Programming assignment 1: Application Layer

CSEE 4119: Computer Networks, Spring 2012

Due Tuesday, February 28th (by noon)

Academic Honesty Policy

You are permitted and encouraged to help each other through Piazza's web board, in order to discuss and understand concepts learned in class or go deeper in a particular subject.
HOWEVER, you may NOT share source code or hardcopies of source code, or answers to written assignments. Refrain from activities or the sharing materials that could cause your source code to APPEAR TO BE similar to another student's enrolled in this or previous years. Cheating will be dealt with severely. Cheaters will be punished. Source code and written answers should be yours and yours only. Do not cheat.

1 Introduction

In this assignment, you will work on the application layer, over a TCP connection. The goal is to implement a collaborative shape drawer. You will be guided step by step. The first steps are quite easy, but the last one demands more initiative.

We provide you with a User Interface, so you will only have to implement the networking part. The first step is to locally connect the graphic interface to be able to draw shapes. Then you will have to connect to a given broadcast server to display other students shapes too. Finally you will add a proxy between your client and the server to be able to filter messages in both directions.

You have about 4 weeks to do this assignment but we advise you to start early because the assignment is long and some parts are not easy.

Sections **3 and 4** explain how to organize and submit your project, **read it carefully** as we will grade your assignments with programs of our own. So **if you don't respect this, we won't be able to grade your submission**! **No late day allowed**!

2 Due Dates and Program Submission

This programming assignment is due on **February 28th**. **No lateness is accepted**.

Before submitting, make sure your program compiles and runs properly on a CS or EE machine, because you will be graded on those machines, and code that doesn't compile will obviously be considered as incorrect and will obtain the grade of 0.

Your program has to follow the architecture described in part 3.

To submit the assignment, you will have to rename the yourUNI folder (from the starting code in the zip file) with your UNI (e.g. ml3302). Then **from the CLIC (or EE) machines**, go in the PA_1 folder, and type «make submit». Be careful, this will submit as the person longed in the machine, so you have to submit being logged as you! You can do that as many times as you need to. Only your last submission will remain. **Before submitting, make sure your program compiles by typing the make command in the relevant folders**!

The grading will be done with a script that will test many cases to make sure your program behaves as described.

3 Getting started

The makefiles:

If you don't know what a makefile is, you can search for it on Google. Roughly it is a tool to automate some repeating tasks such as compiling a project, in a UNIX environment. To start working on the project you will have to unzip the archive you downloaded from the website. In it you will find a makefile and a folder called yourUni that you should rename with your uni (e.g.: Laurent would have to rename it into lc2817).

This is the file structure you'll have in the end (be careful not to change the names):

step_1 -> the files for step 1's client
step_2 -> client -> the files for step 2's client
-> client_unreliable -> the files for step 2's client
step_3 -> client -> the files for step 3's client
-> proxy -> the files for step 3's proxy
step_4 -> client -> the files for step 4's client
-> proxy -> the files for step 4's proxy

We provide you with a makefile in all of these folders to compile and launch your code. However, this is only for when you are working in a **UNIX** environment, like the CLIC lab. These are the commands we will use to compile, run, and grade your program later on, so if your programs work with these makefiles in the CLIC lab, they will be compiled correctly during the grading. For those of you who would like to do this assignment on **Windows** from your personal computers, you will launch your programs from Eclipse (or, in the case of the GUI, by double-clicking on the gui.jar file, as we describe below). However, we recommend that once you have tested all your code locally, that you test it on the CLIC/EE machines, with the make commands.

For those of you that want to program in C, please let us know, we will provide a makefile and some guidelines for you.

