**COP 5615: Distributed Operating Systems Principles**

Fall 2016

Professor Sumi Helal

**Group Term Project**

**Internet of Things Support in Xinu**

Due: 1:00pm EST Tuesday November 25, 2016

**Background**

Designing and implementing a device driver for a *thing* in the Internet of Things (IoT) is different from what is traditionally done in a computer system. This is because in the latter case, the objective is to allow processes in an operating system to access and utilize devices that are either connected directly to the computer, or remotely through the network. In an IoT, things should utilize and use each others. Hence, both processes internal to a thing (e.g., Xinu processes running on a Beagle Bone Black platform) as well as other nearby things (e.g., other Beagle Bone Black or Dragon platforms) should be able to access and utilize each others as devices. Here, we use device and thing loosely interchangeably.

This term project aims to engage groups of 5 students in a challenge based learning experience in which each group will design and implement Xinu device drivers for things that can connect into a Beagle Bone Black, and for the Beagle Bone Black itself as a whole platform (as a thing)!

Coverage should include GPIO as minimally abstracted low-level I/O devices, covering several sensors/actuators and analog/digital interfaces. At least one combination of GPIO should be supported as a higher-level abstracted device. Finally, applications running on the BBB that utilize GPIO should be supported as high-level abstracted device.

Remember, users of these device drivers would be Xinu processes running on BBB as well as other devices and things external to the BBB. The traditional high-level I/O interface (open, close, getc, putc, read, write, cntrl) will need to be revisited for Xinu processes accessing things on the BBB. You need to think: in which way does the high-level interface need to change? What are the new concerns? What are the needed I/O operations? and are there any new operation abstractions that need to be introduced and supported?

And would the high-level I/O interface (even the one you would design for Xinu processes) be appropriate for other external things and devices? Or the same physical device should have multiple interfaces? … one for local access by Xinu processes, and others for access by external things?

**Part One – Onboard Device drivers and High-Level I/O Interface**

You will develop your project in stages building up to a full-featured device driver.

In this first phase, amiliarize yourself with BBB GPIO using available resources on the web. Also, check the I/O interface available for BBB. Scripting, C and C++ languages that compile to BBB provide a simple referential interface to GPIO devices. Study and play with such interfaces to get an idea on how GPIO is accessed from these languages over Linux. You will have no compilers for any of these languages over Xinu though, and you will have to use C to implement your device drivers as part of the Xinu code base.

In this phase, you will analyze the requirements for doing I/O on the BBB. You should consider the following aspects in defining your I/O interface, and in implementing the high-level and low-level device drivers in Xinu:

* Energy. The board has limited energy. How can you preform I/O in the most energy saving way?
* Latency. How fast can the I/O operation be completed?
* Power and appropriateness of the I/O interface. How powerful are the I/O interface system calls? Are they most appropriate for our purpose here in this project?

A design constraint in this phase is the requirement to use a declarative device driver manifest file. You will be required to define a Device Description Language (DDL) and use it to specialize the device drivers to a particular platform (e.g., Dragon, BBB, Libelium, Arduino, etc.) The DDL specification should be machine and human readable, which means you should consider XML or JSON. The DDL specification should be one of the deliverables of your project. Using the specification, you should create DDL files for BBB and all its physical and abstract devices.

The DDL file should be processed by XINU at compile time. Dependencies should be checked and if the DDL file has not been processed previously by an already compiled generator, it should. You will write the generator so that based on the DDL file, it generates as much of the code of the device drivers as possible. For instance, the generator may be able to completely generate the upper-half drivers for you.

How can you start here? What should the DDL specification contain? At this phase of the project, it should include information that aid in generating the drivers, as well as the high-level I/O interface to be used by the Xinu processes.

Think of the lowest level driver, in which you read or write to a pin label.

Then think of a higher-level driver in which you are reading a temperature sensor at appropriate pin.

Then think of an even higher-level driver in which multiple parts of the GPIO are read or written.

