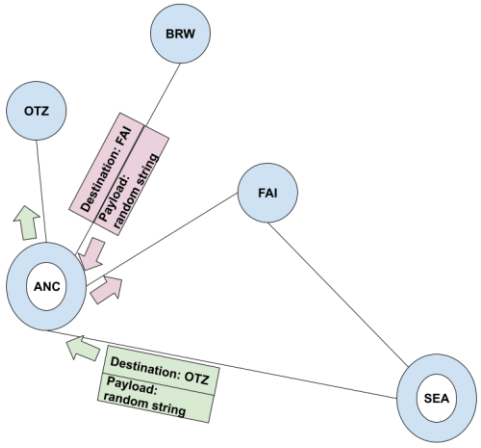
Assignment 1: Socket Programming

This assignment consists of a coding problem and a documentation component. The assignment is to evaluate your progress and understanding of the course material, both principles and coding, rather than to judge you on superficial matters irrelevant to the course material. When in doubt, clarify with your instructor until you are clear and confident. To an extent, this assignment functions as a flipped learning exercise. You learn certain topics through this assignment well before we cover the underlying networking topics in detail in class.

Assignment Description



As part of the coding component, you will write a socket program to simulate the airline traffic of the state of Alaska served by Alaska Airlines as a hub-and-spoke topology. The Figure is representative only. **You should ideally have more than just these 5 nodes.** The goal is the ability to simulate a network through socket programming, with data transfers between the origin and destination via an intermediate node (comparable to a layover in the airport network, or a router in computer networks).

There are a few hubs (ANC and SEA) that will see large traffic as they provide layovers to several passengers in addition to serving as the origin or the destination. Then, there are small airports (BRW and OTZ, for example) that will either have direct traffic with the hubs or have passengers to/from other small airports via a hub. You can ignore the tiny rural airports connected by regional airlines via their bush planes, as those will create more layovers and hubs.

- Each airport should be considered a node. The entire system of airports can run on a laptop as multiple processes, initiated through instances of command line/terminals. You could consider each of these nodes/airports as a “peer” with each node having the server and client capacities (so, they can initiate the sending of the messages).
- Each message between these nodes simulates passengers across airports. The message should contain the from (origin) airport, to (destination) airport, and optionally a layover airport, in addition to the passenger identifying detail (such as their name indicated by the “payload”).

Students can use hostname/IP-Address:Port as IATA airport code. 192.168.1.236:8080, 192.168.1.236:8081, 192.168.1.236:8082, … or maintain a lookup table for the mapping of the IATA codes to these values.

First, the airports should be initialized. Then, the passengers are “routed” through the airport, as passed by command line input from different source airports. A passenger

will, at most, have one layover, as in FAI → ANC → BRW, or have direct flights as in BRW → ANC, FAI → ANC, or SEA → FAI. Hubs read and interpret the message to find the destination of each passenger. Thus, they decide whether they are the final destination or a layover. Students can assume there will be no direct flight between two small airports. To limit the scope of the exercise to a few nodes, any air traffic from/to outside Alaska and Seattle can be ignored. Students are encouraged to make further reasonable assumptions in designing the solution. However, please do not make a system with just 3 – 5 peers.

README

Please answer these three questions clearly in the README.

1) How to run your program? Please state your assumptions clearly.

2) Do you see parallels with systems such as Internet eXchange Points and the Internet traffic, and how did we simulate the traffic with a hub-and-spoke topology here? Explain.

3) Please state your assumptions and your thoughts inspired by this exercise in the context of Computer Networks.

Submission

The submission should consist of

i) a single zip folder that has all the code that needs to run, together with log files.

Make sure your programs print informative logs into the file (s). You should program your nodes (clients and servers) to each print an informative statement whenever they take an action (e.g., whenever they send or receive a message, detect termination of input, etc.), so that you can see that your processes are working correctly (or not!). This also allows the instructor to easily determine from this output if your processes are working correctly. Run your respective programs once to showcase all scenarios and include the log files produced by the execution as part of this Zip folder.

ii) An accompanying report "README" in a Microsoft Word, txt, or PDF format.

This report presents your code and execution as well as your reflections to address the other aspects of the assignment (such as your assumptions and overall reflections). You can call it README, as it can be considered an extended README to help me understand and run your program. But it also indicates that you have addressed all the assignment expectations and serves as a Design Document. There is no maximum page limit or minimum page expectations. The document should mention the requirements (which versions of the programs are needed to run this) and the steps to configure and run your program.

iii) (Optionally) A screencast (video) of your program running.

If you think this will help as documentation, please include a screencast (a capture of the screen, presented as a video) showing all your server and client instances running smoothly.

Rubric

The assignment is evaluated across four different aspects. Each aspect is judged on a scale of 0 – 4, from 0 being “Ungradable” to 4 being “Excellent.

- **Excellent** (4): The work is either perfect or has negligible flaws.
- **Good** (3): The work is of good quality. Only minor revisions would be needed for the work to be Excellent.
- **Satisfactory** (2): While the work has some room for improvement, the student has put in a good-faith effort to complete all the work and demonstrated sufficient mastery of the material. A few revisions would be needed for the work to be considered Good.
- **Needs Improvement** (1): The student has put in a good-faith effort to complete the work, but revealed a lack of mastery in the material that can be addressed via concrete feedback. The work could become Satisfactory or better with some major revisions.
- **Ungradable** (0): The student did not make a good-faith effort to complete the work. This includes not submitting the (relevant part of) work at all but also situations like submitting only placeholder code or code segments merely taken from the textbook.