The GUI:

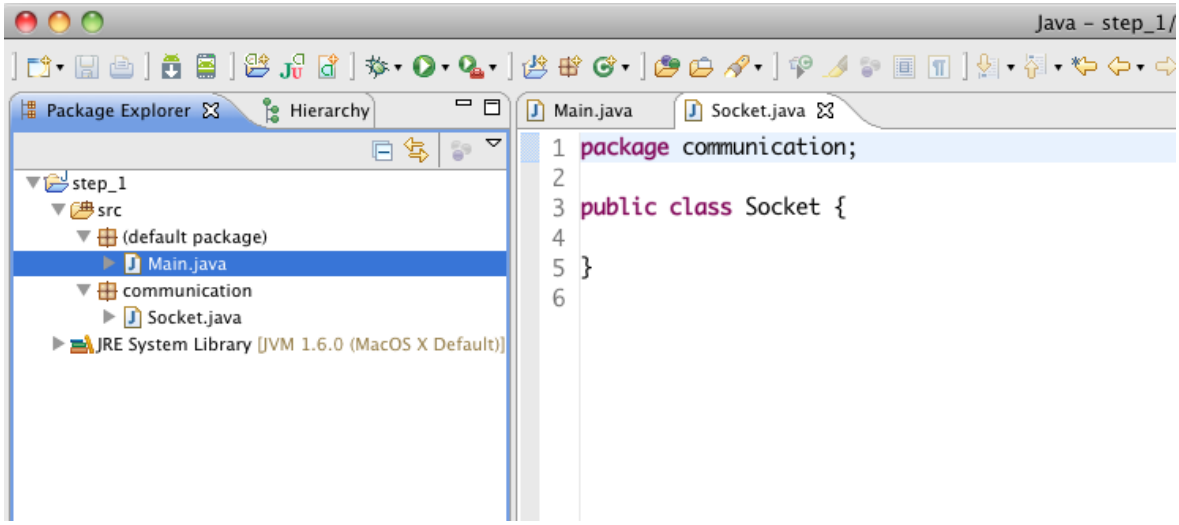
To launch the Graphic User Interface (GUI), double click on the file gui.jar in the root of your folder.

If the GUI does not show up, contact us.

Eclipse:

Then launch eclipse (you can download eclipse with java here: http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/indigosr1) and start building your program by creating a new project. Read the following guideline about packages, and **put the Main class inside the default package by not providing any package name while creating the project**.

To do a step – let's say step1– create a new project in Eclipse from existing source and select the empty folder step_1 extracted from the zip. Later on you can create new packages if you want. The only constraints are that your main file has to be named Main.java, must be in the default package, and must be in the root of the correct folder for whatever part of this assignment you are working on: For example, it must be in the step_1 folder for Step 1, and in the client and client_unreliable folders for the 2 parts of Step 2 (so, two main classes!).

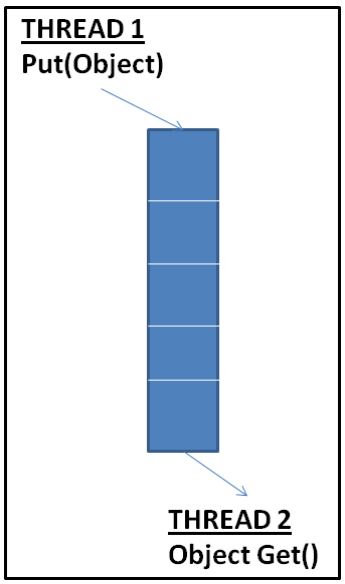


Remote access to the lab:

For those of you who are note familiar with the techniques to remotely access to a computer, look up ssh and scp on Google and come to our office hours if you can't succeed in making it work.

4 Coding tips

The bounded buffer:



We provide you with an implementation of a "bounded buffer". It is basically a buffer implemented as an array, where you can put and remove objects as in a FIFO list. This buffer also allows you to safely communicate between your different threads. You can learn more about it here if you want: http://en.wikipedia.org/wiki/Producer-consumer_problem.

The bounded buffer has 3 methods:

public void put(Object value) that adds the Object value to the buffer

public Object get() that gets the oldest Object of the buffer, removes it, and returns it.

public int getSize() that returns the number of elements in the buffer.