Finally, think of a highly abstracted device that includes code (application) that accesses other lower-level devices.

|  |
| --- |
| **TMP36 ANALOG TEMPERATURE SENSOR**  C:\Documents and Settings\Administrator\Desktop\Saint2009\tmp36.jpg    Figure 1. Snippet of a DDL |

A snippet of an example of some similar XML-based DDL file describing a temperature sensor is shown Figure 1.

Document the high-level I/O Interface and write manual pages for each of its operation.

Finally, develop extensive test programs to test your design choices and your implementation.

Of course with 5 people on board this part of the project should be quick and easy to implement.

**Part Two – Device Externalization to the Edge and the Cloud**

In this part of the project, you will consider inter-thing interactions and the device driver support that will enable them. Essentially, you will extend your work done in Part One above. Initializing a device driver now should setup whatever is necessary within Xinu to enable Xinu processes to utilize your new drivers (e.g., process switch table, etc.). Additionally, initializing a device entails optionally externalizing this device to other platforms, specifically, an edge computer (also known as a gateway), and the cloud.

To understand the rational behind this externalization process, we recall the Cloud-Sensor system architecture covered in class (Figure 2 below) in which a device or a thing may create a representation for itself on a an edge device co-located within the same “smart space”, or even with the cloud. This architecture enables application development and allows for powerful optimizations of the cloud resources and the device energy use. It is also a communication architecture in which heterogeneous devices and things may communicate with each other through an edge or the cloud.

In this part two, you will establish a high level abstraction of the device both on an edge device and in the cloud. The abstraction would interface external interactions to the device as would internal interactions by the Xinu processes.

|  |
| --- |
| Figure 2. The three-tier IoT architecture of the Cloud-Edge-Beneath, studied in the class, and which will be considered here in this project as a requirement. |

The DDL will be the main mechanism in which an edge that a thing belongs to can be specified. Adequate information and credentials should be specified as an edge object in the DDL to enable access and to establish an edge-hosted version of the device. Security should be addressed carefully given the clear text nature of your DDL file.

Similarly, a specification of a Cloud DDL object should be designed to allow devices to have cloud representation. Concurrent representations in the edge and cloud should be allowed (what does that mean? What are the consequences?)

You have to choose an appropriate abstraction for the device representations. You are encouraged to exploit only well established standards to implement this abstraction (what are your options here?)

Make sure the abstraction you chose captures as much as possible of the important new aspects you introduced in your new high-level I/O interface (e.g., the abstractions are powerful, energy-aware, and delay-conscience).

Life cycle of the external device services should be managed. Shutting down Xinu should recall (remove) the device edge and cloud representations.

You are also to develop a web-based dashboard to view and monitor the status of all devices of a running Xinu instance.

Once this part is implemented and extensively tested, you will develop manual pages for the external abstraction and how to browse and reflect them, and of course how to use them.

**Part Three – Application Demonstrations**

To make sure your project is successful, you will develop, describe and document two types IoT applications of different types:

In the first type, your application will consist of Xinu processes accessing the various devices demonstrating the power of your new high-level I/O interface.

In the second type, you will develop applications external to the BBB perhaps using some web-based application engine or something compliant with your abstraction for edge/cloud device services. The application should demonstrate the power of your device driver design in which the device is accessible from the edge (app deployed at the edge), or the cloud.

**Deliverables and Submission**

You will deliver the following:

* A zip file containing: (1) a Xinu source package along with a readme file explaining all updated and new files you introduce to the Xinu code, (2) DDL specification document, and DDL drivers for the specific BBB devices you implemented. Also, you will include here the DDL device driver generator, (3) code for your web-based device browsing and monitoring, (4) documented code for all the applications that you have implemented, and finally, (5) a project report describing your overall design, and the design choices of each part in as much details as possible. You should describe the DDL part including the DDL driver generator and what exactly it generates, the high-level I/O interface that you chose and designed, the Xinu run-time environment (tables you introduced or adopted and changed), etc. The zip file should be submitted on Canvas by the aforementioned deadline.
* A short presentation (no more than 10 min) that you will present during the face to face evaluation.
* Three copies of your project report that you will hand to us during face to face evaluation.