**COP 5615: Distributed Operating Systems Principles**

**Internet of Things Support in Xinu**

**Fall 2016**

**Term Project Report**

**Group 6**

**Yuan Zhou, Jianing Cui, Zhixin Pan, Nimish Jindal, Natasha Mandal**

[**zhouyuanzxcv@gmail.com**](mailto:zhouyuanzxcv@gmail.com)**,** [**cjnseu@gmail.com**](mailto:cjnseu@gmail.com)**,** [**panzhixin@ufl.edu**](mailto:panzhixin@ufl.edu)**,** [**nimishjindal.dce@gmail.com**](mailto:nimishjindal.dce@gmail.com)**,** [**natasha.mandal.93@gmail.com**](mailto:natasha.mandal.93@gmail.com)

**1. Describe your project using this table**

|  |  |
| --- | --- |
| **Part** | **Indicate Completeness (give a no. from 1-10), followed by Description** |
| Xinu I/O Interface design | 10.  We modify the Configuration file such that the new device drivers can be used in the device switch table. When we call read or write from Xinu, it will call the corresponding device driver read or write. The device id is automatically generated in the conf.c and conf.h files given the Configuration file. We added 4 types of devices in this file, tmp36, led, button, and buzzer. The devices tmp36, button only accept read. The devices led, buzzer only accept write. We added some interfaces as follows  extern devcall tmp36read(struct dentry \*devptr, char \*buffer, int32 count);  extern devcall ledwrite(struct dentry \*devptr, char \*buffer, int32 count);  extern devcall buttonread(struct dentry \*devptr, char \*buffer, int32 count);  extern devcall buzzerwrite(struct dentry \*devptr, char \*buffer, int32 count);  extern syscall sendEdge(char\* dip,uint16 dport,char\* msg,uint16 len);  extern syscall recvEdge(char\* msg,uint16 len);  where the first 4 are related to the devices, the last two are related to the communication between the BBB and the edge. |
| IoT-specific concerns your design addressed, including but not limited to Energy | 8.  In our design, we have one cloud server, which hosts the MQTT server and the dashboard. We have 3 edges, one is connected to the BBB with tmp36 (the temperature sensor), one is connected to the BBB with 2 buttons, one is connected to the BBB with a LED and a buzzer. The BBB with tmp36 will send the temperature data every 1 second to the edge, which will publish the data to the MQTT server. The buttons will be pressed to toggle the LED and the buzzer. The energy is consumed by 3 BBBs, 3 edges, 1 server. The BBBs consume least energy.  MQTT is a protocol in which the subscribers receive a message only when information is published to a topic it has subscribed to. This decreases the server to edge interaction.  For the buttons, we tried to minimize the device to edge interaction by sending a message to the edge only if the button is being pressed.  For the LED, buzzer, the main program waits on message from Edge. Only when the message is received it toggles the state of respective device, and sends its status to the edge. |
| Xinu I/O Interface implementation and testing | 10.  For the interfaces change in the read/write function where additional 4 types of device id can be inputted, TMP36, LED, BUTTON, BUZZER, we implemented it by changing the Configuration file to include these devices.  For the added interfaces relating to the 4 devices, they are generated according to the DDL XML files.    For the sendEdge and recvEdge system calls. They are use the udp library provided by xinu. Before udp\_send and udp\_recv, first we use udp\_register to ask for a slot. Then use this slot to do the communication. After the communication, release the slot. |
| Design of IoT Description Language, Language processing and code generation | **Indicate**: XML-, JSON, Other-based  **Source**: any open source used? Indicate the Github or other s/w package name.  **Design:** We use XML as the DDL. We use libxml2 for parsing the XML file as it is implemented in C. In the Makefile, we add  **@(cd $(TOPDIR)/ddl; make; ./ddl\_parser tmp36 led button buzzer)**  in a recipe where the folder ddl contains the makefile for creating the “ddl\_parser” from the “ddl\_parser.c”. The “ddl\_parser.c” uses libxml2 to parse the 4 DDL XML files. Then, it will generate the high level device drivers accordingly.  The DDL is designed to contain 3 parts, description, interaction, interface. The interface part describes whether we use the analog or digital pin, the pin number, and the formula to generate the device driver. The interaction part describes how this device interacts with the outside. It contains various IP addresses, e.g. server address, edge address, BBB address, whether it is shown in the dashboard, the topic to publish its state, and the topics that it should subscribe to.  The DDLs are read when compiling the Xinu for BBB, running the program in the edge computer, and showing the dashboard in the cloud.  Reference for GPIO interface: <https://github.com/embest-tech/AM335X_StarterWare_02_00_01_01> |
| Implementation and testing of IoT Description Language, Language processing and code generation | 9.  In the “ddl\_parser.c”, it handles multiple input DDL XML files. For each one, it parses the file and extracts node values, then it prints to a target .c file in the device/ext folder. The used node values include digital/analog, pin number, computation expression, etc. The generated device drivers correspond to the four devices and are placed in the device/ext folder. We tested it by printing the read/write in the main files and sending the read data to the edge, which will also print the received data. |
| Implementation and testing of overall on-board driver code (upper- and lower-level drivers, including generated code) | 10.  The read for TMP36 is implemented by adding an additional device as ADC, which handles the low level manipulation of registers of the CPU. Following the Chapter 12 of the CPU datasheet, we set the first Step Config and use FIFO0 to read the voltage of the AIN pin. In the device driver of tmp36, we convert the voltage to temperature according to the formula in the DDL XML file. We tested it by printing the temperature in the main file. Also, we use a finger to touch the sensor to see that the temperature goes up.  For the button driver, we extended the GPIO interface to access the GPIO pins on the BBB (since it is a digital sensor and this information is dervied from the DDL). In the generated driver, we set configure the clock for GPIO, set up the pin (derived from the DDL), set the pin’s direction to take input from it (specified in the DDL) and read data from the DATAIN register at the specified pin.  The LED, buzzer device drivers are built over underlying GPIO interface. Device initialization involves a) enabling the functional clocks of given GPIO instance b) performing a pin multiplexing and c) setting the pin direction. Write method was defined to set pin voltage to HIGH or LOW. This were tested to on/off LED from Edge computer. The write method was replaced to be generated automatically from device DDL and re-tested. |