The function get() is blocking the program, meaning that when you call it, the program will wait to get an object and then continue. In particular, if it is called when there are no objects in the buffer, it will block the current thread until an object is placed into the buffer (eg, by another thread). Be careful to not get stuck!

The bounded buffer comes with an example of how to use it with several threads that try to access the same buffer. Basically the example is a program that launches several threads that try to randomly put and remove objects from the buffer. Study the code – it's going to be helpful!
This is an illustration of the Producer-Consumer problem, described in the Wikipedia article above.

This example is here to teach you how to launch threads and how to use the bounded buffer.

Threads:

When you launch a program, your computer is creating what is called a process. A process is an instance of the program (think about the program as a class and the process as an instance of this class). This process can divide itself into threads to be able to execute parallel tasks such as receiving some information and sending some information at the same time.

You will need to use threads to parallelize some actions in your program.

Sockets:

All the information you need about sockets are available in your book (in particular, see section 2.7). It's also always a good idea to look up tutorials and documentation for APIs you are using online.

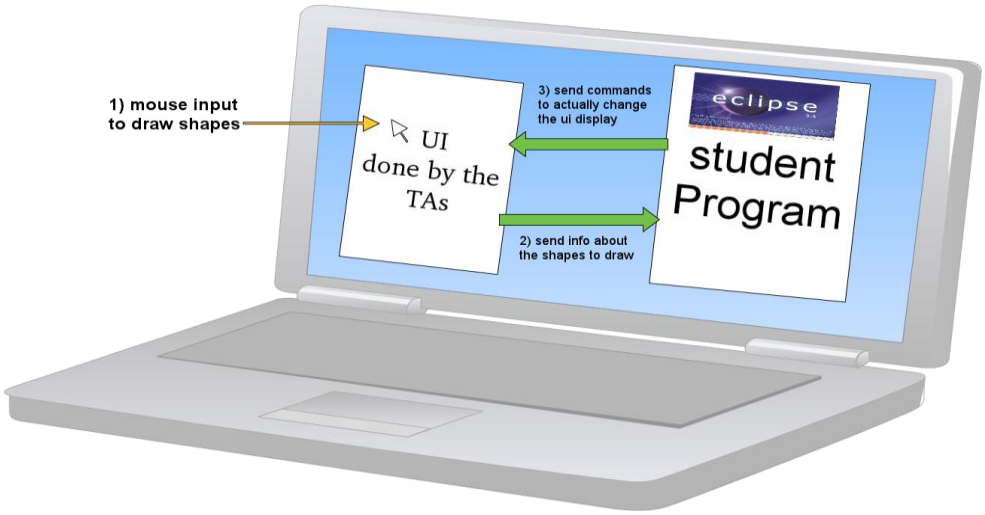
About using code from the internet:

You can use limited sample of code from the internet as long as you mention clearly the source of the code. If you fail to do so and we detect that you used some code from the internet, it will be considered as a plagiarism act.

5 First step: draw shapes (25%)

For this first step, you will have to create a client that connects to the user interface, gets its messages, and resend it to the interface in another port so that the shape is actually displayed.

Here is a little schema of this first step:



Launch the UI: To launch the UI, simply double-click on the gui.jar file. If you are in a UNIX environment, you can also launch the UI by navigating ("cd") to the UI folder, and typing «*make*».

