CS 6378: Advanced Operating Systems
Section 002
Project 3

Instructor: Neeraj Mittal

Assigned on: Wednesday April 6, 2016
Due date: Saturday April 30, 2016

This is a group project. A group must consist of at least two and at most three students. *Code sharing among groups is strictly prohibited and will result in disciplinary action being taken.* Each group is expected to demonstrate the operation of this project to the instructor or the TA.

You can do this project in C, C++ or Java. Since the project involves socket programming, you can only use machines dcXX.utdallas.edu, where XX $\in$ {01, 02, .., 45}, for running the program. Although you may develop the project on any platform, the demonstration has to be on dcXX machines; otherwise you will be assessed a penalty of 20%.

1 Project Description

This project consists of three parts.

1.1 Part 1

Implement a distributed system consisting of n nodes, numbered 0 to $n-1$, arranged in a certain topology (*e.g.*, ring, mesh etc.). At the time of the project demonstration, you will be asked to run your program with a topology specified in a configuration file.

All channels in the system are bidirectional, reliable and satisfy the first-in-first-out (FIFO) property. You can use a reliable socket connection (TCP or SCTP) to implement a channel. *For each channel, the socket connection should be created at the beginning of the program and should stay intact until the end of the program.* All messages between neighboring nodes are exchanged over these connections.

All nodes execute the following protocol:

- Initially, each node in the system is either *active* or *passive*.
- When a node is *active*, it sends a certain number of messages to its neighbors and then becomes passive. To avoid congestion, we will limit the maximum number of neighbors to which a node can send a message when active to maxPerActive. Also, once a node has sent at least maxNumber messages, it cannot become active again. Finally, while a node is active, the minimum delay between its two consecutive send events is given by minSendDelay. Note that the rules allow a node to send more than maxNumber of messages. But, a node cannot become *active* from *passive* after it has happened.
- Only an *active* node can send a message.

- A *passive* node, on receiving a message, becomes *active*.

We refer to the protocol described above as the *REB* protocol.

1.2 Part 2

Implement Juang and Venkatesan's checkpointing and recovery protocol. You can assume that each node takes a local checkpoint in the beginning before it starts executing the REB protocol.

You can simulate crash and subsequent recovery of the system as follows. A node, on failing, simply stops participating in the REB protocol and rolls back to some previous checkpoint selected at random. (All checkpoints taken after the selected checkpoint are discarded.) It then initiates the recovery protocol as described in class. You can assume that there are no additional failures while the recovery protocol is in progress.

The configuration file will contain the sequence of failure events you will be required to simulate at the time of the demonstration. For each failure event, the file will indicate the number of checkpoints a node has to take since the last recovery or the beginning—whatever the case may be—before failing. Note that, once the recovery protocol is finished, you will still need to detect and retransmit lost messages. Nodes terminate once the recovery for the last failure event has been accomplished.

1.3 Part 3

To test the correctness of your implementation of the protocol, implement a vector clock algorithm which a node can use to assign a timestamp to its checkpoint. Whenever an instance of the recovery protocol finishes, each node writes the timestamp of its latest checkpoint to a separate file. Write a tester program to verify that all recovery lines correspond to a consistent global state.

2 Submission Information

All the submissions will be through eLearning. Submit all the source files necessary to compile the program and run it. Also, submit a README file that contains instructions to compile and run your program.

3 Configuration Format

Your program should run using a configuration file in the following format:

The configuration file will be a plain-text formatted file no more than 100KB in size. Only lines which begin with an unsigned integer are considered to be valid. Lines which are not valid should be ignored. The configuration file will contain $2n + m + 1$ valid lines, where n is the number of nodes and m is the number of failure events.

The first valid line of the configuration file contains **five** tokens. The first token is the number of nodes in the system. The second token is the number of failure events to be simulated. The third token denotes the value of *maxNumber*. The fourth token denotes the value of *maxPerActive*. The fifth token denotes the value of minSendDelay (in milliseconds).

After the first valid line, the next n lines consist of three tokens. The first token is the node ID. The second token is the host-name of the machine on which the node runs. The third token is the port on which the node listens for incoming connections.

The next n lines specify the subset of neighbors for each node.

Finally, the last m lines consist of two tokens. The first token is ID of the node that is going to fail. The second token is the number of checkpoints the node needs to take before failing.

Your parser should be written so as to be robust concerning leading and trailing white space or extra lines at the beginning or end of file, as well as interleaved with valid lines. The `#` character will denote a comment. On any valid line, any characters after a `#` character should be ignored.

You are responsible for ensuring that your program runs correctly when given a valid configuration file. Make no additional assumptions concerning the configuration format. If you have any questions about the configuration format, please ask the TA.

Listing 1: Example configuration file

```
5 8 100 3 10

0 dc02 1234  # nodeID hostName listenPort
1 dc03 1233
2 dc04 1233
3 dc05 1232
4 dc06 1233

1 3 4    # space delimited list of neighbors for node 0
0 2 3 4  # space delimited list of neighbors for node 1
1 3      # ...                                    node 2
0 1 2 4  # ...                                    node 3
0 1 3    # ...                                    node 4

1 3              # nodeID numCheckpoints
0 2
1 1
4 5
2 2
0 3
3 2
4 4
```