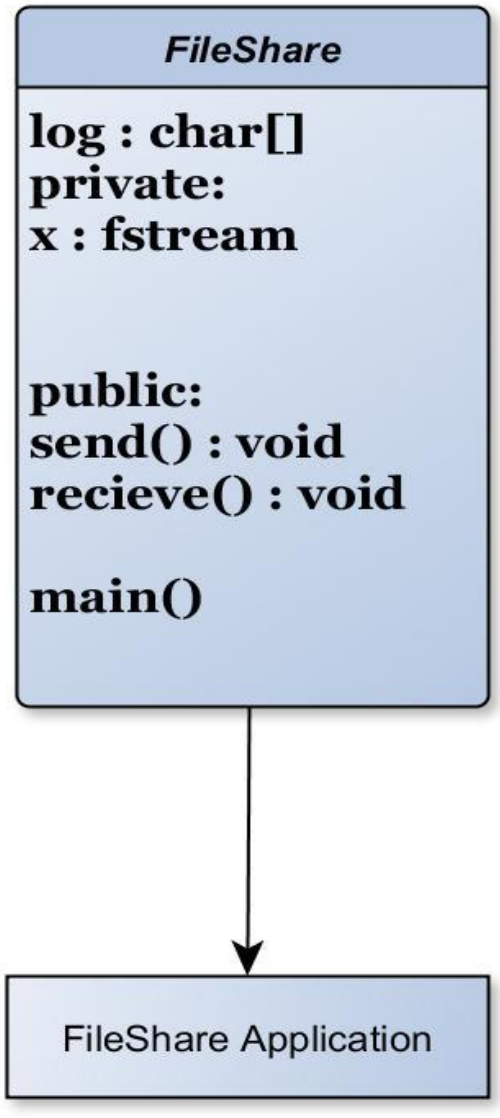


Server Application Code:

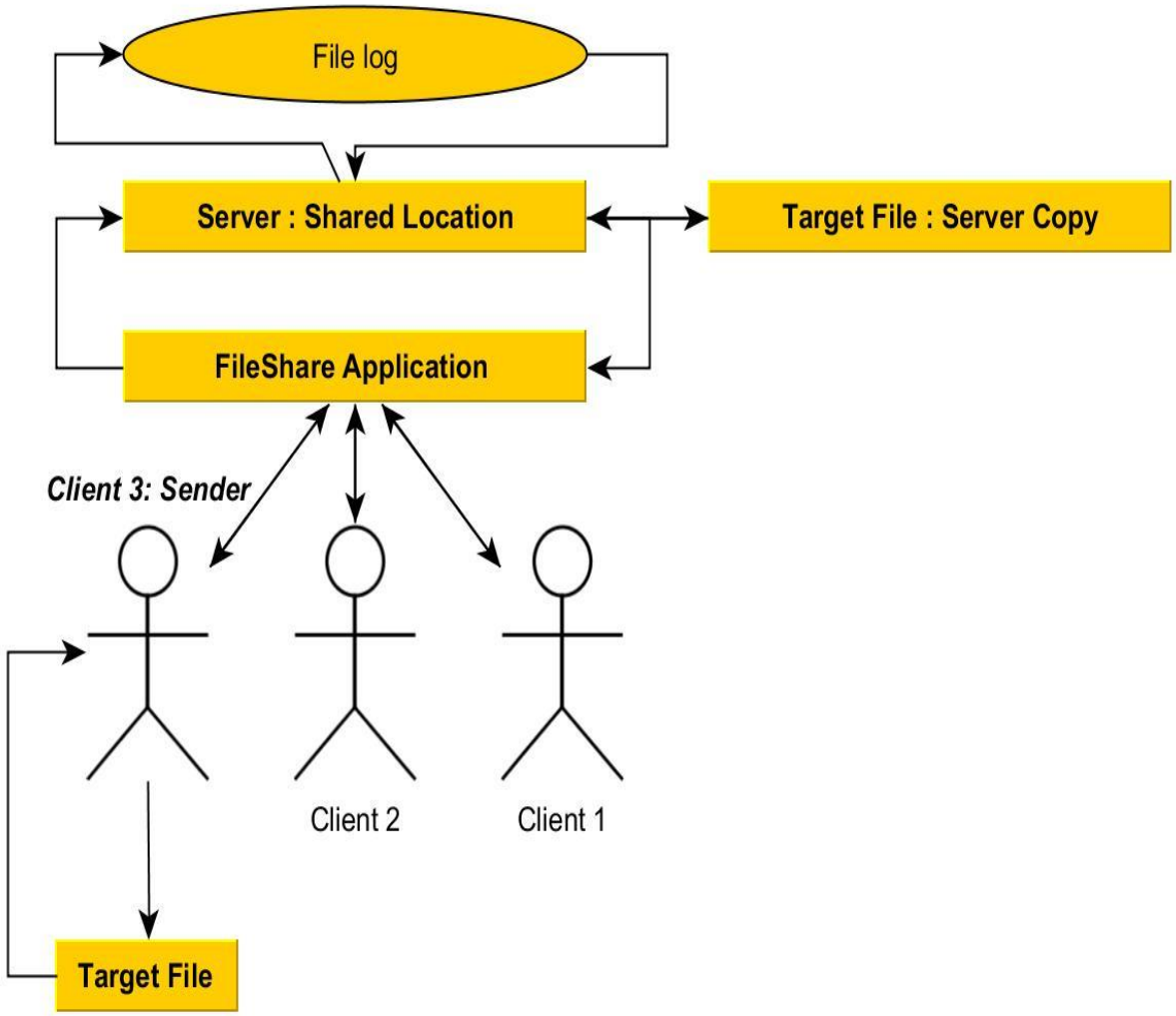
```
#include"Server.cpp"
main()
{
	ServerSide obj;
	while(true)
	{
		FILE *f=fopen("//MINI_CRASHXZ/Users/Mainak/Documents/Voodle/Voodle1.log","r");
		char choice = obj.menu();
		switch (choice)
		{
			case '1':
				{
					obj.create();
					system("cls");
					cout<<"\n\n\n\n\t\t\t ---|SERVER CREATED|---";
					Sleep(5000);
					break;
				}
				case '2':
				{
					obj.message();
					system("cls");
					cout<<"\n\n\n\n\t\t\t --|Message Sent|--";
					Sleep(5000);
					break;
				}
				case '3':
				{
					obj.close();
					system("cls");
					cout<<"\n\n\n\n\t\t\t --|Terminating Connection|--";
					Sleep(5000);
					obj.quit();
					break;
				}
				case 'X':
				{
					obj.quit(1);
					break;
				}
			}
	}
}
```