Receive the messages:
In order to exchange messages with the UI, you will use sockets. You need to:

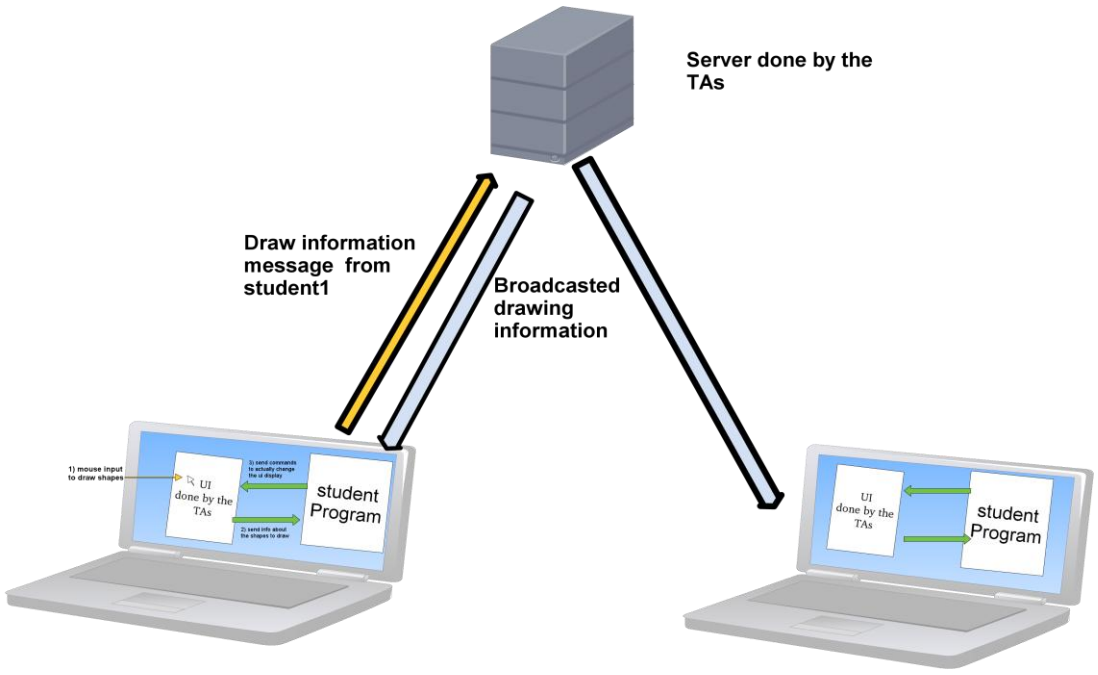
- First, use a socket to connect to the localhost on port 1450 to receive the messages from the UI.
- Then, also connect to the port 1451 (still on localhost) in another thread to forward the messages to the UI. In this step, just forward the string you receive without changing it (if you'd like to see the messages you are forwarding, you can print them to the console). You should use the bounded buffer we provided to safely communicate between the 2 threads.
- The format of the messages is «*Shape x y Color*» (e.g «*Square 123 34 Red*»)

Congratulations! You just finished the first step ;-). At this point, you should be able to display shapes on the UI. If you have any trouble, you can debug using telnet. In a UNIX terminal type **telnet** server_name port (e.g. telnet localhost 80). You can then send and receive messages. To learn more about telnet: http://en.wikipedia.org/wiki/Telnet.

6 Second step: collaborative drawing (35%)

Basic connection:

In order to make your drawing program collaborative, you will connect to a broadcast server. When you send a message to this server, it broadcasts it to every connected person, you included. Be careful not to display your own messages 2 times.



To connect the broadcast server, create a new socket to the IP 128.59.15.27 and port 1452 (think about how you would like to design this connection from a threading perspective…should it be from a new thread, or from an existing one?). You have to forward all the messages coming from your UI to this server, and display others' shapes in addition to yours!

However, we can't just forward messages coming from the UI: we have to add some information. For instance, you want to know who is sending the message. You also have a spot to add the information you want. **This is the format of the message** (it is a string, on one line):

UNI:your_uni Shape:the_shape X:x Y:y Color:color AdditionalInfo:whatever_you_want

The server will regularly send messages, so you should see it being displayed. Well done, you now have a collaborative shape drawer :-).

