# Assignment Description

Create a client and a server program such that the server computes and sends back to the client the long name of any U.S. state given the two-letter abbreviation. The protocol is binary, and is as follows:

(1) Client connects and sends the two-letter state abbreviation in binary form.

(2) Server reads the character string and sends back a single character string containing the long name of the state, also in binary form. If an invalid abbreviation is sent, the server should respond with an appropriate error response.

(3) The server closes the connection to the client.

The server will be a single-threaded server that serializes service to its clients.

The server will stay in a loop in which it accepts a connection, reads the character data sent through the connection, and sends back the character response through the same connection. After that, the server closes the connection and goes back to wait for another connection.

The client should ask the user for a state abbreviation, read the user input, open a connection to the server, and send the data. It should then read the reply and print it (the long name of the state) to the screen.

Test your server and client on your own machine by using two different terminals.

You should also, test your client against another student’s server code, and then test your server code against another student’s client.

Submit your source code along with three screen shots of both the server and client demonstrating the application works correctly for valid and invalid abbreviations.

# 1 Readme Documentation

This program contains a client and server application. The client application will prompt for a 2-letter state abbreviation and server IP address. Then, it will connect to the server and send the abbreviation to the server.

The server application will connect to a client application. Then, it will receive the abbreviation and find the long name for the abbreviation. Then, it will send back the long name and close the connection. The server will run indefinitely until manually closed.

The client will receive the long name for the abbreviation, and then close the connection and exit.

# 2 Flowchart Screen Shots

# 3 UML and Use Case Diagrams

# 4 Source Code of All files (.h, .cpp)

Client application:

#include *<iostream>*

#include *<sys/socket.h>*

#include *<sys/types.h>*

#include *<netdb.h>*

#include *<cstring>*

#include *<unistd.h>*

#include *<array>*

#include *<cstring>*

#include *<algorithm>*

#include *<arpa/inet.h>*

**using** **namespace** **std**;

int main(){

*// Array of state abbreviations*

array<string, 50> stateAbbreviations = {

"AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"

};

*// Getting abbreviation*

string abbreviation;

cout << "Please enter a state abbreviation: ";

bool valid = false;

**while**(!valid){

**try**{

*// Get input*

cin >> abbreviation;

**if**(!cin) **throw**(runtime\_error("Please enter a valid 2-letter state abbreviation"));

*// Convert abbreviation to uppercase*

transform(abbreviation.begin(), abbreviation.end(),abbreviation.begin(), ::toupper);

*// Checking against list of abbreviations*

**for**(int i = 0; i < 50; i++){

**if**(abbreviation == stateAbbreviations[i]){

valid = true;

}

}

*// If invalid*

**if**(!valid) **throw**(runtime\_error("Please enter a valid 2-letter state abbreviation"));

}**catch**(**const** exception& e){

cerr << e.what() << endl;

cin.clear();

cin.ignore(10000, '\n');

}

}

*// Client code*

**try**{

*// Creating new socket*

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

**if**(sock < 0){

**throw**(runtime\_error("Failure to create socket."));

}

*// Building server address*

sockaddr\_in serverAddress;

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8080);

*// Get IP address for server*

valid = false;

**while**(!valid){

string address;

cout << "Please enter the server IPv4 address. Enter 127.0.0.1 for localhost: ";

cin >> address;

*// Throws error if fails to set server IPv4 address*

**if**(inet\_pton(AF\_INET, address.c\_str(), &serverAddress.sin\_addr) <= 0){

cerr << "Invalid IP address. Please try again." << endl;

**continue**;

}

valid = true;

}

*// Attempt to connect to server*

int connectionStatus = connect(sock, (**struct** **sockaddr**\*)&serverAddress, **sizeof**(serverAddress));

**if**(connectionStatus < 0){

**throw**(runtime\_error("Failure to connect."));

}

*// Convert abbreviation to cstring and send to server*

**const** char\* message = abbreviation.c\_str();

int messageSent = send(sock, message, abbreviation.size(), 0);

**if**(messageSent < 0){

**throw**(runtime\_error("Message failed to send."));

}

*// Recieving message from server*

char buffer[1024] = {0};

int bytesRead = recv(sock, buffer, **sizeof**(buffer), 0);

**if**(bytesRead < 0){

**throw**(runtime\_error("Error receiving message."));

}

**if**(bytesRead == 0){

**throw**(runtime\_error("No message received."));

}

*// Outputting message*

cout << "Message from server: " << endl << buffer;

*// Closing connection and exiting*

close(sock);

}**catch**(**const** exception& e){

cerr << e.what();

**return** -1;

}

**return** 0;

}

Server application:

