CSE/IT 353 Project #1

TCP/IP Socket Programming-- Network Nodes Communication
Due: Sunday, September 21 by 11:59pm

Goal

The aim of this project is to practice interprocess communication via TCP/IP sockets. This is designed to prepare students for the main semester project which will use TCP/IP sockets for communication.

Description

Create a program that uses forking/threading to create three nodes: **A**, **B** and **C**. Each process/thread represents one of the three nodes. These nodes will send and receive data using sockets. The transmitted data will be located in specified text files. The data will be transferred as follows:

1. **Node A** will send data from *confA.txt* to **node B**.
2. **Node B** will print the data received from **node A** to the screen (console) in the following format:
    a. Node B received: <data received from node A>
3. **Node B** will send data from *confB.txt* to **node C**.
4. **Node C** will print the data received from **node B** to the screen (console) in the following format:
    a. Node C received: <data received from node B>
5. After **node A** and **node B** have finished sending all data, they will send the string “terminate” to notify their receivers (**B** and **C** respectively) that all data has been transmitted.
6. Finally, all the nodes should close their opened sockets and exit.

You need to give descriptive error messages, such as port conflicts. Your program also needs to work correctly regardless of the start order of the various nodes. This means that each node should check on the status of the other nodes, and wait if necessary. There are references to help you get started for sockets, multiprocess and multithread programming at the end of this document.

Configuration Files

Each node will have its own configuration file, which includes the port number and the data to be sent across the sockets.

Node A has the configuration file named *confA.txt*, which has the following format:

```
<the port number that node B is listening to>
<data line 1>
<data line 2>
……
```

Node B has the configuration file named *confB.txt*, which has the following format:

```
<the port number that this node is listening to>
<the port number that node C is listening to>
<data line 1>
<data line 2>
……
```

Node C has the configuration file named *confC.txt*, which has the following format:

<the port number that this node is listening to>

Here is an example of what *confB.txt* could contain:

```
5000
5005
This is a sample line of text.
This is another sample line of text.
Node C will be printing out these three lines.
```

Each line of data in the text files will have no more than 100 ASCII characters. Your program should work with any configuration files that follow the above format and naming scheme.

Requirements

1. **Language:** You are required to implement the project in one of the following programming languages: Java, C#, or C++.
2. **Environments:** Your program must be able to compile and run on the TCC machines. All projects will be graded on TCC, and if your project does not compile or run on this system, then your grade will suffer.
3. **Make file, build.xml, or MSBuild file:** You are required to create one of these in order to allow easy compilation from the command line. You must include **instructions** on how to use this file in order to compile and run your program from **command line**. If you do not, **I will not grade your project!**
    - **a. C++**: Create a make file.
    - **b. Java**: You can create an Ant build.xml file or a make file. (Yes, make files work with Java programs; it's just not considered best practice).
    - **c. C#**: Create an MSBuild file.
    - *d. There are references at bottom of this document to help you create these files.*
4. **Documentation:** You need to document the following
    - a. Your code should be well commented.
    - b. How to compile and run your program. If your program is platform specific, make sure to document which OS (Windows or Linux) must be used to compile and run your program.
    - c. The names of all files in the project and a brief description of what they are.
    - d. Provide a check list that includes what features are implemented and missing. See below for the checklist.
    - e. A list of all known bugs. If there are no known bugs, clearly state it, or you will not receive credit for this portion. Reported bugs may reduce the corresponding point deduction.

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 configuration files, since they are guaranteed to strictly follow the format defined above.
2. You are not required to check the network status, since they are guaranteed to work.
3. You are not required to provide a GUI. Just print out the output on the screen (console).

Checklist

Include the following checklist in your documentation. Each item should be marked as complete, missing or partial. In the case of a partial status, make sure to add a description. The following is an example. Change the "Status/Description" column for your own project.

Feature	Status/Description
Project code has appropriate comments.	Complete
The project compiles and builds without errors.	Complete
The program starts.	Complete
Creates and runs all the threads or processes.	Complete
Loads the configuration files.	Partial. The last line may be lost.
Sends data through sockets.	Partial. Sometimes the sender crashes.
Receives data through sockets.	Missing. No data received.
Prints out received data.	Missing. No print out.
The program exits after sending/receiving data.	Partial. Sometimes the program halts.

Grading

The project will be graded according to the following rubric:

Implementation (75):

Make file: __________/5

Correct implementation (it works as specified): __________/60

Program exits appropriately (including closing sockets): __________/10

Documentation (25):

In code documentation (comments): _____/10

*Documentation of project in your README: _____/15

*Your README Documentation should include <u>at least</u>:

- DETAILED compilation and run instructions
- File descriptions
- Feature checklist
- List of known bugs (or a clear statement that there are no known bugs)

ALL WORK MUST BE YOUR OWN AND YOU MUST CITE ANY CONTRIBUTIONS THAT YOU RECEIVE FROM CLASSMATES OR EXTERNAL SOURCES.

All rules in the Academic Honesty Policy strictly apply.

Extra Credit (30%)

As a separate project, implement the Linux/Unix "finger" command in C. To do this, write a "client" program which connects to the "finger" server in any of the TCC machines. This program will utilize **socket()**, **connect()**, and **write()** system calls.

In addition, some other system calls will be used, such as: **getservbyname()** and **gethostbyname()**.

The input to your program will be a machine name in any TCC machines and a list of user login names.

The output of your program should be the result of executing the finger command on the input machine for each user name in the input list. Reference the Linux man page for the finger command for details about the output. You do not have to implement the options (i.e. -s, -l, etc.).

Here is a useful hint about the files you should #include in your program and some types (C structures) you may use in your program.

```
#include <stdio.h>
#include <sys/types.h>
#include <errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>

struct sockaddr_in srv_addr;
struct servent serv_info;
struct hostent host_info;
```

As this is a separate program, you must submit a separate README that contains:

- Compilation and run instructions.
- A brief description of what the "finger" command does.
- Any bugs in your program. As this is extra credit, any program bugs will be graded harshly, and undocumented bugs will reduce the grade further.

Project Submission

Project 1 should be submitted online via Canvas.

Please either tar or zip your files and include the course name, your full name, and the project number in the zip/tar file's name.

Example submission file: cse353_project1_JohnSmith.zip

Contacts

If you have questions, please send an email to:
rroach@nmt.edu

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

Multiprocess and multithreads:

Java
"Java Docs": http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.html
"Java Threads Tutorial" (5 pages): http://www.javabeginner.com/learn-java/java-threads-tutorial

C#
"Threading Tutorial": 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

Makefile, build.xml, MSBuild

Java
Make file for Java programs:
http://www.cs.swarthmore.edu/~newhall/unixhelp/javamakefiles.html
build.xml file "Tutorial: Hello World with Apache Ant":
http://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html

C#
"How to: Write a Simple MSBuild Project":
http://msdn.microsoft.com/en-us/library/ms171479(v=vs.90).aspx