EECS 22L: Project 2

Prepared by: Mina Moghadam, Mihnea Chirila, Hamid Nejatollahi, Bo Tsai
Prof. Quoc-Viet Dang and Prof. Rainer Dömer

Feburary 12, 2018

1 Instant Messaging Application

This is the second team programming project of EECS 22L, ”Software Engineering Project in C language”.

The goal of this programming exercise is to design and develop a chat application in which a user can send messages as well as react other users' messages. Each user can talk to each of the other users one-by-one at the same time. The application manages user accounts including creating accounts, adding, and deleting contacts for each user. The provider stores the information of all users so that users can query the provider to find out available contacts to chat with.

This project is designed to be an interesting exercise where you can practice all elements of software engineering and work in teams. In particular, you will practice specifying and documenting the software program, designing data structures and algorithms, designing software modules, writing original source code, testing and debugging the software program, and collaborating and communicating effectively in a team.

1.1 User app:

During first run, the user app provides account and password input fields and allows users to sign-up or log-in. For a new user, one can sign up a user account with the password which shall be transmitted to, checked with, and/or stored on the provider in case of redundancy. For current users, they can log in and become online if they enter the correct credentials.

After logging in, the user should be able to access other allowed users status and TCP/IP addresses maintained on the provider. A user can add or delete another user account as contacts and accept or reject an invitation all via the GUI. Once obtained the permission, they are free to initiate a live chat by connecting to another user's TCP/IP address.

In summary, a user app can:

- Sends to the provider the requests including:
  - **Register** with user's account name and password as well as the `IP` address and `Port` number.
  - **Check** the entered account name and password.
  - **List** of available user contacts.
  - **Update** its status to the provider. Status can be available, idle, or offline.
  - **Add** a user account name to its contact.
  - **Delete** a user account name from its contact.
- Send the request to chat with individual users in the contacts and start to chat.
- Accept or reject invitations from non-contact users.

1.2 Provider

The provider should listen and responds to any user apps requests. To manage user accounts, the host stores the information as well as the status of all users.

- Listen for the queries from the clients
- Send back the response to the clients
- Stores the information about all the users in the network including:
  - Status of the users:
    - Active
    - Idle
    - Offline
  - Account name
  - Account password
  - IP address
  - Port number

2 Program Specification

In this project, we want to design a instant messaging system consisting of a central host and multiple user apps which allow users to create accounts with passwords, to add or delete other users account as contacts/friends, to chat with other users one-by-one at the same time.
There are several features that we expect the software system to have. We distinguish between the minimum requirements for this project, and more advanced options that are goals and optional features (bonus points).

2.1 Basic functions that are required:

1. Each user should be able to register with the provider.
2. Each user should be able to log in to the user app.
3. Each user should be able to add other users to the contacts.
4. Each user should be able to delete users in the contacts.
5. Each user should be able to leave/restart the chat at any time.
6. Each user should be able to see the status of other users in the contacts.
7. Each user should be able to accept/reject non-contact users' invitations.
8. Each user should be able to chat to online users in the contact.
9. Each user should keep track of all its chat threads. Whenever the app receives messages, it highlights the unread message in GUI.
10. The Provider should listen to any requests from user app(s)
11. The Provider should be able to manage the user account information and status.
12. The Provider should respond to users' request with necessary data.
13. The Provider should check a new user sign-up and accept existing user log-in. Otherwise, it rejects the connection.

2.2 Advanced options that are desirable (but optional): (Bonus)

1. Transmit files, pictures, and/or videos
2. Save and display pictures
3. Apply filters to the pictures and/or videos
4. Customize font and/or color
5. The program may provide a graphical user interface (Requires GUI (GTK 2.0)) (as introduced in Lecture 4)
6. Save log of each conversation
7. Retrieve chat history
8. Message seen status
9. User typing status
10. Any other options that make the chat application irresistible to buy!

3 Software Engineering Approach

In the design and implementation of this project, we will follow basic principles of software engineering in a close-to-real-world setting. You will practice the major tasks in software engineering to build your own software product. We will not provide detailed instructions on how to design the program as we did for the assignments in EECS22. Instead, your team needs to come up with your own choices and practice designing the software architecture of a medium-size program and document it.

3.1 Team work

The software design programming in this course will be performed by student teams. Teams of 5-6 students will be formed at the beginning of this project. Team work is an essential aspect of this class and every student needs to contribute to the team effort. While tasks may be assigned in a team to individual members, all members eventually share the responsibility for the project deliverables.

The overall tasks of software design, implementation and documentation should be partitioned among the team members, for example, to be performed by individuals or pairs (pair programming). A possible separation into tasks or program modules may include:

- user app main program
- user interface for user app (textual and/or graphics)
- provider main program
- connections (data structures for sockets)
- user credential module
- log file module
- documentation
- testing

When planning the team partitioning, keep in mind that certain tasks depend on others and that some tasks are best handled by everyone together.

