CS/CE 6378: Advanced Operating Systems
Section 001
Project 3

Instructor: Neeraj Mittal

Assigned on: Friday, November 14, 2025
Due date: Monday, December 8, 2025

This project is a group project. A group can consist of up to two members. *Code sharing among groups as well as copying code from the Internet without permission is strictly prohibited and will result in disciplinary action being taken.* Each group is expected to write their *own code* and 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 project must run on dcXX machines; otherwise you will be assessed a penalty of 20%.

1 Project Description

Implement Koo and Toueg's checkpointing and recovery protocol. Develop a simple application to demonstrate the working of the protocol.

You should design your program to allow *any node to initiate an instance of checkpointing or recovery*. However, you can assume that at most one instance of checkpointing/recovery protocol will be in progress at any time. For example, if node 2 has initiated an instance of checkpointing protocol, then no other node can initiate an instance of checkpointing or recovery protocol until the instance initiated by node 2 has terminated.

You will be provided with a sequence of checkpointing/recovery operations you have to simulate in a configuration file. The sequence will be given by a list of tuples; the first entry in a tuple will denote a node identifier and the second entry will denote an operation type (checkpointing or recovery). As an example, if the list of tuples is (2,c), (1,r), (3,c), (2,c) then your program should execute an instance of checkpointing protocol initiated by node 2, followed by an instance of recovery protocol initiated by node 1, and so on.

Avoiding Concurrent Instances: To ensure that only one instance of checkpointing/recovery protocol is in progress at any time, use the following approach. Once the current instance of checkpointing/recovery protocol terminates, its initiator informs the initiator of the next instance of the checkpointing/recovery protocol (of the termination of the current instance) using simple flooding. The latter then waits for minDelay time units before initiating the protocol. In the previous example, node 2 should inform node 1 which in turn should inform node 3 and so on.

Contents of a Checkpoint: Each instance of checkpointing protocol will have a sequence number associated with it. A node, when taking a checkpoint, stores (a) the sequence number of the checkpointing protocol, (b) the current value of its vector clock (you will need to implement vector clock protocol for this), and (c) any other information you may deem to be necessary for application's recovery.

Testing: Show that your checkpointing and recovery protocols work correctly. Specifically, for the checkpointing protocol, you have to show that the set of last permanent checkpoints form a consistent global state. For the recovery protocol, you have to show that the protocol rolls back the system to a consistent global state.

If Working Alone: Only implement the checkpointing protocol.

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 as well as the names of all your team members.

Only one group member need to submit the files.

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 first valid line of the configuration file contains **two** tokens. The first token is the number of nodes in the system. The second token is the value of minDelay. 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. Finally, the subsequent lines until the end of file will contain the tuples indicating the checkpointing or recovery operation you need to simulate.

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

```
# two global parameters (see above)
5 15

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

(c,1)
(c,3)
(r,2)
(c,1)
(r,4)
```

Any changes to the configuration file format has to be approved by the instructor or the TA.