UNIVERSITY of STIRLING

CSCU9V5: Concurrent & Distributed Systems

Assignment 2 -- Distributed Systems

Assignment Outline

This assignment covers material presented during the lectures on distributed systems and builds upon the work in the distributed practicals. The deadline for submitting this assignment is

Tuesday, 26th November 2019 at 09:00

We will use both the 2-hour laboratory sessions to allow you to demonstrate your solution to the given problem.

For the submission you should prepare a single document which includes a report (roughly four pages plus a cover sheet with your student number) discussing the problem, any assumptions you made, and a description of your solution, **as well as** the code listings of your program. Please use appropriate report headings.

The report should include appropriate diagrams of the design and screen shots of your application. Describe the various classes, their relationships, and outline the class methods. The report should explain how complete your solution is, and if applicable, any special cases when your program is not functioning correctly. It is important that your program code is properly commented, and neatly structured for readability. You will lose marks if the code is not sufficiently commented or formatted!

In short, your assignment should consist of (in a single document):

- a cover sheet giving the module, title, and student number
- a document of about 4 pages discussing the problem and your solution
- a printout of your program code with comments

and

- a zip file of your source code (for reference purposes only).

You will receive marks for:

- the efficacy of the code (basic solution) 40%
- advanced features 30%
- the report 20%
- code comments and structure 10%

Assignment Problem

Basic Problem

This assignment will extend the theme of the third distributed laboratory, where you developed a Distributed Mutual Exclusion (DME) system based on a socket-based token ring. In the lab you had a class implementing a *ring node* and a *start manager* class to inject a token to start off the system.

In this assignment you will develop *a socket-based centralised DME.* A given *coordinator* program runs on a JVM on one of the computers in the system. The *coordinator* passes a unique *token* to each single *node* that requests it, sequentially. When granted the unique token, a node will be allowed to execute the critical section and then return the token to the coordinator. All nodes are connected to the coordinator.

You will be issued with a skeleton solution as a starting point. It is strongly recommended that you study the details of the problem and the general functioning of the provided implementation.

Each *node* will be running on a specific *host* and *port* (the port must be instantiated at launch time, e.g. by the command line). Initially all nodes and the coordinator will be running on the same host. Each (non-coordinator) node will perform the following loop:

1. *request the token from the coordinator*. This is done by passing node's ip and port to the coordinator (the coordinator listens on port 7000 at a known ip).
2. *wait until the token is granted by the coordinator.* Granting the token can be implemented by a simple synchronisation by the coordinator on the node's ip:port, analogously to what done for the socket-based token ring.
3. *execute the critical region*, simulated by sleeping for about 3-5 secs, and printing significant messages marking the start and end of the critical section. Important: with multiple nodes running on different windows, it must be evident that only one node at the time is executing the critical session. Adapt (random) timings to facilitate the understanding of programs' execution.
4. *return the token to the coordinator.* This can also be done by means of a synchronisation message (the coordinator listens on port 7001).

The *coordinator* consists of two concurrent tasks that share a buffer data structure:

- a *receiver* that listen for requests from nodes. Requests consist of *ip* and *port* sent through a socket (on port 7000). On connection, the receiver will span a thread (C_connection_r) that receives *ip* and *port*, and store them in the buffer using a *request* data structure, defined in the code.
- a *mutex* thread that constantly fetches requests from the buffer, if any (!), in a FIFO order. For each request, the mutex grants the token to the requesting node, by a simple synchronisation to the node's indicated port. Then, it waits for the token to be returned by means of a synchronisation (on port 7001).

All sockets/servers must be suitably closed. All exceptions' *catches* must print appropriate messages, declaring occurred exceptions, if any - there should be none! All nodes must print suitable messages showing their activities - star and stop of a critical section for nodes, and granting and receiving the token back for the coordinator - at the minimum. Add some random sleeping times and keep them varied: the order of granting the token to nodes should not be fixed. All relevant primitives, e.g. a synchronisation communication, have been seen in labs.

Develop *a socket-based centralised DME* based on the skeleton code provided.

Advanced Features

You can enhance the basic solution by implementing more advanced features:

1. Implement a priority discipline in the request queue. Imagine a critical process that will have priority in acquiring the token over all the other nodes.
2. Consider the case with three high-priority nodes and a standard one. Can you prevent starvation of the standard-priority node?
3. Modify the nodes so that they can deal with coordinator being closed down (or crashing!)
4. (Adapt if needed and) Run the Centralised DME on different computers.

Plagiarism

Work which is submitted for assessment must be your own work. All students should note that the University has a formal policy on plagiarism which can be found at http://www.stir.ac.uk/academicpolicy/handbook/assessmentincludingacademicmisconduct/#q-8.

Plagiarism means presenting the work of others as though it were your own. The University takes a very serious view of plagiarism, and the penalties can be severe.

Specific guidance in relation to Computing Science assignments may be found in the Computing Science Student Handbook.

We check submissions carefully for evidence of plagiarism, and pursue those cases we find. Several students received penalties on their work for plagiarism in past years. Penalties range from a reduced mark, through to no mark for the module, to being required to withdraw from studies.

Submission on CANVAS

Please ensure you submit your assignment on CANVAS before **09:00 on 26th November 2019**. This should be a single file with your report with the source code listings, and a zip of your source files.

Late submission

If you cannot meet the assignment hand-in deadline and have good cause, please see Dr. Andrea Bracciali to explain your situation and ask for an extension through the Extension Request service on Canvas. Coursework will be accepted up to seven calendar days after the hand-in deadline (or expiry of any agreed extension), but the grade will be lowered by 3 marks per day or part thereof. After seven days the work will be deemed a non-submission.