CSE/IT 353 Project #2
Star Network
GROUP will be published shortly after the MIDTERM
Due: Sunday, November 5th, 2017, 11:59pm

Project Goal

The aim of this project is to implement a simulator of a ***star*** network using TCP/IP sockets as its physical layer. You will instantiate node objects that will connect to a switch object via the local host TCP/IP interface, which will then allow inter-node communication. All data communication must go through the star central hub **switch.** You will design a MAC protocol (syntax/semantics) that manages the network traffic between the star nodes through the core hub, expecting variation of traffic types.

Project Description

Overview

You are to create an object-oriented program that will spawn multiple node objects and threads, as well as a switch object and thread. The switch will use a listening socket to allow nodes to connect to it, and spawn worker threads as needed to allow inter-node communication. The nodes will open a file, and send data to other nodes via the switch. The nodes will save all data that was sent to them in the form of output files.

Main Class

Your main class should instantiate a single switch, and a number of nodes between 2 and 255, which it should get from the command line arguments. It should also wait until all nodes are done sending data, before shutting down all the individual nodes, followed by the switch, and exiting cleanly.

Switch Class

The switch is a single class, which will be instantiated once. It should open a port to allow any number of nodes to connect to it. It can spawn as many threads as needed to perform its function. It should have a global frame buffer, which it can use to store frames before they have been forwarded. Your switch must be able to handle multiple nodes sending data to it at the same time! The switch should also have a switching table; in which it will keep track of which ports have which clients connected to them. Like a real switch operating at layer 2, the switch should examine the source address of the frame, and use that to populate its switching table. If it does not know where the destination node resides (for example, on a clean startup of the switch), it should flood the frame out every active port except the one that the frame came in on. After it learns where the nodes reside, however, it should only send the frame out the correct port (this is the difference between a hub or a “dumb switch” and a real “smart” switch).

Node Class

The nodes are similar to the nodes from Project 1 (utilize as much code as you can from your first project. The nodes should connect to the switch upon start-up, while being recorded in the switch table along with its (possibly) temporary TCP port assignment. It is important to note that the communication with the switch might not be through the same TCP port that it initially connected to! The switch is allowed to dynamically assign a new TCP port number to the client connection (much like a real life multi-threaded server would). The nodes should be robust enough to still synchronize with the switch, even if they are started before the switch is ready to accept connections.
The nodes are to be numbered, and you should design your code so that we can instantiate between 2 and 255 nodes easily. Every node should open and read an **input file**, which contains data that it should send across the network to other nodes (See Files section below). Every node should also create an output file which contains all the data that was sent to it by the other nodes. The nodes should send data to the other nodes via the switch. When the data is successfully received, the node that received it should send back an acknowledgment to the sender via control bits as specified in the frame format (see Frame Format section below).

Files

A script will be provided that will generate data files for the nodes to send. The files will be named node1.txt, node2.txt, node3.txt, etc. The nodes will open their files, and send the data to the other nodes.

The following is an example input file for node 1:
2: ABCDEFG
3: 1234567
5: This data will be sent to node 5.

In this example, node 1 will send `ABCDEFG` to node 2. You may assume that no node will want to send data to itself. The nodes are responsible for creating a file, named node#output.txt, which will contain all the data that was sent to it, and who sent that data. Using the previous example file, Node 2 will create `node2output.txt` which will have the line `1: ABCDEFG`.

Frame Format

You will have to design your own encapsulating frame for this project; it is not sufficient to just send the data across the network and rely on the port number to determine who it was from.
The following is a suggested frame format for this project:

[SRC][DST][SIZE/ACK] [data]
SRC: Source of the frame, 1 byte, 1-255
DEST: Destination of the frame, 1 byte, 1-255
SIZE/ACK:

In a data frame, this field has the size of the data in bytes, from 1-255. When size is 0, the data field is omitted, and this is treated as an ACK. data: The actual Data (1-255 bytes) You may modify the frame format as you see necessary, possibly additional fields), however, the preceding three fields are the bare minimum. You must also create a class for the frame, as this will be important for future projects. It is highly recommended that you have a method in the class that will convert the frame to bytes, to be sent across the socket. Moreover, you should have a constructor that will create a new frame from the bytes that were received from the socket.

Project Requirements

1. **Language** – You are required to implement the project in one of the following programming languages: Java, C#, or C++. I will make your groups based on the languages you used, so I would stick with that language.

2. **Environment** – Your program must compile and run on the TCC machines (login.nmt.edu). All projects will be graded on the TCC machines. If your project does not compile or run on this system, then your grade will suffer.

3. **Build Automation** – You are required to employ some sort of build automation for this project. Acceptable formats are GNU make for C++ or C#, or Apache Ant for Java. Look into your IDE, there's a good chance that it will create makefiles or build.xml files for you!

4. **Documentation** – You need to create a README file that includes the following:
- Names of all group members.
- How to compile and run your program.
- The names of all files in the project and a brief description of their purpose.

- Provide a checklist that includes which features are implemented, and which features are missing. The minimum required checklist for this project is below.
- A list of all known bugs. Documenting bugs will reduce the corresponding point deduction.

5. **Code Style** – Source code should be well-organized, follow accepted conventions for that language, and be well-documented with meaningful comments and variable names.