File Share Design:

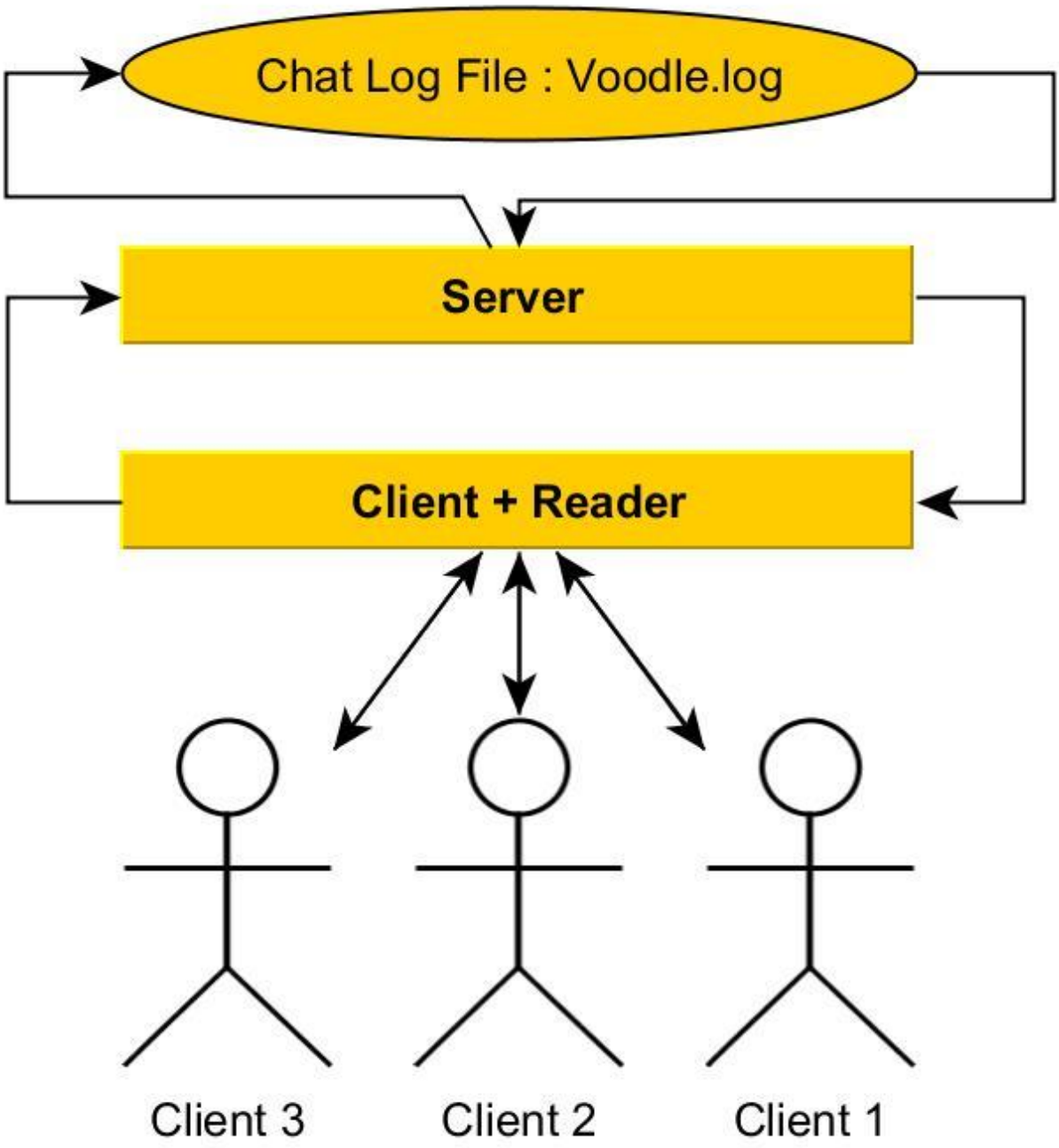
Application Design:



