CIS 505: Software Systems

Project 2 (12% credit)

Due: 19 March 2015 at 11:59pm (no late submissions accepted)

You can discuss high-level concepts with other students, but you are required to write your own code. We will be running the MOSS analyzer to detect any form of plagiarism, including those copying from previous years.
All offenders will **be referred to the Office of Student Conduct (OSC) for first time offence**, regardless of severity of violation. Offenders may be suspended for one semester by our university. In addition, if any solutions or code were obtained from a student who has taken this class previously, that student will similarly be referred to OSC and receive a similar set of points deductions, resulting in a change of grade. Students who have graduated but are found out to have supplied code or solutions to this batch of students will similarly be dealt with under our school's academic integrity guidelines. If you are unsure whether you can use an existing library or data structure, consult with the teaching staff.

1. Introduction

In this exercise, you will design and implement a simple electronic voting system using the "socket" interface (UDP & TCP) and also Sun remote procedure calls.

This simple voting system should allow clients to vote for their favorite candidate in an election over a network. Each implementation needs to support five functions:

- **zeroize(),** which sets or resets the system to an initial state, with no candidates, no voters, and zero votes. Return TRUE if successful, FALSE otherwise.
- **addvoter(int voterid),** which adds the *voterid* to a list of authorized voters. Return OK if successful, EXISTS if the *voterid* is already present in the system, and ERROR if there's any error.
- **votefor(char *name, int voterid),** which adds one vote to the total vote count of the candidate referred to by *name* if the *voterid* has not already voted. If the named candidate is not already present in the system, add the candidate to the system with a vote count of 1. Else, if the candidate is already present in the system, increment his or her vote count by 1. Return EXISTS if the candidate exists and was successfully voted for, NEW if the candidate didn't exist previously and was successfully voted for, NOTAVOTER if *voterid* is not in the list of authorized voters, ALREADYVOTED if the *voterid* had already voted, and ERROR if there's any error.
- **listcandidates(),** which returns a list of candidates currently with votes (but not their vote totals).
- **votecount(char *name),** which returns the integer vote total for the candidate referred to by *name*, or -1 if the candidate isn't in the system.

NOTE: For each system (socket datagrams and RPCs), please specify the data structures, network

protocols, and other aspects of the system (using C, XDR, and English as appropriate) in sufficient detail so that if your documentation is provided to other people they should be able to produce compatible client and server implementations. In particular, for the UDP and TCP parts, please state the format of the five datagram message payloads (and the response message from the server) clearly. Similarly, in the case of the RPC version, specify the various functions and their arguments clearly.
You may impose reasonable limits on certain aspects of the system, for e.g., the number of candidates, the length of candidate names, etc, but be sure to document these limits in your specification.

IMPORTANT: You do not need to handle failures (lost messages, machine crashes) beyond what the socket and RPC mechanisms themselves do.
While your code should be designed to work over the network, you can run the server locally on the same machine as the client by using the "localhost" (127.0.0.1) interface for the server running on any available port number (which you should report when you start the server so you can specify it on the client command line). We propose that you use the last 4/5 digits of your Penn ID as Server's port number if possible. However, do ensure that the port number is greater than 1024 and do mention the port numbers that you use in the write-up. If you plan to run multiple clients on the same machine, you may consider using the ports that are at an incremental offset of 1,2,3..etc from this port number.

VERY IMPORTANT: To submit your work, please put your source and text files in a directory and use the turnin command on the Eniac machine (not speclab). All programs should be written in C (with XDR and RPCGEN) and you should be able to compile and execute them on Speclab Linux machines. **Take the time to write and code your answers clearly and lucidly, whether the language you are using is English or C. Submit only your source code and supporting text.** Do not include compiled or large output files (e.g, write.out or fprint.out) in your submission. Those files can be very large and submitting them can have a disruptive effect on our shared computing resources.

2. Part 1 (33% credit)

For this part, you should use the socket API package and the UDP datagram; write a server for the voting system and five client programs that exercise each of the five functions specified above. Your server may be a single-threaded process (where each incoming call is processed in sequence).
In addition to your server, you should build five separate client programs (which can be executed from the linux command line) that exercise each of the five voting functions mentioned above. Your clients should accept command line arguments that specify the server's IP address and the port number along with the appropriate arguments to whatever function is being executed. E.g., "./vote-udp localhost 7432 bush 01" should send a message to the server running on localhost port 7432 to cast a vote for "bush" with voter's Id as 01. **You must follow the name conventions for clients and the server in table 1, and include any customizations in a README.**

3. Part 2 (33% credit)

Repeat part 1, using TCP stream sockets instead of UDP datagrams.

