

# CS/CE/TE 6378: Advanced Operating Systems

## Section 002

### Project 2

Instructor: Neeraj Mittal

Assigned on: Thursday September 29, 2016

Due date: Tuesday, October 25, 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  $XX \in \{01, 02, \dots, 45\}$ , 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

Implement a *mutual exclusion service* among  $n$  processes using Maekawa's distributed mutual exclusion algorithm using two the different techniques for avoiding deadlocks as discussed in the lecture. Your service should provide two function calls to the application: `csEnter()` and `csLeave()`. The first function call `csEnter()` allows an application to request permission to start executing its critical section. The function call is blocking and returns only when the invoking application can execute its critical section. The second function call `csLeave()` allows an application to inform the service that it has finished executing its critical section.

**Implementation Details:** Design your program so that each process or node consists of two separate modules. The top module implements the application (requests and executes critical sections). The bottom module implements the mutual exclusion service. The two modules interact using `csEnter()` and `csExit()` functions.

**Application:** The application is responsible for generating critical section requests and then executing critical sections on receiving permission from the mutual exclusion service.

Model your application using the following two parameters: *inter-request delay*, denoted by  $d$ , and *cs-execution time*, denoted by  $c$ . The first parameter  $d$  denotes the time elapsed between when a node's current request is satisfied and when it generates the next request. The second parameter  $c$  denotes the time a node spends in its critical section. Assume that both  $d$  and  $c$  are random variables with exponential probability distribution.

**Testing:** Design a mechanism to *test* the correctness of your implementation. Your testing mechanism should ascertain that at most one process is in its critical section at any time. It should be as *automated* as possible and should require minimal human intervention. *For example, visual inspection of log files to verify the correctness of the execution will not be acceptable.* You will be graded on how accurate and automated your testing mechanism is.

## 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  $2n + 1$  valid lines.

The first valid line of the configuration file contains one token denoting the number of nodes in the system. The second token is the mean value for  $d$  (in milliseconds). The third token is the mean value for  $c$  (in milliseconds). The fourth token is the number of requests each node should generate.

After the first valid line, the next  $n$  lines consist of three tokens each. 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.

Finally, the next  $n$  lines consist of the up to  $n + 1$  tokens each. The first token is the node ID. The next up to  $n$  tokens denote the IDs of the quorum numbers of the node referred to by the first token. Note that you should not check if the quorum configuration satisfies the intersection property because it may not!

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 20 10 1000

0 dc02.utdallas.edu 1234
1 dc03.utdallas.edu 1233
2 dc04.utdallas.edu 1233
3 dc05.utdallas.edu 1232
4 dc06.utdallas.edu 1233
```

0 0 1 2  
1 1 2 3  
2 2 3  
3 1 3  
4 1 2