COP5615 – Operating System Principles – Spring 2010
Department of Computer and Information Science and Engineering
University of Florida

Project 1

Due dates:	January 22, 2010 (Friday) 11:59 pm ET (local) January 24, 2010 (Sunday) 11:59 pm ET (EDGE)

Introduction:

The main purpose of this project is to familiarize you with network socket programming in Java. You are asked to develop a simple logical ring structure with multiple nodes running on different physical machines. In a logical ring topology, a control token is circulated among the nodes in a circular fashion. The node which has the token can access the communication channel and send messages or perform tasks that requires exclusive access to some shared resource. Once that task is finished, the node releases the token and forwards it to the next node in the circular chain.

Helpful Resources:

Socket Programming Tutorial: http://java.sun.com/docs/books/tutorial/networking/sockets/index.html
How to invoke child processes: http://java.sun.com/developer/JDCTechTips/2003/tt0304.html
Multi-threading Tutorial: http://java.sun.com/docs/books/tutorial/essential/concurrency/index.html

Node Description:

Each node in this project will have a server module that will receive socket messages from its predecessor node and a client module that will send messages to the successor node in the ring. Each node application will be multi threaded implementation as in this project, even though the operations are sequential in nature, the server thread must be separate so that there is no deadlock in project initiation. Communication will be performed using Java sockets API using connection-oriented TCP/IP protocol. You are required to remotely start 5 nodes on 5 different machines using Java Runtime primitive.

Each node must be able to process command line parameters that are passed to it during execution phase. These command line parameters will provide the port number on which it should start its server, connection details of its successor node in the ring and whether it should generate the very first token for operations to start or it should just wait for the token to come from its predecessor node in the ring.

COMMAND LINE ARGUMENTS LIST:

1. Node ID ex. node1, node2 etc.
2. PORT Number
3. IP Address / Host Name of the successor node
4. PORT Number of successor node
5. true / false : indicates whether this node should generate the original token (true) or not!

Each such nodes are to be started from within an initiator code that you must name `**start.java**'. Note the use of lower-case '**s**' in the name of the Java source code file.

The node operations are very simple. Each node, when it receives the control token, which is receipt of

the message containing token in it, must generate a text message containing its node ID in it and a very simple message and send that message to its successor node in the ring. After this message has been sent, it must forward the control token to the successor node. Once the message circulates the ring and comes back, the message originator must then remove the message from the ring and discard the message. If any other messages are received, the node must simply pass on that message to the successor node.

Each node must print a trace message in the log file that contains the time stamp and any action that it takes upon receiving a message. The log trace message formats will be provided later.

Node Termination:

The initiator node that generated the initial control token must keep a count on how many times the control token has been circulated in the ring. Once the count reaches 5 (five), it must destroy the token i.e. not forward it anymore to the successor node, instead it must send a DESTROY message to its successor node and wait for this message to come back.

Each node when it receives the DESTROY message, must simply forward that message to its successor and after transmission must stop all operations and terminate itself.

When the initial node that originated the DESTROY message receives this message from its predecessor, it must drop the message, stop all the operations and terminate itself. By this time all the nodes in the logical ring topology must have terminated themselves and this would conclude the execution of this project.

start.java Requirements:

Your project must include a configuration file named '**config.ini**'. Note the lower-case '**c**' in the name. This file will contain details of all the 5 (five) hosts where your nodes that form the logical ring will execute. A sample 'config.ini' file format is provided below:

```
#any line beginning with a hash-sign (#) is a comment and must be ignored
node-id: node1
host-name: sun114-31.cise.ufl.edu
port: 41152
successor: node2
generate-token: yes   #indicates that this node will generate the token initially
#any blank lines must also be ignored and any part of the line after # must also be ignored
node-id: node2
host-name: sun114-32.cise.ufl.edu
port: 51122
successor: node3
generate-token: no    #this has be yes for exactly one node. If more nodes have this set to
                      #yes then the first node with 'yes' must become the token originator!
node-id: node3
host-name: sun114-23.cise.ufl.edu
port: 11244
successor-node: node4
generate-token: no
node-id: node4
host-name: sun114-25.cise.ufl.edu
port: 11777
successor-node: node5
generate-token: no
node-id: node5
host-name: sun114-33.cise.ufl.edu
port: 33112
successor-node: node1
generate-token: no
#end of config.ini file ------EOF--------
```

The 'start.java' program must first parse the 'config.ini' file and gather all the necessary parameters that are required to pass to the remote nodes in order to enable them to form a successful logical ring topology.