A team account will be provided on the servers (`bondi`, `laguna`, `crystalcove` and `zuma`) for each team to share source codes, data and document files among the team members. Since teams will compete in the projects, sharing of files across teams is not permitted.

Every student is expected to show up and participate in team meetings. Attendance of the weekly discussion and lab sections is mandatory for the sake of successful team work.

3.2 Major project tasks

We list several steps here to approach the medium-sized programming project.

- *Design the software application specification*: work as a team to decide the functionalities of the program, the *input* and *output* of the program, and other things that describe the *features* of the program for the users.
- *Design the software architecture specification*: work as a team to design the data structure, program modules, application programming interface (API) functions between modules, and basic algorithms that will be used to solve the problem.
- *Build the software package*: write the source code and implement the program. Each team member may be in charge of their own modules and ideally work in parallel on implementation and module testing. Use Makefile for rule-based compilation to integrate the modules from different owners.
- *Version control and collaboration*: use a version control application, i.e. CVS (introduced in Lecture 3) to maintain the team project documentation and source code files. Team members can synchronize their own work with the others through the team repository located in the team account.
- *Test and debug the software*: work as a team to decide the testing strategies, write automated test programs or scripts, and debug the program when some of the test cases fail.
- *Software release*: release the software package with the executable program and documentation, e.g. the README file, user manual, etc. Release also the source code as a package for the further developers or maintainers. In contrast to Project 1, Project 2 has 3 software releases, **alpha**, **beta**, and **final** release.

3.3 Deliverables

Each team needs to work together and submit one set of deliverables each week. Here is the checklist of the files the team needs to submit and the due dates (hard deadlines).

Table 1: The Instant Messaging Project Deliverables

Week	File Name	File Description	Due Date
1	`Chat_UserManual.pdf`	The application specification	02/19/18 at 12:00pm (noo
2	`Chat_SoftwareSpec.pdf`	The software architecture specification	02/26/18 at 12:00pm (noo
3	`Chat_Alpha.tar.gz` `Chat_Alpha_src.tar.gz`	The alpha version of the instant messaging program, including the program source code and documentation	03/05/18 at 12:00pm (noo
4	`Chat_Beta.tar.gz` `Chat_Beta_src.tar.gz`	The beta version of the instant messaging program, including the program source code and documentation	03/12/18 at 12:00pm (noo
5	`Chat_V1.0.tar.gz` `Chat_V1.0_src.tar.gz`	The released software package for the instant messaging game and the program source code and documentation	03/19/18 at 12:00pm (noo

Note that we do require these exact file names. If you use different file names, we will not see your files for grading.

We will separately provide detailed templates (document skeleton, table of expected contents) for the textual documents and a detailed list of contents (directory structure and expected files) for the file archives. These grading criteria will be provided at the Projects tab of the course webpage.

3.4 Submission for grading

To submit your team's work, you have to be logged in the server `zuma` or `crystalcove` by using your **team's account**. Also, you need to create a directory named `chat` in your team account, and put all the deliverables in that directory. Next, change the current directory to the directory containing the `chat` directory. Then type the command:

```
%  ~eecs22/bin/turnin.sh
```

which will guide you through the submission process.

For each deliverable, You will be asked if you want to submit the script file. Type yes or no. If you type "n" or "y" or just plain return, they will be ignored and be taken as a no. You can use the same command to update your submitted files until the submission deadline.

Below is an example of how you would submit your team work:

```
zuma% ls # This step is just to make sure that you are in the correct directory that contains chat/
chat/
zuma% ~eecs22/bin/turnin.sh
==============================================================
EECSL 22L Winter 2018:
Project "instant message" submission for team1
Due date:  Mon Feb 19 12:00:00PM 2018
* Looking for files:
* Chat_UserManual.pdf
==============================================================
Please confirm the following:  *
"I have read the Section on Academic Honesty in the *
UCI Catalogue of Classes (available online at *
http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0) *
and submit original work accordingly." *
Please type YES to confirm.  y
==============================================================
Submit Chat_UserManual.pdf [yes, no]?  y
File Chat_UserManual.pdf has been submitted
==============================================================
Summary:
==============================================================
Submitted on Mon Feb 12 00:05:31 2018
You just submitted file(s):
Chat_UserManual.pdf
zuma% _
```

For a binary package, we expect the user to read the documentation and run the executable program as follows:

```
% gtar xvzf BinaryArchive.tar.gz
% evince chat/doc/chat.pdf
% chat/bin/userapp
% chat/bin/provider
```

For a source code package, we expect the developer to read the documentation and build the software as follows:

```
% gtar xvzf SourceArchive.tar.gz
% evince chat/doc/chat_software.pdf
% cd chat
% make
% make test
% make clean
```

Again, please ensure that these commands execute cleanly on your submitted packages.