| Did you use the same existing device driver structure and mechanisms in Xinu? | Yes or No. If No, explain.  Yes. We changed the Configuration file such that additional devices can be added to the generated “conf.c” and “conf.h” files. They will be added to the device switch table. |
| Approximate % driver code generated with respect to overall on-board driver code | Give % only here, no description.  5%  Assuming that the code for functions in separate files and our drivers call those functions, we can give the following breakdown:  TMP36 – 5%  Button – 5%  If we do not consider auto-generation of low level implementation (data in/out from registers), we could fully generate from DDLs - the upper level device drivers for devices. |
|  |  |
| Which device externalization abstraction have you chosen (which existing technology or any new ideas)? You may, or may not explain the reason for your choice. | We use the DDL to represent the device in the edge, and in the cloud. The edge reads the DDLs and know that which topic it should publish to, and which devices it will depend on (which topic to subscribe). The cloud also reads the same DDLs to know which devices should appear in the dashboard, and which topic does each device corresponds to, hence update the data in the callback. |
| How, where, and when do you specify the edge and cloud addresses of the device? Explain how device configuration and initialization are done including device externalization. | The devices are externalized by the DDLs. The DDLs include various IP addresses, including BBB, edge, server addresses. In the BBB level, the tmp36 is connected to a BBB through a breadboard, its data pin is connected to an analog pin, the two buttons are connected to another BBB through a breadboard and some resistors, the LED and buzzer are also connected to a BBB. The devices are initialized in the in the low level code. In the edge and the cloud level, the devices are represented by the DDLs, the edge and the cloud follow the MQTT protocol to communicate. The DDLs specify the topics that they should publish to or subscribe to. |
| Give the details of the externalization abstractions design. | We use the Cloud-Edge-Beneath architecture. Please see the attached “part2\_architecture.jpg” file for the design. In summary, we use 3 BBBs, 3 edges and 4 devices. BBBs control the devices and communicate with the edges, which communicate with the cloud by mqtt. The devices are depicted in the XML files. Since the cloud and the edges all read the XML files. They will know the interaction among these devices. For example, given the following part of the XML for the LED  <Interation>  <Topic>LED</Topic>  <Dashboard show='1'>  <Format>binary(on/off)</Format>  <Unit></Unit>  </Dashboard>  <BBB\_Address>192.168.1.101</BBB\_Address>  <Edge\_Address>192.168.1.104</Edge\_Address> <Server\_Address>128.227.170.197</Server\_Address> <Dependence\_Topic>Button</Dependence\_Topic>  </Interation>  The dashboard in the cloud will know that it will show it in the dashboard, also it should subscribe to the topic “LED” to update it. The edge knows that the BBB that it communicates with has a device called LED, hence the data it received should be published to the topic “LED”, also the LED will need the information of button, hence it will subscribe to the topic “Button” and relay the information to the BBB. |
| Describe the implementation of the abstractions (how they connect to the actual device), and discuss any IoT-specific concern (including energy) that may have been addressed by your implementation. | Please see the attached file “part2\_architecture.jpg” for some details. In the cloud, we use mosquitto for the MQTT server. It handles all the subscribe/publish actions, hence connects the edges and the cloud. The dashboard is implemented by python with Django, it subscribes to the topics for updating the data. The edge runs the same program that is applicable to all the edges. The program is written in python (“general\_edge.py”). Given different input XML files, the program can know the devices that it handles and relays the information flow. The BBBs have the same copy of Xinu on it except the main files that are different for each BBB. For energy consumption, MQTT is able to efficiently send data. The main file in Xinu has only one process and sleeps regularly, hence saves energy. |
| Describe your on-board IoT devices Demo App. | **Devices**: describe any of the basic, composite and virtual devices used.  **App:** For the BBB with tmp36, the main function prints the temperature every 1 second.  For the BBB with the buttons, there are two buttons. One button is used to control the LED while the other button is used to control the buzzer. This is done by sending ‘1’ for the LED and ‘2’ for the buzzer from the main function of the button BBB.  For the LED and buzzer, the main function waits on message from Edge computer to on/off the respective device, and then sends the on/off status to Edge as acknowledgement. |
| Describe your web-based IoT devices Demo App. | The dashboard will   1. show the real time information of the devices read from the DDL whose field named “show” equals 1. And on the dashboard, each sensor has a button, which will publish different message to the topic “button”, the message will control the different sensor. 2. The dashboard has a warning system: 3. **Detection of fever:** It will let LED flash if the temperature is beyond 38.0, until the temperature is below 38.0 4. **Detection of boiling:** It will always change the Buzzer to buzzing or silence if the temperature is beyond 100.0, until the temperature is below 100.0   Use command **python manage.py runserver 0.0.0.0:8000** to run the server on the port 8000  **Front-end:** The front-end page is generated from html in template directory. The content of elements in html are passed by views.py. The style of the page is used a CSS file in static directory. There are JavaScript code for the button and for the refresh of the page.  **Back-end:** I use Django frame work as back-end frame work. The manage.py is the initialize file, it will set up the MQTT client configurations and it will start up the dashboard server, when the message come, it will call the showDashboard to update the data in dashboard. The urls.py is the router for different URLs, each URL will be assigned to a function in views.py. In views.py, the showDashboard function will update the dashboard and show it in the front end. There are some functions for button, they will publish message to topic button. |

