CIS 505: Software Systems

Project-2 (12% credit)

Due Date: Milestone 1 – 26 Feb 2016 & Milestone 2 – 14 Mar 2016

(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.

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 following functions:

- **changepassword(),** which enables the admin to change the password of the server. You should use standard input instead of command line arguments to ask for current username and password, and should allow to change the password only if the entered information is correct. The username can just be considered as a name for the server, hence we are not interested in changing that. Return TRUE if successful, FALSE otherwise.

- **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.

- **viewresult(char *username, char *password),** which returns the list of candidates with their vote count and also declare the winner/tie/null(for no candidate) from the server and shuts down the server so that no new votes can be added. The processing of the result should be done on the server side. The client program should be able to view the results only if the username and password, passed as arguments, match with the current admin credentials of the server. Return the list if successful, else return UNAUTHORIZED or ERROR if there's any error.

NOTE:

- If the server is invoked without any command line arguments, it should have a default username **"cis505"** and password **"project2"**, otherwise you should set the username and password of the server according to the input arguments. Print an error if the number of arguments is not equal to 0 or 2.
- 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 seven 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.

Milestone 1 (Due: 26 Feb 2016 at 11:59pm)

Part 1 (28% credit)

For this part, you should use the socket API package and the UDP datagram; write a server for the voting system and seven client programs that exercise each of the seven 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 seven separate client programs (which can be executed from the linux command line) that exercise each of the seven 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.**

Part 2 (28% credit)

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

Milestone 2 (Due: 14 March 2016 at 11:59pm)

Part 3 (28% 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 seven separate client programs (executed from the linux command line) that exercise each of the seven 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.**

Part 4 (16% credit)

Non-blocking I/O - For this 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 should follow the same naming convention as part 2, from the table 1.

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

	Part 1	**Part 2**	**Part 3**
Server	./server-udp <username> <password>	./server-tcp <username> <password>	./server-rpc <username> <password>
changepassword()	./change-password-udp <server_ip_address> <server_port>	./change-password-tcp <server_ip_address> <server_port>	./change-password-rpc <server_ip_address>
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>
viewresult(char *username, char *password)	./view-result-udp <server_ip_address> <server_port> <username> <password>	./view-result-tcp <server_ip_address> <server_port> <username> <password>	./view-result-rpc <server_ip_address> <username> <password>

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.

Submission guidelines

5.1. What to turn in

For this project place all required folders and files into a folder as described below (Points will be deducted if this convention is not followed). You will need all of the following items to receive full credit for this project.

Milestone 1 -

• Source code: Submit the source code for part1, part2 in separate sub-folders, as **proj2_pennkey_ms1/part1** and **proj2_pennkey_ms1/part2**. If these parts share common files, you may place them in the parent directory **proj2_pennkey_ms1**, where pennkey is your pennkey.

• Readme file: Submit two individual README for each of the two systems you implemented for this milestone, named as README1 and README2 you can place them either **inside each individual source code subfolder.**

- Please include Makefiles for your submissions. Each part should have its own makefile within the sub-folders. The command **"*make part1*"** and **"*make part2*"** should compile the respective parts.

Milestone 2 -

• Source code: Submit the source code for part3 and part4 in separate sub-folders, as **proj2_pennkey_ms2/part3** and **proj2_pennkey_ms2/part4**. If these parts share common files, you may place them in the parent directory **proj2_pennkey_ms2**, where pennkey is your pennkey.

• Readme file: Submit two individual README for each of the two systems you implemented for this milestone, named as README3 and README4 you can place them either **inside each individual source code subfolder.**

- Please include Makefiles for your submissions. Each part should have its own makefile within the sub-folders. The command **"*make part3*"** and **"*make part4*"** should compile the respective parts.

5.2. Turning it in

Use the command **"turnin –c cis505 -p proj2ms1 <folder>"** to submit the project milestone 1 and **"turnin –c cis505 -p proj2ms2 <folder>"** to submit the project milestone 2.

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 –p proj2ms1 -v" or "turnin –c cis505 –p proj2ms2 -v"** to check the status of your respective 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