4. Part 3 (34% credit)

For this part, you should use the RPCGEN package, write a server for the voting system and clients that use each of the RPCs you specified above. Your server may be a single-threaded process (where each RPC call is processed in sequence).

In addition to your server, you should build five separate client programs (executed from the linux command line) that exercise each of the five voting functions mentioned above. Your clients should accept command line arguments that specify the name of the server along with the appropriate arguments to whatever function is being executed, e.g., "./vote-rpc localhost hillary 01". **You must follow the name conventions for clients and the server in table 1, and include any customizations in a README.**

To standardize the grading, please use the following command line arguments to implement each function.

	Part 1	**Part 2**	**Part 3**
Server	./server-udp	./server-tcp	./server-rpc
zeroize()	./vote-zero-udp <server_ip_address> <server_port>	./vote-zero-tcp <server_ip_address> <server_port>	./vote-zero-rpc <server_ip_address>
addvoter(int voterid)	./add-voter-udp <server_ip_address> <server_port> <voter_id>	./add-voter-tcp <server_ip_address> <server_port> <voter_id>	./add-voter-rpc <server_ip_address> <voter_id>
votefor(char *name, int voterid)	./vote-udp <server_ip_address> <server_port> <candidate_name> <voter_id>	./vote-tcp <server_ip_address> <server_port> <candidate_name> <voter_id>	./vote-rpc <server_ip_address> <candidate_name> <voter_id>
listcandidates()	./list-candidates-udp <server_ip_address> <server_port>	./list-candidates-tcp <server_ip_address> <server_port>	./list-candidates-rpc <server_ip_address>
votecount(char *name)	./vote-count-udp <server_ip_address> <server_port> <candidate_name>	./vote-count-tcp <server_ip_address> <server_port> <candidate_name>	./vote-count-rpc <server_ip_address> <candidate_name>

Table 1: Naming Conventions

For the RPC implementation, you can also choose to include all the RPC calls in one file. If you do so, indicate in a README.

5. Submission guidelines

5.1. What to turn in

For this project place all required folders and files into a folder called proj2_pennkey (where pennkey)

is your pennkey.
You will need all of the following items to receive full credit for this project.
• Source code: Submit the source code for part1, part2 and part3 in separate sub-folders, as PART1, PART2 and PART3. If these parts share common files, you may place them in the parent directory proj2_pennkey.
• Readme file: Submit three individual README for each of the three systems you implemented above, named as README1, README2, README3, you can place them either inside each individual source code subfolder or in the parent directory.
• Please include Makefiles for your submissions.

5.2. Turning it in

Use the command **"turnin –c cis505 -p proj2 <folder>"** to submit the project.

You can use **"turnin -l -c cis505"** to see whether project is current open for acceptance at any time. To make sure your submission is successful, you can use **"turnin –c cis505 –v –p proj2"** to check the status of your submissions.
If you want additional information for "turnin", there is a turnin man page which explains the usage in more detail, including some additional arguments. And the below link might help.
http://manpages.ubuntu.com/manpages/maverick/man1/turnin.1.html

6. Extra Credit (15% credit)

Non-blocking I/O extra credit. For this extra credit part, enhance your implementation in part 2, but utilize non-blocking I/O for implementing your TCP communication **(HINT: use select() which was covered in class).** You can follow your own name convention for clients and the server, but you need to clearly specify those in your README file.

IMPORTANT: For this part, place all your source code and a README file inside a folder called EXTRA, put the EXTRA folder inside proj2_pennkey (where pennkey is your pennkey), and submit all your regular and extra credit work together using "turnin".

The extra credit section is entirely optional. We offer extra-credit problems during the semester as a way of providing challenges for those students with both the time and interest to learn and pursue certain knowledge in more depth. **You should only attempt the extra credits after you have completed the regular portions of the project.** We recommend that you only start working on extra credits after you have finished the regular credits.