Programming Assignment 2: Internet of Things - Smart Home Edition

DESIGN DOCUMENT

Submitted by: - **Rahul Raj and Olenka Dey**

---

1. Introduction

The main aim of this assignment was to understand the working and implementation of IoT using distributed client-server architecture and to develop simple security system for a smart home. For this assignment, we have constructed three sensor nodes: motion sensor, door sensor and temperature sensor, two device nodes: smart bulb and smart outlet and a multi-tiered, central gateway. Gateway is the central server with which all the sensors and the devices are connected. Gateway keeps track of all the activities and implements security system. We have used Remote Procedure Calls (RPCs) for communicating between processes and have also used events, locks and threads. Finally, we implemented a simple security system that predicts the events based on the occurrences using logical clocks and "raise alarm" if necessary.

Following is the file structure which we have implemented.

--"**eventLog**" – This is a basic text file which contains all the events that a user or a gateway will perform during the run. It contains two tags: "User" and "Gateway" for each event to redirect them to their respective processes for initiation. It also indicates the presence sensor, "PS" with the door activities for security system.

--"**run_ me.sh**" – Top most module. Used to start all the processes. Every node has two files associated with it.
Suppose the node name is "motionSensor", it has two files:

|---"**motionSensor.py**" – It initiates the application of the sensor that responds to user activities like enter or exit and to the gateway when it queries something. This waits until it gets any signal from its leader election process.

|---"**leaderElection_motionSensor.py**" – It runs continuously at the backend. This file implements leader election algorithm among all the processes in the system and notifies the process at its application level when a leader is being elected. And it also re-initiates the election if the existing leader gets disconnected or goes off the grid.

Similarly, every node has these two files. Gateway is a multi-tiered server which contains frontend and backend. Both frontend and backend have their similar files respectively. Additionally, it has "user" process. Therefore, it contains 8 "nodeName.py" and 8 "leaderElection_nodeName.py" files.

|---"**timePlot.py**" – This file plots the events based on the lamport's clock reported.

2. How to Run

Running the application involves 2 steps:

1. eventLog File – Look into this file to have an idea of how events are mentioned and check the possible general and testing events. Modify them accordingly for any random sequences and save.

2. running run_me.sh in the terminal. This will start all the application and leader election processes of each node and no need for the user to open in n different terminals, as all the messages will be printed in the single terminal for the user convenience. Use the following command to run it.

```
./run_me.sh
```

This command should work, if it doesn't work you have some permissions issue, use **chmod 775 run_me.sh** to give the desired permission to this shell script. Please look into the Output/Results document for the different results. To check if leader election starts again when the leader dies, get the process id using the command, **ps -eaf | 'grep python'** and kill that process using command, **kill -9 process_id.**

3. Design and Implementation

Design:

We choose Python as the programming language for this lab. We have used simpleXMLRPC for the all nodes in IoT implementation. We have written a shell script which starts all the nodes together. Following are few of our design considerations.

- Random events can be given to the system using the eventLog File. It can be either the user or gateway initiated activities. We have added tags with each activity to differentiate between them
- For testing the security system if it detects user/intruder 'entry' or 'exit' properly, we added two user events in the eventLog file. Those events are: "user: enter" and "user: exit"
- Motion senor and door sensor is designed as a push-pull architecture whereas the temperature sensor is only pull based architecture.
- Smart bulb and smart outlet is also a push-pull implementation as user can also control manually and in that case, the devices must push their current state to the gateway.
- Gateway is a two-level server where all devices and sensors connect only to the frontend and frontend connects to the backend to maintain database of all the activities and their current state.
- Every node is given a global Id when it registers itself to the gateway.
- Every node has their associated leader election process that runs in background continuously.
- We used "Ring topology" to implement leader election algorithm and "Berkeley algorithm" for clock synchronization.
- The implemented leader election algorithm is able to re-elect new leader if the existing leader dies.
- Clock synchronization happens every time a new leader is elected.
- We have used causally ordered multicast algorithm "Lamport's Logical Clock" for determining the ordering of push and pull events.
- We keep track of the event ordering and verify it by plotting those events based on their logical clocks.

Implementation:

We have implemented interfaces asked in the lab requirement documents. Implementation of the IoT system is according to the structure discussed in the "Introduction" section.

1. ***register(type, name)***: This is implemented in gateway "frontend.py". Sensor/devices register themselves by calling this function. It is only called once when the system starts. On registering the devices and sensors with the frontend and unique ID is assigned to each of the sensors and devices.
2. ***query_state(int device id)***: This function helps the gateway frontend to know the current state of different devices and sensors. It is also implemented in gateway frontend.py.
3. ***report_state(int device id, state)***: Whenever there is a change in state of the devices/sensors it is reported back to the gateway frontend which in turn logs those changes to the gateway backend file called 'history.csv'. For example, if a user enters the house, the state of the door, motion sensor and the bulb device changes and these changes will be automatically reported to the front end for logging.
   TempSensor is the only sensor which is pure pull based sensor so the gateway can anytime request the state of this sensor.

4. ***change_state(int device id, state):*** Users and Gateway (frontend) have the ability to change the state of the smart devices. And the state after being changed is reported back to the gateway automatically.

As an output, the system also generates three files:

a. current_status.txt: it tells you about the latest state of each of the sensors/device.
b. history.csv: It keeps track of all the state changes in that happened in the sensors and devices during the activities.
c. LamportEventLog.txt: This file is to register all the events and their respective clock value. This file is to verify the event ordering. Here first two columns indicates the source information and next two column indicates the destination information for a particular event. One can verify the sequence of events with the actual events using this output file.

Tradeoff and Limitations

- We have fixed the Id for each node in the leader election process so that communication becomes little easy by directly connecting to the given id. We could improve that by selecting the id generating by the gateway and use them for connection. In that way, we could randomize the leader selection process.
- One tradeoff it that we are keeping track of maximum up-to neighbor's neighbor in the ring topology. We can extend it by simply adding more neighbors to the neighbor's list.