Preparing to Teach an Undergraduate Embedded Linux Course

Krista Hill, University of Hartford, Member ASEE

Abstract—It seems that computers are everywhere. The term embedded refers to systems that may be enhanced with a computer, but that we don’t think of as containing one. Today’s cars have reduced emissions and enhanced performance. Likewise, High Definition Television (HDTV) provides higher resolution than previously possible.

The wide adoption of Linux for embedded systems is a significant shift in paradigm. Unlike classic embedded systems, Linux is a multi-programmed operating system that runs on enhanced hardware and easily connects to the Internet. Such systems enable development of the Internet of things. Our interest here is in the devices layer.

To illustrate how classic embedded systems compare with embedded Linux, we compare two microcontrollers which our students are familiar with, to a Raspberry Pi system running Linux. As shown below, these are very different types of systems.

A Linux distribution and system board was chosen that can be headless, which like many embedded systems do not have a video display, keyboard, or mouse, but rather are accessed from a host PC. Such systems may provide a general purpose interface port and use a serial bus to communicate with peripheral devices such as a real-time clock, an analog to digital converter, and a digital to analog converter.

To enable a graphical user interface we use the secure shell program to exchange X-Windows messages with a host PC. Internet connectivity and power are provided by a USB cable. While our students are introduced to shell scripting, the primary programming language is Python.

Index Terms—Embedded Linux, embedded systems, Python, courseware, engaging students.

I. INTRODUCTION

In preparing to teach the embedded Linux course, there were many considerations and choices to make in terms of resources and content. To start, this course is intended for sophomore and junior students.

In planning content, it is first necessary to provide the students a sense of perspective of what they will be dealing with. We start with an outline how Raspberry Pi (RasPi) systems are very different than those with, to a Raspberry Pi system running Linux. As shown below, these are very different types of systems.

To paraphrase the Linux foundation[1], Linux is an operating system which like many embedded systems do not have a video display, keyboard, or mouse, but rather are accessed from a host PC. Such systems provide functionality such as network communications and expanded memory. The Arduino can have add-on boards called capes that provide high performance and can be used to execute a single application, a Linux system can run many applications.

To paraphrase the Linux foundation[1], Linux is an operating system; it is software that manages all of the hardware resources in a computer, for use by application software. An operating system essentially manages the communication between your application software and the hardware, providing necessary services. Without the operating system neither the applications or hardware would function.

The target system chosen for our use is the Raspberry Pi Zero-W (RasPi-0W) board along with a real-time clock module. The choice of TinyCore Linux is outlined further below along with the development environment which makes use of X-Windows[2] graphics.

The primary programming language used is Python[3]. A four layer model of the Internet of Things (IoT) is presented, with emphasis on the devices layer. The RPi.GPIO library and I2C serial communications are used in class activities, to use a digital to analog converter as well as an analog to digital converter. The gateway and cloud computing IoT layers can be the topics of an advanced course.

A. Linux Capable Enhanced Hardware

The use of an advanced operating system such as Linux in an embedded system is a significant change in paradigm from that of microcontrollers. To understand this difference, we compare the RasPi-0W to two microcontrollers used in undergraduate courses.

Table 1 summarizes some properties of these systems. The Arduino Uno development board[4] is based on the ATmega328P[5] 8-bit microcontroller and the NXP 9S12C128[6] is a 16-bit processor.

First consider the properties in Table 1 that provide a clear comparison. The process of developing applications for a microcontroller generally involves a separate, larger computer, called the host. The Arduino and 9S12 are intended to be used for a single application. By comparisons, a RasPi system can serve as a host to itself and can simultaneously execute several applications.

A RasPi system provides a micro-SD card and several network interfaces. In particular, with the RasPi-0W we use the USB interface to tunnel Internet messages through an attached PC, using Internet Connection Sharing (ICS). Otherwise, with HDMI video and the dual-use USB interface, a RasPi-0W can be used as a small desktop computer. By comparison, the Arduino and 9S12 are slave USB devices. The Arduino can have add-on boards called capes that provide functionality such as network communications and expanded storage.

