ECE 4122/6122 Lab 5: TCP Sockets

(100 pts)

Category: sockets
Due: Tuesday November 19th, 2024 by 11:59 PM

Objective:

In this assignment, you will use C++ to simulate the movement of a robot on a 2D screen using SFML for graphics and UDP sockets for communication. The robot's location will be updated by sending direction data over a UDP socket from a client to a server. The server will render the updated direction for the robot on the SFML window.

Overview:

- **Server (SFML):** Renders a robot icon on a 2D graphics window. It listens for incoming UDP messages from a client and updates the robot's position based on the received direction data.
- **Client:** Allows the user to control the motion of the robot. It sends the motion commands to the to the server using UDP sockets.

Requirements:

UDP Communication: Implement UDP socket communication between the client and server.

SFML Setup: Use SFML to create a 2D grid representing the robot's environment on the server.

Robot Movement: Update the robot's position based on the data received from the client.

User Interaction: Allow the client to input the robot's direction of motion, which will be sent via UDP packet to the server.

Assignment Steps:

1. Set Up Server (SFML): *(60 points)*
    a. Initialize a blank black window SFML window (e.g., 800x600).
    b. Set up a UDP socket to listen for incoming packets. The port number is set using a command line argument.
    c. When the initial packet is received, a simple shape (e.g., a circle) representing the robot is drawn in the center (3x3pixels).
    d. **10 pts extra credit** for using an image for the robot and rotating the image as the robot's direction changes
    e. On receiving a packet with new commands, update the robot's position on the SFML window.
    f. The robots initial speed is 3 pixels/second.
    g. The server application ends when the window is closed.
    h. The server must inform the client that the server is about to close.
    i. If the client sends a disconnect message the robot icon disappears.

2. Set Up Client: *(40 points)*
    a. Implement a UDP socket to send UDP packets to the server.
    b. Prompt the user to input a direction of motion.
    c. The client can the move the robot up, down, left and right by pressing the “w”, “s”, “a”, and “d” keys.
    d. Each time one of the keys is pressed a message is sent to the server to change the robot direction of motion.
    e. The client can change the speed of the robot by entering “g” for faster and “h” for slower.
    f. You are free implement the message format however you choose.
    g. Entering a “q” closes the client. The client needs to send a message to the server that it is disconnecting.

Turn-In Instructions

If you upload a video of your program showing the required functionality, then your code does not need to compile on pace-ice.

Zip up your server file(s) into **Lab5Server.zip** and your client files into **Lab5Client.zip** upload these zip files on the assignment section of Canvas.

Grading Rubric:

If a student’s program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA’s will look through your code for other elements needed to meet the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE! Unless a video is uploaded and the requirement is waved.
Does Not Match Output	Up to 90%	The code compiles but does not produce correct outputs.
Clear Self-Documenting Coding Styles	Up to 25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score – 0.5 * H	H = number of hours (ceiling function) passed deadline

Appendix A: Coding Standards

***Indentation*:**

When using *if/for/while* statements, make sure you indent **4** spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird).

This applies for functions and member functions as well!

The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function represents. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: your name
Class: ECE4122 or ECE6122 (section)
Last Date Modified: date

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.