Let's make it funny:

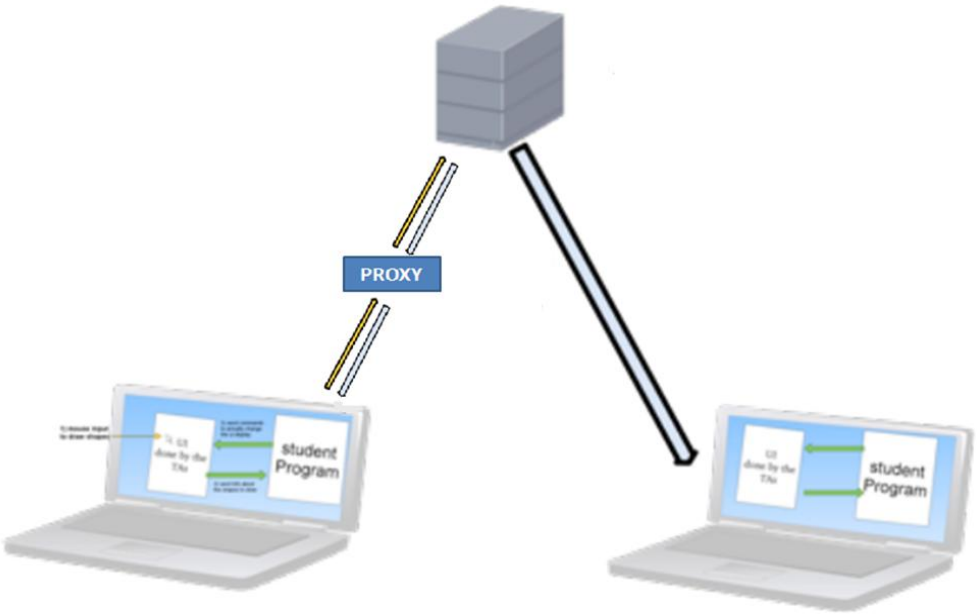
If you connect to the server on port 1453, your messages are still broadcasted, but there is a probability that your message is dropped BEFORE being broadcasted. The server is thus unreliable!

You have to make sure that your message is actually broadcasted, and resend it if you have not received it after 300 milliseconds. Note that if you receive a message you have just sent, it has been successfully broadcasted (it is broadcasted to nobody or everybody). Also, if you receive a message a second time, ignore it.

This code goes in the client_unreliable directory (see section 3).

7 Third step: build a (local) proxy (30%)

In addition to your client (that nearly doesn't change for this step) you will build a proxy that goes between your client and the broadcast server. You will thus have to have both your client and your proxy running on your computer for this to work.



Minor client modifications:

Instead of hard coding the IP and port your client is connecting (for the remote server), you have to pass it as command line arguments. When in the client directory, typing in **«make localhost 1337»** has to make your client connect to localhost on port 1337 instead of the broadcast server. Note that this 'make' command results in "localhost" and "1337" being passed into your main method as command-line arguments. The first argument should be "localhost" or an IP address, while the second should be a port number.

Build your proxy:

Your proxy has a client part, that connects to the RELIABLE server (IP 128.59.15.27 port 1452).

Your proxy has a server part, that waits for a connection on a port given in command line argument (like for the client: **«make 1337»**). You obviously have to start your proxy BEFORE you start your client, because the server part has to be running when your client tries to connect.

For this first proxy part, you just have to forward the messages of your client to the server, and in the other way around.

8 Last step: filter the messages (10%)

For this last step, you're going to have a bit more freedom. The goal is to filter messages that arrive to your proxy. You will maintain a list of the UNIs you want to filter, and your proxy has to drop all the messages coming from these UNIs.

To build this list, your CLIENT (eg, the program connecting to the proxy) has to accept the following command line arguments:

[-a:UNI1,UNI2,UNI3] to add the UNIs in the list (with commas,without spaces, without the []).