#include *<iostream>*

#include *<sys/socket.h>*

#include *<sys/types.h>*

#include *<netdb.h>*

#include *<unistd.h>*

#include *<cstring>*

#include *<string>*

#include *<map>*

**using** **namespace** **std**;

*// Map of abbreviations to long names*

map<string, string> stateMap{

{"AL", "Alabama"},

{"AK", "Alaska"},

{"AZ", "Arizona"},

{"AR", "Arkansas"},

{"CA", "California"},

{"CO", "Colorado"},

{"CT", "Connecticut"},

{"DE", "Delaware"},

{"FL", "Florida"},

{"GA", "Georgia"},

{"HI", "Hawaii"},

{"ID", "Idaho"},

{"IL", "Illinois"},

{"IN", "Indiana"},

{"IA", "Iowa"},

{"KS", "Kansas"},

{"KY", "Kentucky"},

{"LA", "Louisiana"},

{"ME", "Maine"},

{"MD", "Maryland"},

{"MA", "Massachusetts"},

{"MI", "Michigan"},

{"MN", "Minnesota"},

{"MS", "Mississippi"},

{"MO", "Missouri"},

{"MT", "Montana"},

{"NE", "Nebraska"},

{"NV", "Nevada"},

{"NH", "New Hampshire"},

{"NJ", "New Jersey"},

{"NM", "New Mexico"},

{"NY", "New York"},

{"NC", "North Carolina"},

{"ND", "North Dakota"},

{"OH", "Ohio"},

{"OK", "Oklahoma"},

{"OR", "Oregon"},

{"PA", "Pennsylvania"},

{"RI", "Rhode Island"},

{"SC", "South Carolina"},

{"SD", "South Dakota"},

{"TN", "Tennessee"},

{"TX", "Texas"},

{"UT", "Utah"},

{"VT", "Vermont"},

{"VA", "Virginia"},

{"WA", "Washington"},

{"WV", "West Virginia"},

{"WI", "Wisconsin"},

{"WY", "Wyoming"}

};

int main(){

*// Create socket for server*

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

*// Create address for socket*

sockaddr\_in serverAddress;

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8080);

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

*// Bind address to socket*

bind(sock, (**struct** **sockaddr**\*)&serverAddress, **sizeof**(serverAddress));

*// Wait for connection on socket*

listen(sock, 5);

*// Runs indefinitely*

**while**(true){

*// Accepts new connection*

int clientSocket = accept(sock, **nullptr**, **nullptr**);

*// Receive message from client*

char buffer[1024] = {0};

int bytesReceived = recv(clientSocket, buffer, **sizeof**(buffer), 0);

**if**(bytesReceived < 0){

cerr << "Message not received" << endl;

}

*// Outputting message from client*

cout << "Message from client: " << buffer << endl;

string abbreviationString(buffer);

string longName;

*// Finding long name for abbreviation*

**for**(**auto** itr = stateMap.begin(); itr != stateMap.end(); itr++){

**if**(itr->first == buffer){

longName = itr->second;

**break**;

}

**else**{

longName = "Abbreviation not found!";

}

}

string message = "Here is your abbreviation in long form: " + longName;

*// Converting message to cstring and sending to client*

int messageSent = send(clientSocket, message.c\_str(), message.length(), 0);

**if**(messageSent < 0){

cerr << "Message not sent" << endl;

}

*// Closing socket*

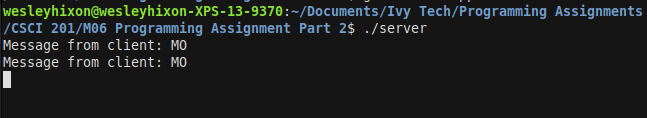
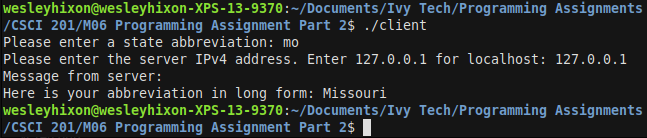
close(clientSocket);

}

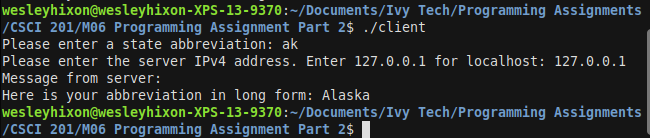
}

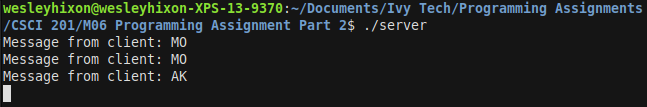
# 5 Three Use Case Screen Shots

Run 1:



Run 2:





Run 3:

