MObile and DIstributed Systems (MODIS) Exam

Exam
9-10 December 2019

Formalia (Read Carefully!)

You must submit on learnit before 14.00 December 10, 2019 a **single** zip-compressed file containing:

- A folder `src` containing the *source code*. You are only allowed to submit source code files in this folder.
- A file `report.pdf` containing a *report* (in PDF) with your answers; the report can be at most 10 A4 pages (font not smaller than 9). The report must contain 4 sections (see below for a detailed specification of the report format). The report cannot contain source code (except possibly for illustrative code snippets).
- Optionally a text file `log.txt` containing log(s).
- If there are any ambiguities, inconsistencies or apparent mistakes, you must state this in your answers, and describe how you interpret these for your assignment.

  Your submission must have been created by you and only you. This applies to both written report, code, examples, logs, etc. Group-work is not permitted in the examination. Your submission must contain, in the report the following statement:

  I hereby declare that this submission was created in its entirety by me and only me.

  The above statement can be included in the beginning of the report and should take no more than 2/3 lines, which count for the total number of allowed pages.

How is my exam going to be evaluated?

Your submission will be evaluated on the following 5 criteria, weighted as indicated:

- 15% Description of your protocol.
- 25% Analysis arguing for correctness of your protocol in the absence of failures.
- 25% Analysis arguing for correctness of your protocol in the presence of failures.
- 20% Other questions.

- 15% Implementation of the design.

Your report.pdf file must contain answers to the first four, each in its own section. Your folder src must contain the implementation.
Note. To pass, you must pass each criteria individually. If any pass is missing, the exam will be marked as failed.

A Distributed Auction System

You must implement a *distributed auction system*: a distributed component which handles auctions, and provides operations for bidding and querying the state of an auction. The component must faithfully implement the semantics of an auction system, and must be resilient to at most one (1) crash failure of any node.

API

Your system must be implemented as some number of nodes, that can possibly be run on distinct hosts. Clients direct API requests to any node they happen to know. Nodes must respond to the following API:

Method	Inputs	Outputs	Comment
`bid`	amount (an integer)	ack	given a bid, returns outcome (fail, success or exception)
`result`	amount (an integer)	outcome	if over, it returns the result of an auction, otherwise the value of the highest bid.

Semantics

Your component must have the following behaviour, for any *reasonable* sequentialisation/interleaving of requests to it:

- The first call to `bid` registers the bidder.
- Bidders can bid several times, but bidding at the n^{th} time must be higher than the bid made at $n - 1^{\text{th}}$ time.
- after a specified timeframe, the highest bidder ends up as the winner of the auction.
- bidders can query the system in order to know the state of the auction

It is part of the exam to *disambiguate* the meaning of "reasonable" and provide the disambiguation as part of your "Analysis arguing for correctness of your protocol in the presence of failures" (see report structure).

Faults

- Assume a network that has reliable, ordered message transport, where transmissions to non-failed nodes complete within a known time-limit.

- Your component must be resilient to the failure-stop failure of at most one (1) node.

Report

Write a report of **at most 10 pages** (Times New Roman, 9pt). The 10-page limit includes text, pictures, any possible references you may wish to include. Moreover, the report must have the following structure (create four sections named with the **bold strings** below – the introduction is optional):

1. **Introduction.** {Optional} A short introduction to what you have done.
2. **Protocol.** A description of your protocol, including any protocols used internally between nodes of the system.
3. **Correctness 1.** An argument that your protocol is correct in the absence of failures.
4. **Correctness 2.** An argument that your protocol is correct in the presence of failures.
5. **Other Questions**:
   (a) An account of whether your implementation uses P2P (is it structured or unstructured?), replication, or something else.
   (b) [BSc students only] An analysis (in English) of the security aspects of your distributed auction system.
   (c) [MSc students only] Imagine, that we want to create an automated bidding agent for premium customers. We would like to implement an extra call in the system called `top`, that automatically increments the current bid by 1.
      i. if no calls to `bid` have been made, a call to `top` is equivalent to calling `bid` 1
      ii. if the highest bid is n, then the result of calling `top` should be equivalent to calling `bid` $n + 1$

   Discuss what consistency means in the context of the auction system, and what challenges are introduced by the adding `bid` to the API.

   Discuss, what kind of concurrency controls can be used, to implement `top`.

Implementation

- Implement your system in Java. Use only the frameworks and libraries provided as standard in the JDK SE and, if you like, the JUnit framework. If you wish to use another (main stream) language, you can do it as far as you do not use non-standard libraries.

- If you implement a running system, demonstrate that it fulfils the semantics and describe to what extent you achieved fault resilience. Do so by submitting a log documenting a correct system run under failures. Your log can be a collection of relevant print statements, that demonstrates the control flow trough the system. It must be clear from the log where crashes occur.
- Submit the log as a separate file, not as part of the report; see Formalia above.
- A partial implementation of your protocol may give partial credits. In this case, use stubs if necessary.

Hints

For the implementation, keep it simple. Do not spend your time on details immaterial to the course, say, pretty-printing debug information.

For the analysis of fault tolerance, try to consider all the situations in-between a client issuing a request and the contacted node returning a reply where a node crashes.

In your report in general, the more you can usefully appeal to and use concepts from the course, the better.