CS 6378: Advanced Operating Systems
Programming Assignment 1

Instructor: Ravi Prakash

Assigned on: February 2, 2024
Due date: February 21, 2024

This is an individual project. Sharing of code among students or using fragments of code written by others is strictly prohibited, and will be dealt with as per the university's rules governing academic misconduct. You are expected to demonstrate the operation of your project to the instructor or the TA.

Requirements

1. Source code must be in the C/C++/Java programming language.
2. The program must run on UTD lab machines (`dc01, dc02, ..., dc45`).
3. You will need to know thread and socket programming and its APIs for the language you choose. It can be assumed that each process (server/client) is running on its own machine (`dcXY`), where $01 \leq XY \leq 45$, and two processes communicate through a reliable socket connection between them. Please get familiar with basic UNIX commands to run your program on *dcXY* and UNIX/Linux system calls for directory and file operations.

Project Description

In this project, you will use socket connections to emulate communication between four computers, say C_1, C_2, C_3 and C_4, and implement causally ordered broadcasting, as described below:

- You will start four processes, one on each of the four computers.
- Each process should have a stream socket connection with the other three processes. You can choose any sequence of steps to set up these connections, so that all four processes know when all connections have been established.
- Once all pair-wise connections are set up, each of the four processes can perform causally ordered broadcasting by executing the following two steps in a loop, for a total of 100 message broadcasts:
  - Wait for a random amount of time in the range (0,10] milliseconds.
  - Send a message to all processes: emulate broadcast as a set of unicasts along all the local socket connections.
- When a message, m, is received by a process, and all its causal predecessors have been delivered locally: (i) m is delivered, and (ii) all messages in *local message buffer* that now become eligible for delivery are removed from the message buffer and delivered. Otherwise, message m is placed in the *local message buffer*.
- Each process should terminate when it has sent its 100 messages, and delivered, in a manner consistent with causal ordering, 100 messages from each of the other three processes.

You are required to report the following:

1. The number of messages delivered at each process.
2. How did you verify that messages were delivered in the correct causal order?
3. Is there any difference in the message delivery sequence at the four processes? If so, how do you explain such a difference? If all processes delivered all messages in the same sequence, why is it so?

Emulating variable network delay

Repeat the experiment described above with the following modification:

- When a message arrives at a destination, it is buffered for a random amount of time in the range [1,5] milliseconds before it is processed.
- Report, if this modification results in any change in the message delivery sequence at the four processes, compared to the previous situation when no destination buffering was employed. If there is a difference, what is the explanation for it?

Submission Information

The submission should be through eLearning in the form of an archive consisting of:

1. File(s) containing the source code.
2. The makefile or corresponding instructions used for compilation purposes.
3. The directories and files you used to test the execution of your code.
4. A PDF document containing your observations about differences, if any, in message delivery sequences and corresponding explanations.

Your source code must have the following, otherwise you will lose points:

1. Proper comments indicating what is being done
2. Error checking for all function and system calls