**2. Challenges**

*Challenges your group faced. What was the most time consuming parts of the project? what piece(s) would you have really liked to have us provide to you so the total effort is more manageable (again, if any)?*

The biggest challenges are the development of device drivers. We tried 4 devices, 1 tmp36 as analog sensor, 1 button as digital sensor, 1 LED and 1 buzzer as actuators. It requires reading the datasheet of the CPU and the sensors. Because none of us has any experience to this work, it is very difficult to write device drivers. We can only resort to searching the internet, hoping to find codes that work through modification.

As we started reading on device drivers, we started asking many questions which were related to signal processing and computer architecture and required to be answered to develop a complete picture of device interaction. Given the time frame of project, it was a healthy challenge to gather all the required information to develop clear understanding, and then implement these at the same time.

Another challenge is the setup, because our use case requires at least 4 computers. Also, these computers will connect to the BBBs through a router, or share wifi over the Ethernet to the BBB. The setup is very inconvenient. We can only do it in the library occupying several tables. Also, the Xinu onboard the BBB may not connect to the internet sometimes.

**3. Overall Experience**

*Overall experience. Describe your overall experience good or bad.*

Overall, we believe that it was a good project. It helped us learn a lot and helped us see that IOT has a lot of potential.

This project made us think deeper into the questions that are fundamental to Internet of Things-

1. How interaction between two devices takes place in wired connection and in wireless connection.
2. How operating system provides abstraction to control these devices.
3. How security, portability, simplicity, flexibility can influence design decisions when considering interaction between a) two devices and b) interaction between OS application and device.

**4. Effort Distribution**

*Report only if effort was considered by any member of the group to not be even. In this case a table showing the names, ID’s, and percentage of effort should be provided.*