Programming Assignment 3 (MP3)
CS 425/ECE 428 Distributed Systems
Spring 2015
Assigned: April 14, 2015
Due: by 7 p.m. on April 30, 2015

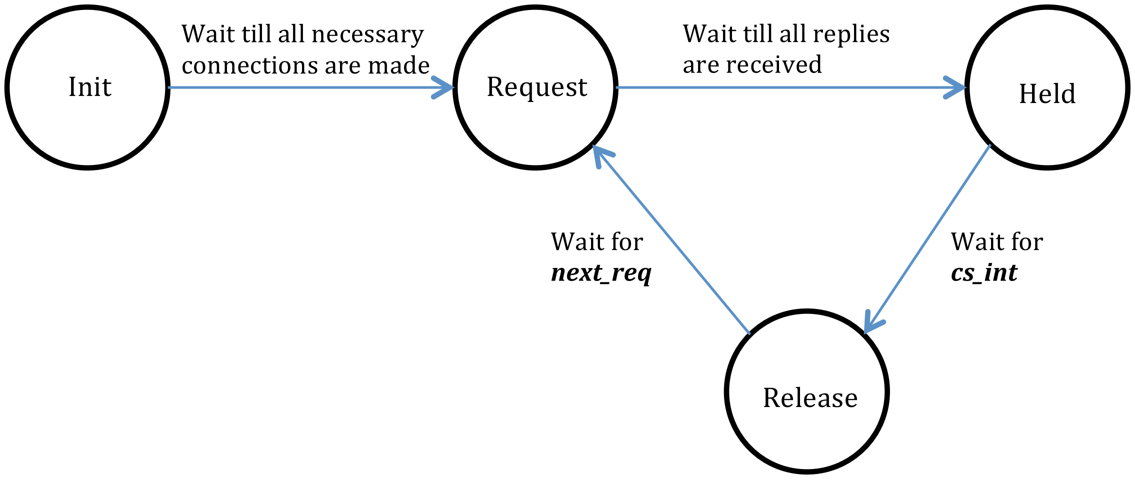
*This **MP must be performed <u>individually</u> by each student. Each student will need to electronically submit their own MP3.***

The purpose of this assignment is to implement a mutual exclusion algorithm *based on* Maekawa's algorithm. Please see section 15.2 of the textbook for Maekawa's algorithm.

Initially, your system will consist of a single process, which will then initiate N=9 threads or N other processes (threads or processes is your choice). These N threads/processes will simulate N nodes performing the mutex algorithm. You may assume that N is fixed (equals 9).

You may replace "thread" by "process" in the discussion below.

Each of the N threads should implement the following state machine.


Init
Wait till all necessary connections are made
Request
Wait till all replies are received
Held
Wait for *cs_int*
Release
Wait for *next_req*

- Each thread will start in the 'Init' state and establish communication channels with other threads.
- Then each thread will go to the next state, named 'Request'. On entering the Request state, each thread will perform the "entry" section for your mutual exclusion algorithm.
- When the thread is allowed to enter the critical section, it will enter the 'Held' state.
- It will stay in the 'Held' state for ***cs_int*** milliseconds.

- Then, it will enter the ‘Release’ state and perform the “exit” section of your mutual exclusion algorithm.
- After exiting from the critical section, a thread stays in the ‘Release’ state for ***next_req*** milliseconds, then it moves into the ‘Request’ state, and continues the state transitions as described above.
- When waiting in any state, each thread still needs to respond to incoming messages.
- Your system should terminate after duration ***tot_exec_time*** in **seconds** (to be specified as a command line input).

Notes:

1. You may implement communication between the different threads by any mechanism of your choice. You may use any available libraries for this purpose.
2. You may define the voting set for each thread in a manner you find suitable, while ensuring that the voting set for different threads are not identical.
3. **Maekawa’s algorithm describes in the textbook does not guarantee deadlock-freedom. You should modify the algorithm to ensure that deadlock will not occur, or the system recovers from a deadlock.**

Input arguments

There will be 3 arguments passed in command line to make grading easier.

1. ***cs_int*** – the time a process spends in the critical section (This is common for all the threads)
2. ***next_req*** – the time a process waits after exiting the critical section before it requests another critical section entrance (This is also common for all the threads)
3. ***tot_exec_time*** – the total execution time for a thread (This is also common for all the threads)

The arguments will be provided in the same order as explained. For instance, for linux, the format is:

```
./mutex <cs_int> <next_req> <tot_exec_time> <option>
```

An example execution will look like

```
./mutex 5 7 20 1
```

For the meaning of <option>, see below.

Submission:

Electronically submit your code & a report. The report should describe the voting set for each of the threads, and *also your mechanism for avoiding/handling deadlocks*. Include instructions for compiling and running your code.

If we are not able to compile your code, then the student will necessarily be asked to do a demo for a T.A.

Grading rubric:

Ensuring mutual exclusion using voting sets	35%
Deadlock mechanism	35%
Report	20%
Miscellaneous (comments, organization of the code)	10%

Output

Each thread should print to the screen the following log whenever it enters the critical section (regardless of the <option> parameter value):

Time Thred_identifier node-list

where Time is the system time at which the thread enters the critical section, Thread_identifier is the identifier of the thread entering the critical section, and node-list is the set of nodes from whom this thread has obtained permission to enter the critical section. node-list should be a list of identifiers separated by spaces.

Additionally, when <option> input parameter is specified as **1**, each thread should also print to the screen a log whenever it receives a message from another thread. The log format should be as follows:

Time Thread_identifier From Message_type other-information

Time is the system time when the log is printed. Thread_identifier is identifier of the thread printing the log. From is the identifier of the thread that sent the message. Message_type should be a string that describes the message type. other-information is any other information about the message that you consider relevant (this could be empty as well).

Additional instructions for logging information may be provided in the next few days.

Student interview:

Each student may be asked to meet a T.A. to discuss their project.