CS/CE 6378: Advanced Operating Systems
Section 002
Project 1

Instructor: Neeraj Mittal

Assigned on: Monday, August 24, 2020
Due date: Monday, September 21, 2020

This is an individual project. Students can however discuss ideas with each other including algorithm design and program implementation, but *must write their own code. Code plagiarism and/or sharing among students is strictly prohibited and will result in disciplinary action being taken.* We may use plagiarism detection software to detect instances of cheating. Note that copying large sections of code from the Internet is also considered plagiarism.

Each student is expected to demonstrate the operation of this project to the instructor or the TA to obtain *any* credit. You will be graded not only on the correct behavior of your project, but also on your responses to questions concerning your implementation which the instructor or TA may ask during your demo.

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. The topology and information about other parameters will be provided in a configuration file. A process can exchange messages only with processes that are its *neighbors* in the given topology.

All channels in the system are bidirectional, reliable and satisfy the first-in-first-out (FIFO) property. You can implement a channel using a reliable socket connection via TCP or SCTP (preferred). 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.

1.2 Part 2

Implement a *synchronizer* to simulate a synchronous distributed system as follows. All nodes execute a sequence of *rounds*. In each round, a node sends one message to each of its neighbors,

then waits to receive one message from each of its neighbors sent in *that* round and then advances to the next round. Observe that any message received by a node from a "future" round should be buffered until the node has moved to that round.

1.3 Part 3

Design and implement a *distributed* algorithm that uses the synchronizer built in Part 2 to allow *each* node in the system to compute its *eccentricity*. The *eccentricity* of a node is defined as the maximum distance between a node to all other nodes in the topology. Your algorithm can assume that, initially, all nodes know n in addition to their 1-hop neighbors.

2 Submission Information

All the submission 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. You should also include `launcher.sh` and `cleanup.sh` scripts which run your program on the machines as configured and cleanup your processes.

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 $2n + 1$ valid lines. The first valid line of the configuration file contains **one** token that denotes the number of nodes in the system. 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. After the first $n + 1$ valid lines, the next n lines consist of a space delimited list of at most $n - 1$ tokens. The k^{th} valid line after the first line is a space delimited list of node IDs which are the neighbor of node k. 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

```
# global parameter
5

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

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

4 Output Format

If the configuration file is named `<config_name>.dat` and is configured to use n nodes, then your program should output n output files, named in according to the following format: `<config_name>-<node_id>.dat`, where `node_id` $\in \{0, ..., n-1\}$.

The output file for node j should be named `<config_name>-j.dat` and should contain the following: its k-hop neighborhood for $k = 0, 1, \ldots, n-1$ followed by its eccentricity (all on separate lines). An example file is described below.

Listing 2: Example output file for node 4

```
4
0 3
1 2


2
```

In the above example, 3-hop and 4-hop neighborhoods of node 4 are empty.