<table>
<thead>
<tr>
<th></th>
<th>Self hosting</th>
<th>CPU speed</th>
<th>Total memory</th>
<th>USB comm.</th>
<th>Video</th>
<th>Network Comm.</th>
<th>Expanded storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>RasPi 0W</td>
<td>Yes</td>
<td>1GHz</td>
<td>512 MByte</td>
<td>OTG dual role (M/S)</td>
<td>HDMI</td>
<td>WiFi, Bluetooth, Ethernet dongle, USB tunneling</td>
<td>Micro SD card</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>No</td>
<td>16MHz</td>
<td>35k Byte</td>
<td>Slave</td>
<td>None</td>
<td>Cape</td>
<td>Cape</td>
</tr>
<tr>
<td>NXP 9S12</td>
<td>No</td>
<td>50MHz max</td>
<td>132k Byte</td>
<td>Slave debug adapter</td>
<td>None</td>
<td>None</td>
<td>Expanded bus mode</td>
</tr>
</tbody>
</table>

Table 1: Comparing Arduino Uno to Raspberry Pi Zero-W
The speed of the RasPi-0W is almost two orders of magnitude faster than the Arduino and one order faster than a 9S12C. Finally, the RasPi-0W has approximately four orders of magnitude more memory than either the Uno or the 9S12C128 microcontroller.

B. Choice of Linux Capable Hardware

The choice to use a RasPi board came partly from their apparent wide use as well as the active port of TinyCore Linux for RasPi systems. Of the boards, the RasPi-0W, shown in Figure 1, was chosen for several reasons: First off, it is less expensive than larger boards. Also, unlike larger boards which are USB masters, the RasPi-0W dual-use connector greatly simplifies the development environment. There is no need for a network router or USB bridge cable. And finally, the RasPi-0W provides adequate resources for our use.

In Figure 1, the processor System on a Chip (SoC) is the dark rectangle just to the left of the center of the board. The micro-SD and HDMI connectors are to the left, holes for the GPIO connector are across the top, a connector for a camera is to the right, and micro-USB connectors are to the lower right. Of these, the USB connector to the right is only for powering the board.

The RasPi-0W is similar to actual embedded Linux systems. To understand this comment, consider that the mission of the RaspPi Foundation[7] is to provide, “low-cost, high-performance computers that people use to learn, solve problems and have fun.” In other words, most RasPi boards are small desktop-like computers.

A first hurdle to overcome as an embedded system is that RasPi boards do not include a real-time time clock, as a cost saving measure. Unlike a desktop computer, embedded systems may not have immediate network access to a time of day server. As such our systems each include a PCF8523 RTC real-time clock module[8].

Regarding resources, an advantage of the RasPi-0W is that its dual-use USB connector allows the board to be used as a slave device to a PC, making use of the keyboard, mouse, network connection, and power provided by the PC. There is no need for an additional display, keyboard, power source, or pointing device of any kind.

C. Choice Of A Linux Distribution

In considering Linux distributions, the focus here is on embedded systems. I was first introduced to TinyCore Linux[9] several years ago when I was working with graduate students to develop Intel x86 type embedded systems. TinyCore boots into a RAM file system, which means that the system always starts with a pristine version of the operating system. In addition, in using a RAM file system, the performance of TinyCore is fast compared to similar systems.

The Core project supports both TinyCore which traditionally includes a graphical interface, as well as MicroCore, which provides only a command line type shell interface. TinyCore and MicroCore are not turnkey desktop distributions. The base distributions for TinyCore and MicroCore are 16 Mbytes and 11 Mbytes, respectively.

By comparison, the official Linux distribution supported by the RaspPi Foundation is called Raspbian[10], which is based on Debian and is optimized for RaspPi hardware. In actually being a desktop distribution, Raspbian is large in size and executes from an SD type memory card, which surely acts as a performance bottleneck. Also, given that the system executes from attached memory, there is more risk of corrupting the file system. The SD memory card must not be removed while such a system is running.

