

**COP 5615  
OPERATING SYSTEM PRINCIPLES  
FALL 2006**

**Project 1**

**Date Assigned: August 31, 2006**

**Date Due: 4:59 pm EST, September 13<sup>th</sup> (Wednesday), 2006 for on campus students**

**Date Due: 4:59 pm EST, September 18<sup>th</sup> (Monday), 2006 for EDGE students**

**Introduction:**

The purpose of project 1 is to help you learn and understand multi-threaded programming in JAVA. Distributed operating systems components are made up of many modules that need to communicate with remote modules on different systems and therefore you must learn about socket programming. This project will help you learn socket programming in JAVA.

**Details:**

In this project you will design, document, implement and test a talk-like application in JAVA. This is an individual project and NOT a team project. Your application will consist of both the server and the client needed to chat on the network. You will design two standalone JAVA applications. The user input module (UI) will handle all the user inputs including the commands and the chat messages to send. It will send these to the command server. The second application is the server, and will run three components; command server, talk message receiving server and the talk message sending module. Each of these will be implemented as a thread.

**Part 1: User input module**

[50%]

In this part you will write a Java application that will handle the user inputs on the standard console command line. The user input module should handle these following commands:

- exit – used to terminate the system
- alias – used to define named aliases
- remove – used to remove an existing alias
- send – command to send the messages
- list – lists all the current existing aliases
- setalias – allows to set one's own alias name

When the command 'exit' is sent, the input handler application must send the 'exit' command to the command server and then terminate itself.

The command 'alias' is used to define a named alias for the current session. The format will be:

**alias <name> <host name> <port number>**

An example command may look like this: **alias presto sand.cise.ufl.edu 65143**

The command 'remove' is used to remove a named alias from the current session database. The format will be:

**remove** <alias name>

An example command may look like this: remove **presto**

Messages to be sent across the network must be preceded by the keyword 'send' and an alias name. The format for this command will be:

**send** <alias name> <message>

Example:- if the user wants to send 'Hello! How are you?' across then he must input it as – **send presto Hello! How are you?** If keyword '**all**' is found in the <alias name> position then the message must be sent to all the existing aliases in the current session database.

The command '**list**' shall list all the existing aliases for the current session.

The command 'setalias' allows the application to set its own alias name. Its format will be:

**setalias** <alias name>

An example command may look like this: **setalias dexter**

Anything else may be ignored. Whenever the program receives an input from the console, it firsts checks if it is preceded by the valid command keyword or if it is the 'exit' command. If valid command has been entered the application must spawn a new thread and pass the just received command to that thread. The new thread then creates a UDP datagram containing the command as the data payload and then send it to the command server. After sending the command to the command server the thread must terminate. Assume that the command server is already running before the client application is started. Also pass the command server host name or IP address and its port number as a command line argument to this user input handler application.

A sample run for the user input module application may look something like this:

---

```
$ java ClientApp sand.cise.ufl.edu 12345

    resolving sand.cise.ufl.edu.. done! sand.cise.ufl.edu resolved to IP 128.227.205.18
    initializing system parameters :
    setting command server ip: 128.227.205.18 [ok]
    setting command server port: 12345 [ok]
    initializing command prompt now ...

    % setalias dexter
    % alias presto sand.cise.ufl.edu 65143
    % list
    % send presto Hello! OS Project is so cool!
    % send gameboy Dude lets have a pizza party tomorrow at my place.
    % exit

    Terminating all threads and exiting now.

$
```

---

You may need to resolve the host name to its IP address in case it is not entered as an IP address in the first place. In the above sample run the command server thread is running at port number 12345 on host sand.cise.ufl.edu.

**Part 2: Server module**

[50%]

In this part you will write the Java application that runs the command server, talk receiver server and the talk sender module each as a separate thread. The command server and the talk receiver server threads will start when the application starts and will terminate when the 'exit' command has been received.

This application when executed, spawns a UDP server (talk receiver server) thread that starts listening for all incoming messages from remote servers at any random free port. It must also display what port number it is listening to along with its own IP address.

It should also spawn a second thread running again as a UDP server (command server) that will listen for any incoming commands from the user input module again at any random free port. Again it should display what port number it is listening to along with its IP address. This information will help you to start the user input module as you must pass this port number and IP address as the command line argument to the application in part 1 above.

