Project 2[1]

Given out: November 2nd, 2015
Due: November 17th, 2015, 11:59 PM Via Blackboard.

In this project, we will implement the IoT (Internet of Things) gateway using a ***multi-tier architecture*** and also use 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 whenever there is a state change (e.g., the door is opened or when the door is closed).

Second add another sensor called **Motion Detection Sensor** which has two states: **True or False**. Add one more sensor **KeyChain** which also has two states: **True or Fals**e.

Third, we will make the **gateway** a multi-tier application with two tiers.

Backend Tier

The back-end 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 separate process from the front-end gateway process. The database process should interact with the front-end 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**). Be sure not to maintain the entire file in memory since it is important for the data in the database to be persistent. Each row in the table should contain the following information in order.

SensorID,SensorType,SensorState,TimeStamp,IPAddress,PortNumber.

where

ID: The integer value the sensor/device had been assigned
Type: Door/Motion/Keychain/SecurityDevice
State: (Open/Close: for door & True/False: for Motion & KeyChain
On/Off: for Security Device)
TimeStamp: Seconds since Jan 1, 1970 (Return value of ***time(NULL)*** in C)
IPAddress: IP Address of the sensor
PortNumber: Port number of the sensor.

For example, below is a sample. For more detailed samples refer to SamplePersistentStorage.txt attached.

Sample
1----Door----Open----1445850587----10.23.33.44----1034

[1] Acknowledgement to Dr. Prashant Shenoy's CS677 distributed systems course from which this project has been adapted.

Front Tier

The front tier is the application tier that interacts with sensors and smart devices (same as in the last project). It 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 network sockets and the two tiers should be capable of running on different machines. You can decide the message format and commands. You should implement at least an “insert” command. Be sure to mention the Format of the supported messages in the README.txt

Part 1: Logical or Vector clocks

While clock synchronization can be used to synchronize 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 Lamport's clocks or vector clocks for this purpose. Use the causally ordered multicast algorithm to determine an ordering of all push and pull events. Record the logical/vector time-stamp in the log file.

Part 2: 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 using the 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. Gateway does this reasoning and output with timestamp at which that happens.

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.

While the system cannot 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 addressed by adding a "presence sensor" which is simply a beacon attached to the user's key-chain. When someone enters

the home, the presence or absence of such events from such sensor can be used to distinguish between the user and an intruder.

Sensors

There are three different sensors in the system.

1. **Motion Sensor**

The sensor will sample every 5 seconds and sends True or False depending on whether there is a motion or not.

Gateway on receiving the message will send Insert request to the back-end which will store the information to the database.

SensorType: Motion

SensorState: True/False

Input 1: MotionSensorConfigurationFile

Input 2: MotionSensorStateFile

Input 3: OutputFile

2. **KeyChain Sensor**

This sensor will sample every 5 seconds and sends True or false depending on whether the key chain is present inside the room or not. You can assume that the sensor will detect the presence only when the keychain is inside the room.

Gateway on receiving the message will send Insert request to the back-end which will store the information to the database.

SensorType: KeyChain

SensorState: True/False

Input 1: KeyChainConfigurationFile

Input 2: KeyChainStateFile

Input 3: OutputFile

3. **Door Sensor**

This is a push-based sensor which will send message to the gateway only when the door is opened or closed. It will **NOT** sample during every interval. That is when the door is just opened, it will send Open message to gateway. Then when it closes, it will send close message to the gateway.

Gateway on receiving the message will send Insert request to the back-end which will store the information to the database.

SensorType: Door

SensorState: Open/Close

Input 1: DoorConfigurationFile

Input 2: DoorStateFile

Input 3: OutputFile

Device

1. **Security System**

The security system will get input from Gateway if it should be on or off. After switching off or on, send message to Gateway specifying the current state. The gateway will then record it to the back-end DB.

DeviceType: SecuritySystem

DeviceState: on/off

Input 1: GatewayConfigurationFile

Input2: OutputFile

Gateway (Front Tier)

Gateway should decide whether the security system should be on/off according to the values it received from different sensors and according to the logic mentioned above. Also on reception of each message from the entities, it should send insert messages to the database corresponding to each of them.

Input 1: GatewayConfigurationFile

Input 2: OutputFile

Back-End (Database)

The back-end will just store all the data in the format mentioned above. Make sure that the data gets updated to the persistent storage file frequently.