This program must then start all the remote node programs using Java Runtime API. It must monitor all these remote child processes for their termination. Once all these 5 (five) remote processes have successfully terminated, then 'start.java' module must terminate itself after printing 'END OF PROJECT 1 EXECUTION' message on the console.

Additional Requirements:

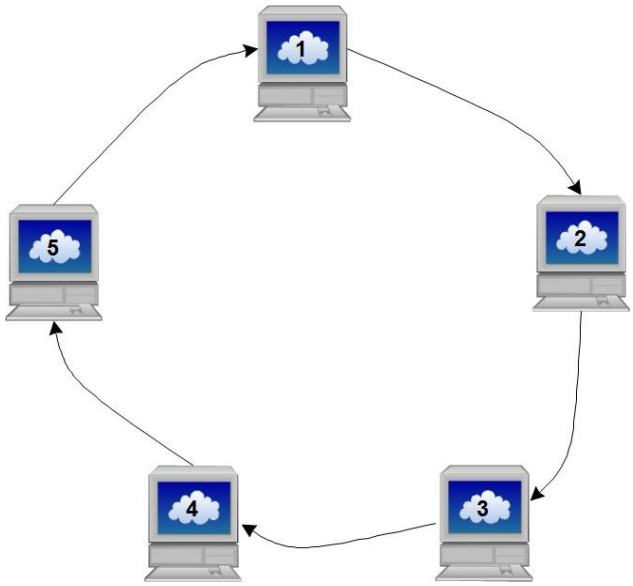
- Remote nodes once started must start their server thread immediately. Main threads must wait for 5 seconds before establishing a connection with their successor nodes. This will give enough time for all remote nodes to be stared from 'start.java' and therefore when they initiate a connection to their successor nodes, will not result in socket exceptions.
- The log files are to have same name as the node ID and with an extension '.log'. These log files are to be created in the current folder where the java binary files reside. An example log file name would be 'node1.log'.
- Remember that remote nodes can not output any messages on the console and if you do so, it may result in execution getting hanged. Take care not to do that. You must output any trace message in the log files only.
- We will use a script file to test your project. Make sure your submission works perfectly with our execution script. **Failure in doing so will result in you getting an initial 0 grade for the project**. **Even after resolution later, you will at-least lose 20% of your grade just for failing this requirement**.
- We will test your submission under CISE sand and rain machines only, therefore make sure your submission works under this environment. Under no circumstance will we use MS Windows to test your project.

Execution Script:

```
#!/bin/bash
rm *.java;
mv *.tar proj1.tar
tar xvf proj1.tar; rm *.class
javac *.java;
read  -p 'Compilation Done. Press Key to start mode 1:'
java start
read  -p 'Execution Done. Press Key to read the report and log files:'
cat ?ep*.??? | more
read  -p 'Press Key to read the node 1 log file:'
cat *1.??? | more
read  -p 'Press Key to read the node 2 log file:'
cat *2.??? | more
read  -p 'Press Key to read the node 3 log file:'
cat *3.??? | more
read  -p 'Press Key to read the node 4 log file:'
cat *4.??? | more
read  -p 'Press Key to read the node 5 log file:'
cat *5.??? | more
```

Before you use this script file to test your project, make sure you have backed up your source code at some other location. We will not be responsible if you lose all your source code after you execute our script. This script is meant to be run with just the project tar file and this shell script located in that directory.

Sample Execution Scenario:



Consider the figure in the left. And assume that the 'config.ini' entries are valid and accurate, node1 will generate the original control token and circulate it to its successor node2 which in turn will pass it on to node 3 and so on.

The actual message formats that are exchanged among the remote nodes are not specified in the project specification and is left for you to design.

