CS 677 Distributed Operating Systems

Spring 2015

Programming Assignment 2: Internet of Things - Smarter Home Edition

Due: 5pm, Wed April 8, 2015

- You may work in groups of two for this lab assignment.
- This project has two purposes: first to familiarize you with problems such as clock syncronization, logical clocks and multi-tier architectures.
- You can be creative with this project. You are free to use any programming languages (C, C++, Java, python, etc) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed. You can also build on the code that you wrote for the previous lab. You have considerable flexibility to make appropriate design decisions and implement them in your program.

- **A: The problem**
  - In this project, we will implement the IoT gateway using a multi-tier architecture and also use clock syncronization and logical clocks to reason about ordering of sensor events. First add a new type of sensor called the door sensor. The door sensor has two states: open and closed, which indicates whether the entry door is open or closed. While the door sensor can be polled to check its current state, it is primarily a push-based sensor that pushes a notification wheever there is a state chage (e.g., the door is opened or when the door is closed). Your code must still support all sensors and devices from the previous lab assignment. Second, we will make the gateway a multi-tier application with two tiers: the front tier is the application tier that interacts with sensors and smart devices (same as before). The backend tier is a database tier that maintains a database of sensors and their current states as well as the history of prior state changes / events seen by sensors and devices. For simplicity, we will not use an actual relational database. Instead implement the database tier as a separare process from the front-end gateway process. The database process should interact with the frontend tier and maintain its state on a disk file (the file contains data for the "database" and each record is a single line with text entries). You can be creative about the schema for your database and how you store and retrieve data from your file. Be sure not to maintain the entire file in memory since it is important for the data in the database to be persistent. The front-end tier should record state changes into the database history "table" and also update the current state of each sensor in the database when it changes. The front-end tier can also query for data from the history or check the current state from the database. The front-end and back-end processes should interact with one another using RPCs, RMIs or network sockets and the two tiers should be capable of running on different machines. Further, while the API exposed by front-end servers is identical to lab 1, you should implement your own internal interface to handle interactions between the front-end and back-end processes (you can deisgn it any way you wish and this interface should be documented in your README file).

Part 1: Leader election and clock syncronization

Assume that the front-end, back-end, sensor and device processes run a leader election algorithm to elect a time server. You can use any leader election algorithm discussed in class for this purpose. Once a leader has been elected, this node also acts as a time server. Implement the Berkeley clock syncronization algorithm to syncromnize their clocks. Each node then maintains a clock-offset variable that is the amount of time by which the clock must be adjusted, as per the Berkeley algorithm. We will not actually adjust the system clock by this value. Rather to timestamp any request, we simply read the current system time, add this offset to the time, and use this adjusted time to timestamp request. Each push or pull event is now time-stamped with the syncronized clock value; this time stamp is recorded in the database so that the gateway can track when the state change occured.

Part 2: Logical or Vector clocks

While clock syncronization can be used to syncronize clocks and reason about ordering of events, it is also possible to do so using logical or vector clocks. Assume that all sensors and devices as well as the front end server maintain a logical or vector clock. You can either use Lamports clocks or vector clocks for this purpose. Pick either the totally ordered multicast or causally ordered mutlicast algorithm to determine an ordering of all push and pull events. Like before, record the logical/vector timestamp in the database.

Part 3: Event ordering

With these algorithms in place, we can now reason about events and take appropriate actions. First design an algorithm that takes motion sensor and door sensor data to infer when someone entered or left the house. For instance, if motion is sensed first and the door open event happens later, it follows that someone has exited the home. Conversely, if the door open event is seen first and motion is detected later, someone has entered the house.

Of course, your algorithm should determine which event happened before the other event using two methods: physical clock timestamps that syncronized using clock syncronization and logical/vector timestamps to determine event ordering. Log each inferred activity saying "user came home" or "user left home" in addition to logging raw events.

Based on this reasoning, we can automate the process of turning the security system on or off automatically. If the user leaves the home, turn the system from HOME to AWAY and turn on the system. If the user enters the home, turn the system to HOME mode and turn off the security system. Like before, when the user is home, motion detection is used to turn on lights.

While the system can not automatically distinguish between the user process entering the home from the intruder process (and whether to raise an alarm or not), this issue can be addrresed by adding a "presence sensor" which is simply a beacon attached to the user's keychain. When someone enters the home, the presence or absence of such a push events from such sensor can be used to distinguish between the user and an intruder.

Requirements:

1. You need to implement all three parts.
2. All the requirements of Project 1 still apply to Project 2.

- **Other requirements:**

No GUIs are required. Simple command line interfaces and textual output of scores and medal tallies are fine.

You are free to develop your solution on any platform, but please ensure that your programs compile and run on the edlab machines (See note below).

B. Evaluation and Measurement

1. Deploy one sensor of each type and one device and demonstrate that your system works as expected.
2. Meaure the latency and performane of your multi-tier gateway in terms of latency seen at each tier to process a request.
3. Design a test to show your totally-ordered multicasting, clock syncronization algorithm really do the work as you expect.
4. Design tests to show your algorithm to reason about ordering of events to determine when the user enters or leaves the home works as expected.

   Make necessary timeline plots or figures to support your conclusions.

• C. What you will submit

• When you have finished implementing the complete assignment as described above, you will submit your solution in the form of a zip file that you will upload into moodle.
• Each program must work correctly and be **documented**. The zip file you upload to moodle should contain:

1. An electronic copy of the output generated by running your program. Print informative messages when a client or server receives and sends key messages and the scores/medal tallies.
2. A seperate document of approximately two pages describing the overall program design, a description of "how it works", and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). You also need to describe clearly how we can run your program - if we can't run it, we can't verify that it works.
3. A program listing containing in-line documentation.
4. A seperate description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.
5. Performance results.

• D. Grading policy for all programming assignments

1. Program Listing
   works correctly ---------- 50%
   in-line documentation -------- 15%
2. Design Document
   quality of design and creativity ---------- 15%
   understandability of doc ------- 10%
3. Thoroughness of test cases ---------- 10%
4. Grades for late programs will be lowered 12 points per day late.

- **Note about edlab machines**

- We expect that most of you will work on this lab on your own machine or a machine to which you have access. However we will grade your submission by running it on the EdLab machines, so please keep the following instructions in mind.
- You will soon be given accounts on the EdLab. Read more about edlab and how to access it here
- Although it is not required that you develop your code on the edlab machines, we will run and test your solutions on the edlab machines. Testing your code on the edlab machines is a good way to ensure that we can run and grade your code. Remember, if we can't run it, we can't grade it.
- There are no visiting hours for the edlab. You should all have remote access to the edlab machines. Please make sure you are able to log into and access your edlab accounts.
- IMPORTANT - No submissions are to be made on edlab. Submit your solutions only via moodle.

Stumped?

1. Stumped on how to proceed? Better yet, ask the TA or the instructor by posting a question on the Piazza 677 questions. General clarifications are best posted on Piazza. Questions of a personal nature regarding this lab should be asked in person or via email.