The program functions as follows: when the command server thread receives a datagram from the user input module, it extracts the payload. If it is a 'send' command then further extracts the alias name and the message to be sent. It then checks if the alias exists in its internal alias database or not. If no such alias exists it outputs an error message otherwise it then spawns a new sender thread and passes the message and the alias parameters such as the IP address and the port number to be sent across from it. The sender thread then creates a new Datagram packet with the IP address and the port number of the remote talk server and the message as the data payload. It also adds its own socket information such as the IP address and the port number and its own alias name of the talk receiver server so that the remote client could respond back if it chooses to. It sends this datagram across and then terminates itself. Remember: if the alias-name entry is 'all' then the message must be sent to all existing aliases.

When the talk receiver server receives a UDP datagram, it extracts the payload and further extracts the true message and the remote talk server connection information such as its IP address, port and the alias name and then displays the contents on the standard output. Furthermore it checks its own internal alias database for the existence of the alias extracted from the just received message. If the alias already exists and its corresponding IP address and the port number matches then no further action is needed. If the alias does not exist in the database, it creates a new entry for the new alias and stores the corresponding IP address and the port number. If a duplicate alias record is already found in the database it updates the corresponding IP address and port with the most recent one and also generates a message notifying the user of the update.

When the command server receives the command 'setalias' it then sets its own alias name to the argument following the command.

When the command server receives the command 'exit' it must ensure that all the thread terminates properly and then the application itself terminates.

A sample run for the server module application may look something like this:

---

```
$ java ServerApp
```

```
Starting talk receiver server thread now ...
talk receiver server thread started successfully on sand.cise.ufl.edu at port 54132
Starting command server thread now ...
command server thread started successfully on sand.cise.ufl.edu at port 12345
```

```
**Received a new message
goblet says: howdy
```

```
updating the alias list :: adding [goblet rain.cise.ufl.edu 5142] ... [ok]
*Incoming command: setalias dexter
setting self alias to dexter ... [ok]
*Incoming command: alias presto sand.cise.ufl.edu 65143
updating the alias list :: adding [presto sand.cise.ufl.edu 65143] ... [ok]
*Incoming command: list
```

```
+++++
goblet rain.cise.ufl.edu 5142
presto sand.cise.ufl.edu 65143
+++++
```

```
**Received a new message
goblet says: dude are you there?
```

```
Updating the alias list:: adding [goblet rain.cise.ufl.edu 5142] .... [failed]
entry already exists!!!!
```

```
*Incoming command: send presto Hello! OS Project is so cool!
Sending message ... [ok]
*Incoming command: send gameboy Dude lets have a pizza party tomorrow at my
place
Sending message ... [failed]
no entry found for alias gameboy
*Incoming command: exit
Setting exit flag for command server thread ... [ok]
Setting exit flag for talk receiver server thread ... [ok]
Waiting for all child threads to terminate ....
Command server thread exiting now!
Talk receiver server thread exiting now!
```

---

```
$
```

In addition to the above specified module descriptions you are supposed to handle any incomplete commands and invalid user entry at the console and any other unexpected errors appropriately.

**Project Requirements:**

Your submission must contain a comprehensive documentation named *Programmer's Guide* explaining your code structure. It should contain a brief writeup of various modules and functions that you use and also explain briefly your design philosophy. It must also contain a separate file named *Installation Guide* containing all necessary instructions for the TA to successfully execute your code. Also follow clean and structured coding style making use of proper indentation and appropriate comments. It is extremely important that your threads terminate properly.

- Missing Programmer's Guide – 10% penalty
- Missing Installation Guide – 5% penalty
- Shabby coding style – 10% penalty
- Runaway threads (threads do not terminate on program termination) – 20% penalty

**Submission Instructions:**

You will use UF-IBA submission system to submit your project. Please follow these instructions closely while submitting -

1. Tar all your files including the comprehensive documentation file into a single file using this command – `tar cvf proj1.tar <file list>`
2. Login to UF-IBA system using your login and image password
3. Choose submit homework options from cop5615fa06 control options
4. Submit your tar file and select project 1 from the drop down list
5. Note your confirmation number
6. Late submissions are discouraged and will result in heavy penalty