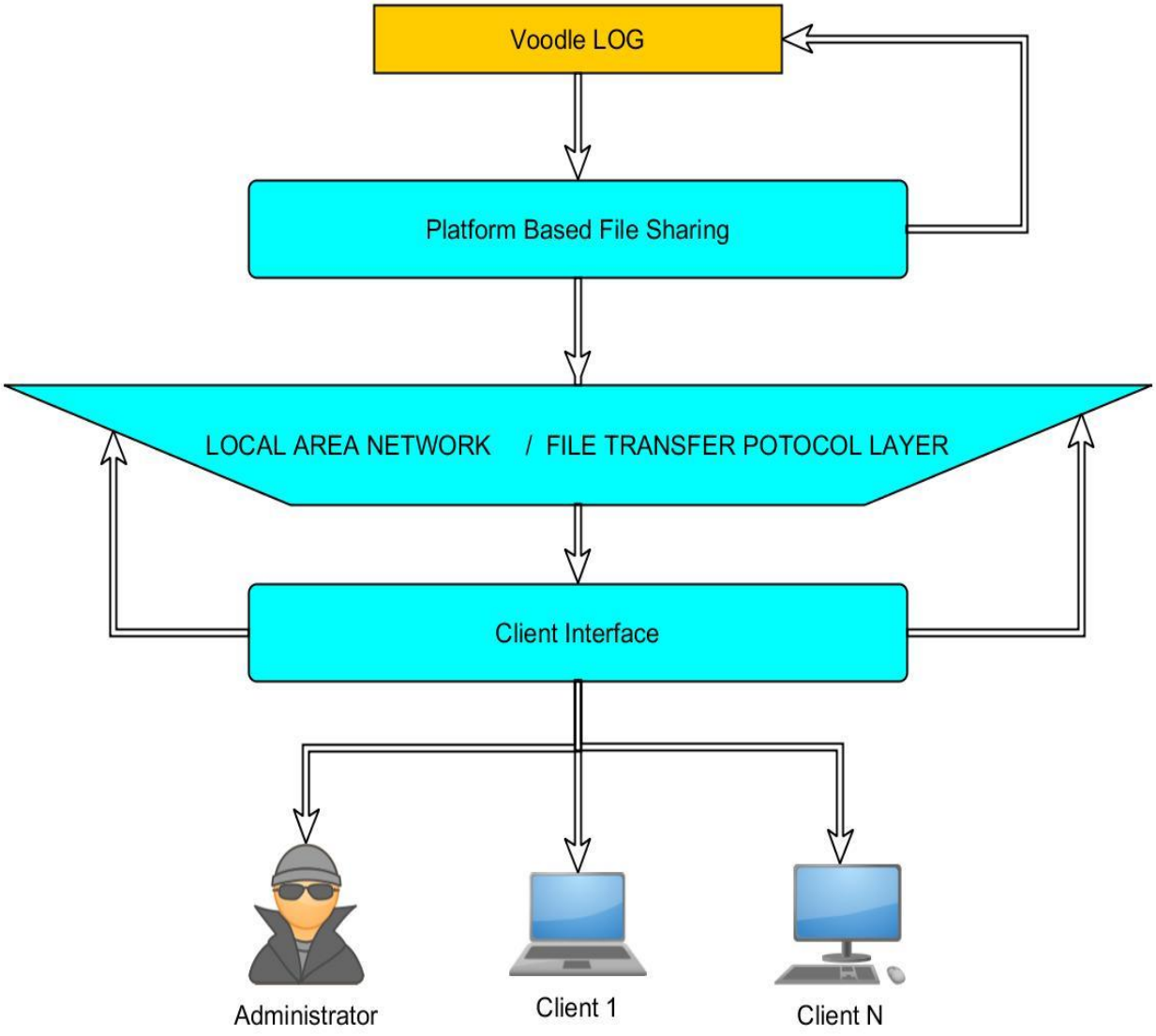
Server Design:



IRC Design:


Chat Log File : Voodle.log
Server
Client + Reader
Client 3
Client 2
Client 1

How It all Works?



Conclusion

There are a lot of Internet Relay Chat Clients available but a one with about a mere 400 slick lines of code is nearly impossible due to dynamic platform variations. The power of Object Oriented programming with the current platform file sharing modules overcomes those barriers and provides us with a console based IRC software that is exactly **4 Megabytes!** All programs should be written with modularity in mind. The code we Write should be reusable. Any type of code once written, by anyone, shouldn't be rewritten, unless you can write it better and faster!

The End

--------!--------