The grading matrix:

1. Code functionality: The submitted code is complete in its functionality. The program scales reasonably to the complexity of the task: a reasonable number of airports simulated as sockets/processes and a reasonable number of passengers simulated by the messages sent between these sockets. A main program to run the entire system is included to showcase this complexity and scalability with all the nodes (airports) and messages (passengers).
2. Code relevance: The submitted code connects well with the course materials (as justified through the README). i.e., the code does not merely function. But rather, serves its purpose as the learning material for this class. **This is to ensure you are using socket programming for the assignment.**
3. Logs: The submitted code produces meaningful logs. The logs are attached to the submission properly. The logs match the task well, explaining the procedure.
4. README: The submitted README shows an understanding of the assignment and states the assumptions well. Aspects such as scalability and caveats are documented, and the questions raised in the assignment are answered well.

Total points possible: 4 * 4 = 16.

General Information

For general information on ethics, late submissions, and best practices, please refer to the syllabus. The deadline information is incorporated in the respective Blackboard assignment, “Assignment-1”.

Programming assignments (Assignment-1 and Assignment-2) are individual works. No group work is allowed. However, you are free to seek help from or provide help to your peers in good faith. Just don't plagiarize or do someone else's homework for them. Any help received from your peers should be acknowledged, as in, "Acknowledgment: Jane helped me understand socket programming in C++. She fixed the code udp.cpp lines 21 - 23. Joe helped install the VMWare hypervisor for this assignment. I thank and acknowledge their help in this assignment." It is just a good practice in academia (and anywhere in life, actually) to acknowledge the help you receive towards achieving your goal and be transparent about it.
The same goes for using external sources (The Internet or AI tools) in getting this assignment done. There is no need to cite the primary textbooks of the course (or the provided resource materials in this assignment). However, if you use a website or an AI tool that helped you design the solution for this assignment, you must acknowledge that. Please review the syllabus carefully on the use of AI in completing homework assignments. Improper use of AI tools will lead to a failure or a low grade in the assignment.

Coding Guidelines

You may write your programs in Python, Java, C++, or C on any platform (Liux/Unix, Mac, or a Windows PC). Other languages are OK, but please ask me first. See http://www.lowtek.com/sockets/ for links to tutorials on the WINSOCK API.

Programming the assignment in Python

We discussed socket programming in detail in class, providing client/server implementations for both TCP and UDP in Python. Therefore, Python may be your default choice for this assignment, unless you are more familiar with another language. If you are doing the assignment in Python, here’s a page that will get you started that will get you started in Python: http://www.eurion.net/python-snippets/snippet/Threaded%20Server.html. The 8th edition of the textbook (and what we’ve discussed in class) uses sockets in Python. A Python socket tutorial is http://docs.python.org/howto/sockets.html

Programming the assignment in Java

If you are doing the assignment in Java, see the Sun's socket programming tutorial, http://java.sun.com/docs/books/tutorial/networking/sockets/index.html This page is for JDK 8. Later changes may not be reflected. Here is the Java-based socket section of the text that is from the 5th edition: http://gaia.cs.umass.edu/cs453_fall_2013/hw_pa_labs/K_R_sockets_in_Java.pdf
Java encapsulates the concept of a client-side connection-oriented (TCP) socket with the class, Socket. Details about the Socket class can be found at

https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html Java encapsulates the concept of a server-side connection-oriented socket with the class, ServerSocket. You will need to use the accept() and close(0) methods of this class. There is no reason to limit your code to Java/JDK 8, of course. Please feel free to use later versions of Java.

Programming the assignment in C

If you are doing the assignment in C, you will need to master the C system calls needed for socket programming. C gets you closest to the operating system's socket-related system calls. These include *socket(), bind(), listen(), accept(), connect(),* and *close().* For tutorials on socket programming specifically under Linux, see http://www.lowtek.com/sockets/ and http://beej.us/guide/bgnet/. You will also need to learn about the *fork()* system call. Here is an example that will get you going: http://www.tutorialspoint.com/unix_sockets/socket_server_example.htm

Programming notes

Here are a few tips/thoughts to help you with the assignment:

- You must choose port numbers greater than 1023 (to be safe, choose the server port numbers larger than 5000). If you want to explicitly choose your client-side ports, also choose numbers larger than 5000.
- You may need to know your machine's IP address, when one process connects to another. You can telnet to your own machine and seeing the dotted decimal address displayed by the telnet program. You can also use the Unix nslookup command . On Windows, see the ipconfig utility. On a Mac, you can run the terminal program and use the ifconfig commend (just type in ifconfig or ifconfig | grep "inet ").
- Many of you will be running the clients and senders on the same machine (e.g., by starting up the receiver and running it in the background, then starting up the relay in the background, and then starting up the sender. This is fine; since you use sockets for communication, these processes can run on the same or different machines without modification. Recall the use of the ampersand to start a process in the background. If you need to kill a process after starting it, you can use the Unix kill command. Use the Unix ps command to find the process id of your server
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port IDs, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use, if you are using a UAA server or another multi-user computer (other than your personal laptop/computer) for your testing.

Acknowledgments

The Coding Guidelines section is adapted and extended from an assignment provided in the main textbook.
https://gaia.cs.umass.edu/kurose_ross/programming/simple_socket/simple_socket_program_PA1.docx