CS 677 Distributed Operating Systems

Spring 2015

Programming Assignment 1: Internet of Things and Smart Homes

Due: 5pm, Friday March 6, 2015

- You may work in groups of two for this lab assignment.
- This project has two purposes: first to familiarize you with sockets/RPCs/RMIs, processes, threads; second to learn the design and internals client-pull and server-push systems as well as client-server systems.
- 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 have considerable flexibility to make appropriate design decisions and implement them in your program.

- ## A: The problem

  - Internet of Things (IoT) broadly refers to sensors and "smart" devices with network capabilities that can be employed to design various applications. Here we will use IoT concepts to design a distributed system that can be deployed in a "smart home" to automate simple tasks.

    We will assume three classes of IoT objects:

    1. *Sensor*, which can sense surrounding conditions and report it.
    2. *Smart Device*, which can be remotely turned on or off
    3. *Gateway*, which is a tiny server that interacts with sensors and smart devices.

    The goal of the assignment is to build a distributed system with two sensors, two smart devices, and one central gateway that controls these sensors and devices. We will use this system to implement two automation tasks in the smart home.

    Assume two types of sensors: a temperature sensor that reports the current temperature, and a motion sensor that reports whether it senses motion in the surrounding environment. The temperature sensor is a pull-based sensor, where the gateway must send it a query and it responds by sending the current temperature. The motion sensor is a push-based sensor where it pushes a notification to the gateway whenever it senses motion and does nothing when there is no motion.

    Assume two types of smart devices: a smart light bulb and a smart outlet. Both smart devices have remote control capabilities, which means that in addition to being turned on or off normall by users, they can be remotely controlled by the gateway and turned on or off through commands sent by the gateway. Both smart devices need to support query capabilities where they are queried by the gateway about their current state and they respond whether they are currently on or off. Both also support control capabilities where the gateway can send a command to change the state from on to off, or off to on, and they perform the action and send an acknowledgement.

The gateway, sensors and devices the following interfaces:

- *register(type, name):* initially all sensors and devices must register themselves with the gateway. The type can be "sensor" or "device" and the name can be one of "temperature", "motion", "bulb" or "outlet" Sensors only have query or reporting capabilities, while devices can be qeuried as well as actuated. Upon registration, all sensors and devices are given a global ID, which is simply a unique integer.
- *query_state(int device id):* all sensors should support pull-based query where they receive a query_state request from the gateway and report their current state. A temperature sensor reports state as the current temperature; a motion sensor reports state as yes or no, while devices report state as on or off. The reply to query state should contain two parts: the device id and the reported state.
- *report_state(int device id, state):* any push-based sensor or device can push its current state to the gateway without being asked via the query_state interface. This is achieved via report_state call to the gateway. The sensor or device includes its device id and current state. The interface can be used to implement a push sensor where its reports its state whenever a change is observed (e.g., motion can be reported when the state changes from no motion to motion, or absence of a motion is reported when motion has stopped).
- *change_state(int device id, state):* any smart device with control capabilities also supports the change_state command where the gateway can specify the state as "on" or "off" and it changes its state to the specified state.

Assume that the smart home has one temperature sensor, one motion sensor, one smart light bulb, and one smart outlet. In addition, there is central gateway that interacts with all sensors and devices. Given this setup, use this basic capabilities of sensors and smart devices to implement two automation tasks.

Task 1: Preventing water pipe bursts: The winter has seen many snow storms and artic colds and you are worried that the cold temperature may cause the water pipes to freeze and burst causing your home to flood. In order to prevent water pipe from freezing, you decide to buy a pipe heater, which is simply a small heater that you plug into a power outlet and it keeps the area near water pipes warm and avoid freezing of water pipes. However, rather than keeping the heater on all winter, which is a waste of electricity, you deploy a temperature sensor near the pipes and plug the heater into a smart outlet. When the sensed temperature drops below a certain value, the heater is turned on by the gateway actuating the smart outlet. When the temperature rises above a second threshold, it is turned off. Assume that the heater needs to turn on whenever the temperature drops below 1 celsius and is turned off when the temperature rises above 2 celsius. This way, the heater is on only during nights when the temperatures drop below zero and is turned off during day hours when the temperature rises a little above zero.

Task 2: Preparing for spring break: Your home also has a motion sensor and a smart bulb. Both are deployed in living room and you have configured the gateway to receive push notifications from the motion sensor each time it sees motion or stops seeing motion. Upon seeing motion, the gteway turns on the light bulb and when it no motion is reported for more than 5 minutes, the bulb is turned off automatically. In this case, whenever you walk into the living room, the light turns on and once you leave the room, the light turns off after five minutes.

Since you are tired of the long winter, you decide to visit a friend in a warmer location for spring break. Since you are worried about the security of your home, you decide to extend this setup to implement a simple security system. In this case, whenever you are away from the home, if motion is sensed in your living room, an alert is sent to the user indicating a possible intruder (lights are not turned or off when you are away). When you are home, the presence of motion is used to turn the light on or off and no alerts are sent. In this case, the gateway supports an additional command *change_mode(mode)* where the mode can be HOME or AWAY. The user process can set the mode to AWAY when leaving for spring break or anytime when they leave their home for any purpose (causing the security system to be activated). The user process sets the mode to HOME when entering the home, causing the security system to be disabled and motion-actived lighting to be activated instead. The user process can also receive a *text_message(string)* which is used to send notifications or security messages.

Implement each sensor, device, gateway and the user as separate processes that are capable on running on different machines. All entities communicate with one another via the above interfaces. This is essential a distributed client-server application where the gateway is the central server entity. The gateway process should support concurrency where it should be capable or interacting with multiple sensors or devices at the same time.

- **Other requirements:**

  IF you use a multi-threaded server, be aware of the thread synchronizing issues to avoid inconsistency, race conditions or deadlock in your system.
  No GUIs are required. Simple command line interfaces and textual output of scores and medal tallies are fine.

B. Evaluation and Measurement

1. Deploy 2 sensors, 2 devices, a gateway and the user process. They can be setup on the same machine (different directories) and have the gateway make several requests and provide the output as part of your submission.
2. Next deploy the entities such that at least some of the processes are on different machines and demonstrate that your system works over a network for both client-pull and server-push modes. 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).

3. Conduct a few experiments to evaluate the behavior of your system under different scenarios. Change the rate at which updates are requested or events are generated,and measure the latencies to get a reply. Measure the latency of server push (motion) sensors and client pull (temperature) sensors Report any insights, if any, from your experiments, as to which strategies succeed the best.

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 separate 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 separate 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. What is the Internet of Things? Read about it at on Wikipedia or "ask Google"
2. Not sure how to proceed? 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.