TinyCore Linux is a base distribution that a developer customizes with packages from the TinyCore repository as well as their own custom software. TinyCore Linux serves as a useful educational stepping stone toward custom Linux development tools, such as Yocto[11] which builds a custom Linux distribution from source code.

D. Development Environment

In developing an embedded Linux system, the command line shell interface is most basic. Students are introduced to the Ash shell, provided by BusyBox[12], which is commonly used with embedded Linux systems. The Ash shell is similar to the Bash[13] shell. Students use either a serial COM port or secure shell[14] (ssh) to communicate with Ash.

A feature of secure shell is that it can forward X-Windows[2] message traffic, allowing for a graphical environment. On an attached PC we use the X-Windows server from Cygwin[15] to provide a graphical terminal that can support a development environment.

Figure 2 shows the TinyCore application browser GUI, which provides easy access to the TinyCore repository. In this case the package for Python is selected. The choice shown above Python is for the well known RPi.GPIO library, which students use with the GPIO pins on the board. Also, the process of building an example TinyCore package is presented to the students.

Apart the Ash shell, our students primarily use the Python language[3] along with its standard integrated development environment (IDE) called Idle. The program Idle is run on the RasPi with the
corresponding display on an attached PC, as shown in Figure 3; the Idle shell along with the source code editor which contains a file named count3x.py are shown. The debugger control window is also shown.

Figure 2: Application browser accessing TinyCore repository

Figure 3: Idle shell along with editor and debug control window

E. Internet Of Things

The term Internet of Things (IoT) [16] describes how a hierarchy of computers in a network can work together to provide enhanced services. Suppose that as you drive home, as outlined in Figure 4, a GPS receiver and embedded system in your car communicate through the cloud to your home. During your travel, the heating in your house is adjusted to be more comfortable. Also your refrigerator sends a message suggesting that you should stop by the supermarket, as milk, cereal, and strawberries are on sale.

In considering the IoT hierarchy it helps to have a layered network model similar to the Open Systems Interconnection model[17]. For the purpose of this paper consider a four-layer model of computers in a network that describes aspects of the IoT.
1. Devices, which are essentially the things used for sensing and for controlling products.

2. Gateways, which provide communications between devices and the Internet. A gateway device can provide more robust security than that provided by a simple device.

3. Cloud computing, provides services for devices as well as users who use the Internet to access IoT devices.

4. Big data, provides large scale processing and data mining of information.

In teaching this course, my primary interest is in the devices layer. The vast majority of everyday electronic and computer engineering hardware design involves the devices layer, which involves sensors as well as actuators. The gateway and cloud computing layers provide connectivity to and through the Internet, which can be the topics of an advanced embedded Linux course.

In this course, our students are introduced to the RPi.GPIO library[3], which provides a convenient application programmer interface for GPIO pin interfacing. Among the topics, students investigate how to describe state machines, which is a useful paradigm for interactive applications. Students are also introduced to I2C communications which they use with Python to make use of a digital to analog converter, as well as an analog to digital converter.

F. Assessment

As of this writing, the embedded Linux course is off to a good start. In class the students are engaged and express joy in completing the hands-on class activities and projects. While handling the glitches that have happened, the students were interested in the cause and solution to each specific situation that arose.

II. CONCLUSION

In teaching this course for the first time, all the choices that were made have been helpful. The Ash shell provided by BusyBox is a basic tool that students are introduced to. The use of a Raspberry Pi board along with X-Windows graphics and Idle for Python provides an effective introductory development environment. TinyCore Linux is a base systems that our students learn to modify for their use. In considering the Internet of Things, our concern here is the devices layer. To interact with the environment the RPi.GPIO library as well as I2C communications are useful. Among the hands-on activities and projects, the students write Python code to interact with an analog to digital converter as well as a digital and analog converter.