The sample log files for nodes 1 and 2 are shown below. Other log files will have similar entries. You must design your project to follow our log file entry formats as closely as possible. Slight variations are allowed.

node1.log sample

```
Execution Started: node1
Parameters received: sun114-31.cise.ufl.edu
Server Thread: Started [OK]
Sleeping for 5 seconds now …
Establishing socket connection to node2 …
Connection Established [OK]

Incoming connection … [accepted]

Generating Control Token: hhagw651ha [random]
Fixing Control Token for the run duration [OK]
Message To Send: "node1 says HI"
Sending to node2 [success]
Transmitting Token to node2 [success]

Incoming Message: "node1 says HI"
Identifying self-message [done]
Discarding Message: "node1 says HI"
Incoming Message: "node2 says HI"
Forwarding Message to node2 [success]
Incoming Message: "node3 says HI"
Forwarding Message to node2 [success]
Incoming Message: "node4 says HI"
Forwarding Message to node2 [success]
Incoming Message: "node5 says HI"
Forwarding Message to node2 [success]
Incoming Token

Incrementing Round Counter +1 : Value 2
Message to Send: "node1 says HI"
Sending to node2 [success]
Transmitting Token to node2 [success]
.
.
.
… and so on [incomplete log file shown here]
```

node2.log sample

```
Execution Started: node2
Parameters received: sun114-32.cise.ufl.edu
Server Thread: Started [OK]
Sleeping for 5 seconds now …
Incoming connection … [accepted]

Establishing socket connection to node3 …
Connection Established [OK]

Incoming Message: "node1 says HI"
Forwarding Message to node3 [success]
Incoming Token

Message To Send: "node2 says HI"
Sending to node3 [success]
Transmitting Token to node3 [success]

Incoming Message: "node2 says HI"
Identifying self-message [done]
Discarding Message: "node2 says HI"
Incoming Message: "node3 says HI"
Forwarding Message to node3 [success]
Incoming Message: "node4 says HI"
Forwarding Message to node3 [success]
Incoming Message: "node5 says HI"
Forwarding Message to node3 [success]
Incoming Message: "node1 says HI"
Forwarding Message to node3 [success]
Incoming Token

Message to Send: "node2 says HI"
Sending to node3 [success]
Transmitting Token to node3 [success]
.
.
… and so on [incomplete log file shown here]
```

Report:

You are required to include a 1 page project report with your submission. It should contain your experience and challenges you faced while doing project1. You may include any additional comments and suggestions to the TA in this report file. Make sure your report file is named '**report.txt**'. Since we will be using UNIX environment to execute your project, we suggest you use a UNIX friendly text editor like pico/nano, vi etc. to write your report. **Make sure the lines in your report does not exceed 80 characters per row.**

Runaway Processes:

Your threads should terminate gracefully. While testing your programs run-away processes might exist. However these should be killed frequently. Since the department has a policy on this matter, your access to the department machines might be restricted if you do not clean these processes.

To check your processes running on a UNIX	`ps -u <your-username>`
To kill all Java processes easily in UNIX	`skill java`
To check runaway processes on remote hosts	`ssh <host-name> ps -u <your-username>`
To clean runaway processes on remote hosts	`ssh <host-name> skill java`

Grading Criteria:

Correct Implementation/Graceful Termination	70
Proper Exception Handling	15
Nonexistence of Runaway processes upon termination	15
TOTAL	**100**

Submission Guidelines:

- Tar all your source code, config.ini and report.txt into a single tar file.
- Name that tar file – proj1.tar
  - on UNIX use `tar cvf proj1.tar <file list to compress and add to archive>`
- Do not keep sub-directories in your tarred structure, keep all files in the same path.
- Non existence of sub-directories is crucial for successful execution of your code with our script
- Test your tar file for successful execution with the provided shell script above.
- Log in to lss.at.ufl.edu portal using your gatorlink username and password
- Go to COP5615 course page and submit your project under assignments section.

DOCUMENT CREATED ON January 13, 2010 at 10:20 AM
Author: Piyush Harsh
Version: 1.0