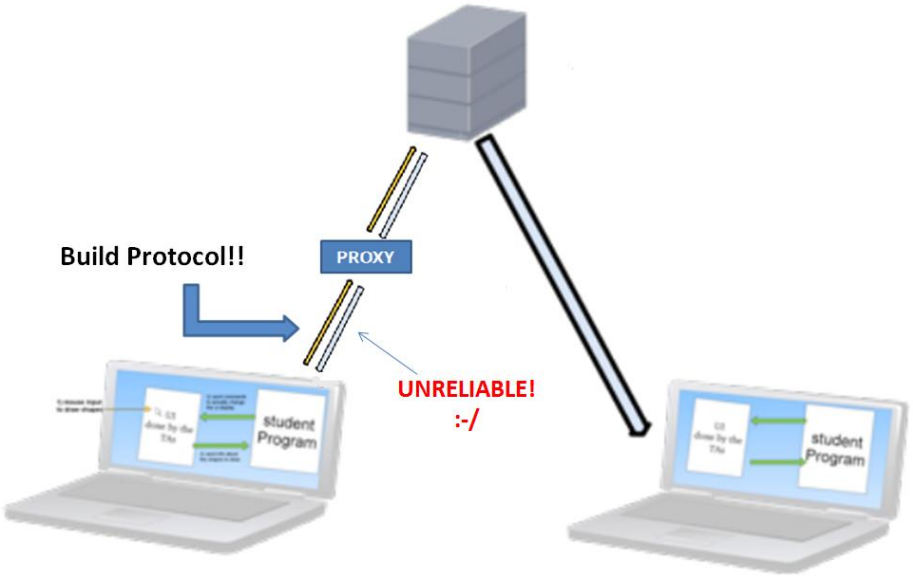
[-r:UNI4,UNI5] to remove UNIs from the list (with commas,without spaces, without the []).

Note that all the following are valid ways to call your client:

```
make localhost 1337
make localhost 1337 -a:UNI1,UNI5
make localhost 1337 -r:UNI4,UNI6,UNI7
make localhost 1337 -a:UNI1,UNI2 -r:UNI3
make localhost 1337 -r:UNI3 -a:UNI1,UNI5
```

Your goal is to create a protocol between your client and your proxy to transmit the data to update the list. You are free to build whatever protocol you want. However, you have to consider that the connection between your client and your proxy is unreliable, causing some packets to be lost (while grading, we will add a program ours between your client and your proxy to randomly drop packets). Note that packets can be dropped in BOTH directions (client -> proxy, and proxy -> client). You have to make sure that your protocol is built in a way that the UNIs to add or remove are reliably passed to the proxy, despite this unreliable connection.

Note that we can kill and relaunch your client while your proxy is still running. This is why you have to implement a function to add AND to remove UNIs from list of banned UNIs!



Summary of 'make' commands

- <UI Folder> -- "make": Launches UI (can also launch UI by double-clicking gui.jar)
- step_1 -- "make": Launches step 1's client
- step_2\client -- "make": Launches step 2's client, connected to reliable broadcast server.
- step_2\client_unreliable -- "make": Launches step 2's client, to unreliable server
- step_3\client -- "make localhost 1337": Launches client, connecting to proxy at provided host (eg, "localhost") and port (eg, "1337").
- step_3\proxy -- "make 1337": Launches proxy, listening for connections (from the client) on the provided port (eg, "1337")
- step_4\client – "make localhost 1337 -a:uni1,uni2 -r:uni3,uni4": Launches client, connecting to proxy at provided host (eg, "localhost") and port (eg, "1337"), and adding/removing the provided UNIs to the proxy's filter list.
- step_4\proxy -- "make 1337": Launches proxy, listening for connections (from the client) on the provided port (eg, "1337")
- "make submit" (from root folder, named <youUni>): submits your code

Remember: It is very important that your programs work properly, with the make commands outlined here, so that our automated tests can grade them properly!!!

Also recall that these commands only work from a UNIX environment.

Conclusion

You built your own network application on top of TCP! We hope it helped you understand how internet applications works, and why it is great to have only the application layer to take care of!

Feel free to ask all the questions you have :-).