Input1: BackEndConfigurationFile

Input2: PersistentStorageFilePath

Configuration Files Syntax

For GatewayConfigurationFile only the Line2 exist.

Line 1: GatewayIPAddress,GatewayPortNumber

Line 2: CurrentIPAddress,CurrentPortNumber

StateFile Syntax (for Motion & KeyChain)

Line 1:<start-time 1>;<end-time 1>;value 1

Line 2:<start-time 2>;<end-time2>;value 2

.

.

Linen:<start-time n>;<end-timen>;value n

where (start-time n) = (end-time n-1) + 1

(start-time and end-time inclusive)

StateFile Syntax (Door)

Line 1:<time 5>;Value@5

Line 2:<time 7>;False@7

.

.

Line n:<time i>;Value@i

Where values at 2 consecutive lines will not be the same

Notes:

1. On booting up the system, the registration to the gateway should happen first.
2. Assume that there is only one MotionSensor, one KeyChain, One Door, 1 Device, I Gateway and 1 Back-end.
3. Start the system only after every sensor gets registered to the gateway and store its state in the backend.
4. Back end's persistent storage (last input parameter file) should be updated frequently
5. Keychain and Motion Sensor sends status messages every 5 seconds
6. Door Sensor is a push device, as mentioned before
7. The final parameter of each of the component in the system is an output file to which each component will log its activities with time stamp.
8. Each sensor, device, door, Gateway and Back-end should print the messages it send and received during each of the operations. Also print the time time-stamp and vector clocks during each event. There is no fixed format for printing the output of sensors and devices. But Each event should be in a separate line.
9. When logging events at each component, make sure to be precise. But don't forget to log all important events. Please **don't** be more verbose like "The Motion Sensor is on at time 234534234". But choose formats like "Motion Sensor : on : 234534234" or similar.

Submission Procedure

Submit a Zip file (not tar or tar.gz files) with the following contents with the following exact structure. Please make sure all the naming conventions are strictly followed. Upload the zip. **You code should execute on a Linux (possibly Ubuntu) machine. Please test your code on a Linux machine (or virtual machine). You can work in groups of two but only one of the group members need to submit the project. We will share a google doc where we will be asking for the group members.**

Instructions.txt: It should contain the details about how to compile and run the different files

README.txt: Should contain any assumptions made during the project

compile.sh: This file should contain the shell script to compile the different files inside the different folders and put their corresponding outputs in the output folder. For each folder inside src folder, there should be a corresponding binary in the output folder. That is, after running compile.sh, the output directory should contain 6 files "door", "motion", "keychain", "securitysystem", "gateway" and "database"
For example, if you are using gcc to compile the shell script can contain the following lines. The below lines are just samples and for illustration purpose only.

```
gcc -pthread -o output/sensor src/Door/door.c
gcc -pthread -o output/motion src/Motion/motion.c
gcc -pthread -o output/keychain src/KeyChain/keychain.c
gcc -pthread -o output/securitysystem src/SecuritySystem/securitysystem.c
gcc -pthread -o output/gateway src/Gateway/gateway.c
gcc -pthread -o output/database src/Database/database.c
```

Zip File Structure

ZipFile

- Instructions.txt
- README.txt
- compile.sh
- output/
  - <nothing need to be inside this folder>
- SampleConfigurationFiles/
  - <sample configuration files for all the entities>
- src/
  - Door
    - <src files for Door Sensor>
  - Motion
    - <src files for Motion Sensor>
  - KeyChain
    - <src files for Key Chain Sensor>
  - SecuritySystem
    - <src files for Security System >
  - Gateway
    - <src files for Gateway>
  - Database
    - <src files for Database>

Attachments: (The IP Addresses & Port numbers are mentioned for illustrative purpose only. Please use IP Addresses which make logical sense)

SampleGatewayConfigurationFile.txt: Gateway Configuration file sample
SamplePersistentStorage.txt: Persistent storage format of Back end component
SampleSensorInputFile1.txt: Sample Input file for Keychain and Motion Detector
SampleSensorInputFile2.txt: Sample Input file for Door Sensor
SampleSmartDeviceConfigurationFile.txt: Sample Configuration file for Security System
SampleSensorConfigurationFile.txt: Sample Configuration file for Keychain, Motion Detector and Door sensors