Assumptions

The following is a list of assumptions you are allowed to make in regards to your program:

1. You are not required to verify the format of the input files, since they should strictly follow the format defined above.
2. You are not going to have more than 255 nodes connected at any given time.
3. A single frame of data will not contain more than 255 bytes.
4. No node will attempt to send data to itself.

You are not required to provide a GUI of any kind. Output to the console is both acceptable and encouraged.

Feature Checklist

Include a specification of the frame format in your README. This should include the order of the fields, the byte sizes and acceptable ranges for the fields, and a short description of the function of the field.
Include the following checklist in your README. Each item should be marked as complete, missing, or partial. In the case of a partial status, make sure to add a description as to what works and what does not. The following is an example; it is expected that you will change the "Status/Description" column to indicate progress in your project.

Feature	Status/Description
Project Compiles and Builds without warnings or errors	Complete
Switch class	Complete
Switch has a frame buffer, and reads/writes appropriately	Complete
Switch allows multiple connections	Complete
Switch floods frame when it doesn't know the destination	Complete
Switch learns destinations, and doesn't forward packet to any port except the one required	Switch acts like a hub
Node class	Partially Complete – see below
Nodes instantiate, and open connection to the switch	Complete
Nodes open their input files, and send data to switch	Complete
Nodes open their output files, and save data that they received	Partial – They also save flooded frames

Extra Credit

There is an opportunity to implement extra features in the code in order to get a higher grade. The following is a list of features that can be implemented, as well as their point values if they work properly. NOTE: You cannot receive any extra credit if the above basic requirements are not met! This means that you can't implement all of the switch extra credit, and have non-working nodes!

Description	Point Value
Backbone Switching: Instantiate two switches, connect them to each other, and have the nodes connect to one or the other. Allow data to be sent between switches.	**30**
Star of Stars Switching: Instantiate three or more switches, and make one switch be a backbone switch for the other ones, effectively making a star of stars topology.	**30**
Frame Priority: Add a "high priority" flag to the frame format, and make the switch send that frame first, regardless of its position in the frame buffer.	**10**

You are allowed to make differing instances of the project to showcase all of these features (e.g. you don't need one set of code that handles all of these features).
If you think of more features, you can receive more extra credit for them, but the ideas have to be submitted for approval, in writing, to the class TA before any extra credit will be given. Note: You will have to demonstrate your implementations to the TAs to receive credit for them!

Project Presentations

A first for this class, you will be required to present your project to both of the class TAs, at a time period outside of class (by appointment slot). The presentation is to be short (between 5 and 10 minutes), and it should showcase your features, your challenges, how you overcame them, and your bugs, as well as a demonstration. All group members should participate in the presentation. The presentations will be held after the final project submission date.

Project Submission

Place all of your source code, your build files (makefile or build.xml), and your README, into a directory named "<Lastname1>_<Lastname2>_<Lastname3>_CSE353_Project2" and tarball the directory. The tar archive should also be named using the same convention: "<Lastname1>_<Lastname2>_<Lastname3>_CSE353_Project2.tar.gz". Zip files are also acceptable. Submit your archive file to Canvas before the deadline. Please see the class syllabus regarding late assignment policy.

Academic Honesty

ALL WORK MUST BE YOUR OWN! YOU MUST CITE ANY CONTRIBUTIONS THAT YOU RECEIVE FROM CLASSMATES OR ANY OTHER EXTERNAL SOURCES. This doesn't include contributions from your group members.

All rules in the New Mexico Tech Academic Honesty Policy strictly apply.

Grading
The project will be graded according to the following rubric:

TOTAL POINTS	**300**
General Project	50
Build System (Makefile)	10
Clean Exit	5
Frame Format Design	30
Proper naming of directory and tarball	5
Switch	100
Accept multiple connections	25
Global Frame Buffer	10
Switching Table	25
Initial Packet Flood	10
Smart frame switching (no flooding after you learn where your nodes reside)	30
Node	30
Proper instantiation of nodes	5
Read input file	10
Write output file	10
Only accept frames destined for it	5
Documentation	70
Frame Format Specification	20
Compilation Instructions	10
Useful Comments and Self-documenting variable names	20
Feature Checklist	20
Presentation	50

Contact Information
If you have questions, please send a message through Canvas to the class TA. If this fails for some reason, please send via email to amir.mirzaeinia@student.nmt.edu, alahy.ratul@student.nmt.edu or stop by TA office hours.

Recommended References
Sockets:
Java
All About Sockets: http://docs.oracle.com/javase/tutorial/networking/sockets/index.html
C#
Sockets: http://msdn.microsoft.com/en-us/library/b6xa24z5.aspx
C++
Practical C++ Sockets: http://cs.baylor.edu/~donahoo/practical/CSockets/practical/
Sockets Tutorial: http://www.linuxhowtos.org/C_C++/socket.htm
Multi-process and Multi-Threaded Programming:
Java
JavaDoc – Thread: http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html
Java Threads Tutorial: http://www.javabeginner.com/learn-java/jav
a-threads-tutorial
C#
Threading Tutorial (C#): http://msdn.microsoft.com/en-us/library/aa645740(v=vs.71).aspx
C++
Introduction to C/Unix Multiprocess Programming: http://www.osix.net/modules/article/?id=641
Threads in C++: http://www.linuxselfhelp.com/HOWTO/C++Programming-HOWTO-18.html
Switching
Overview of Layer 2
Switching:https://supportforums.cisco.com/document/68421/overview-layer-2-switched-networks
-and-communication