Evaluating possible uses of a Raspberry Pi in an academic library environment

J.L. Cooper, J.P. Knight

Abstract

Loughborough University's Library Systems Team investigated two potential uses for Raspberry Pis. The first use to be identified for investigation was using the Raspberry Pi as a replacement for the existing OPAC hardware. While it met a majority of the requirements there were issues with the responsiveness to user input at certain times. The second use for Raspberry Pis investigated was to provide a number of digital signs to display details about resource bookings and the availability of PCs in IT labs around campus. The Raspberry Pi demonstrated that it was ideally suited to this task.

Introduction

The Raspberry Pi is a credit card sized computer developed and produced by the Raspberry Pi Foundation. Their aim is to increase the number of children learning to program computers. However, the low cost of the Raspberry Pi has led to it being used by many people for a wide variety of projects [1, 2, 3]. Given this low cost it was a natural reaction for Loughborough University Library to investigate their potential for use in an academic library environment.

Hardware

There are two different versions of the Raspberry Pi, the Model A and Model B. Both share the same base hardware specification:

- Broadcom BCM2835 system-on-a-chip (providing CPU, Graphics, Memory Controller and USB 2.0).
- SD Card for on-board storage (SDHC is supported)
- Composite video output
- HDMI video output
- 5v power source required via a MicroUSB connector.

In addition to the base specification the Model A has 256MB of RAM and a single USB port. Early revisions of the Model B had 256MB of RAM but this was increased in later revisions to 512MB of RAM. All revisions of the Model B have a 3 port USB hub; two of the ports are accessible to the user, while the third one is used internally to provide a 10/100Mb Ethernet port.

In addition to the Raspberry Pi itself the following hardware is required to use it as a traditional computer:

- SD Card
- Keyboard
- Mouse
- Monitor (and possibly a HDMI to VGA/DVI adapter, depending on the monitors inputs)
- MicroUSB power supply

The SD Card used also needs to have a suitable Operating System (OS) installed on it. While there are a number of OS's available for the Raspberry Pi their recommended one is Linux, specifically a version of Debian called Raspbian.

Replacing OPAC PCs

Loughborough University Library has a small number of dedicated Library Catalogue PCs (OPAC) which give Library users access to the Library Catalogue (Ex Libris's Primo) and the institutional Online Reading List System (LORLS [4]). Traditionally these OPACs are average desktop PCs running a customised version of Linux in a kiosk mode. Given the potential cost saving it was decided that this would be a good task to try using a Raspberry Pi for. In addition it would also help identify potential limiting factors of the Raspberry Pi architecture.

OPAC Requirements

Three key requirements were identified that the Raspberry Pi would have to meet to be suitable replacement hardware for the existing OPACs:

- Must work with latest version of PRIMO
- Must work with latest version of LORLS
- Must feel responsive to users

Configurations tested

A number of different configurations were tested as new versions of the hardware and software became available. By the end of the investigation the following configurations had been tested:

- 256MB Model B using Raspbian with software floating point operations
- 256MB Model B using Raspbian with hardware floating point operations
- 512MB Model B using Raspbian with hardware floating point operations

For all configurations tested, a screen resolution of 1024x768 was used and all non-essential processes were terminated. The web browser chosen to be used was the light weight Midori.

Results and analysis

PRIMO worked on all three configurations and felt responsive. LORLS worked on all three but didn't feel acceptably responsive on large lists (600+ items). The responsiveness improved as the configuration moved to using hardware for floating point operations, but it still felt sluggish while loading and displaying large reading lists.

These differing results very clearly demonstrate a difference in the technology used in PRIMO and LORLS. PRIMO produces the HTML on its server and passes that to the client browser. This increases the load on the server but reduces the load on the client. In contrast, the LORLS server passes the data to the client browser with a piece of JavaScript which then generates the HTML to be displayed. Finally, the browser has to render that HTML to produce the page. This approach reduces the load on the server but increases the load on the client.

There was also another point where, after a period of usage, the performance of the client browser would decline for configurations 1 and 2, but not configuration 3. After investigating the issue it was discovered that this drop in performance was due to the system memory being full and the OS starting to swap out parts of memory to its mass storage (i.e. the SD card).

OPAC conclusion

The Raspberry Pi met 2 out of 3 of the requirements. It was capable of working with both PRIMO and LORLS, but it didn't perform well enough on large reading lists for it to be an acceptable replacement of the current OPACs.

At the time of testing there was no hardware accelerated video driver available for X-Windows on the Raspberry Pi, and therefore the processor has to do a lot of the rendering work that would traditionally be done by the GPU. Recently it was announced that Raspbian would in time support an X-Windows alternative, Wayland [5], and that this would be hardware accelerated. Once this becomes available it would be worth investigating the performance again as, in theory, the hardware acceleration should improve the Raspberry Pi's performance.

Digital Signage

During the summer of 2013 Loughborough University's library building underwent a major refurbishment. As part of this refurbishment it was decided that a number of display screens would be included to provide information to Library users, specifically bookings for rooms and current availability of PCs in IT Labs around campus. This task appeared to be an ideal one for a Raspberry Pi and given that there would be no direct user input there wouldn't be the responsiveness issue previously encountered.

Requirements

The initial set of requirements were very simple: it should be able to display a HTML5 web page, and it should be able to regularly run a JavaScript routine to update the content of the page.

Developing the HTML pages to display

Despite there being a large number of resources for which information would need to be displayed, there were only two separate systems the information would have to be pulled from. Both had suitable Application Programming Interfaces (APIs) available, having already been developed for the Library's mobile webApp [6]. This reduced the actual number of HTML pages that would be needed down to just two dynamic ones.

The technology chosen for both of the dynamic pages was HTML5 and the JQuery [7] framework. These were chosen due to local knowledge of them and fast development time.

The first page developed was PC Availability. This displays two tables, one for open access IT labs and the other for teaching labs, which it populates with the latest information from a PC Availability API. The contents of the tables is updated every 2 minutes, which is often enough for the information to be current without putting undue stress on the server by making a large number of calls to the API within short periods.

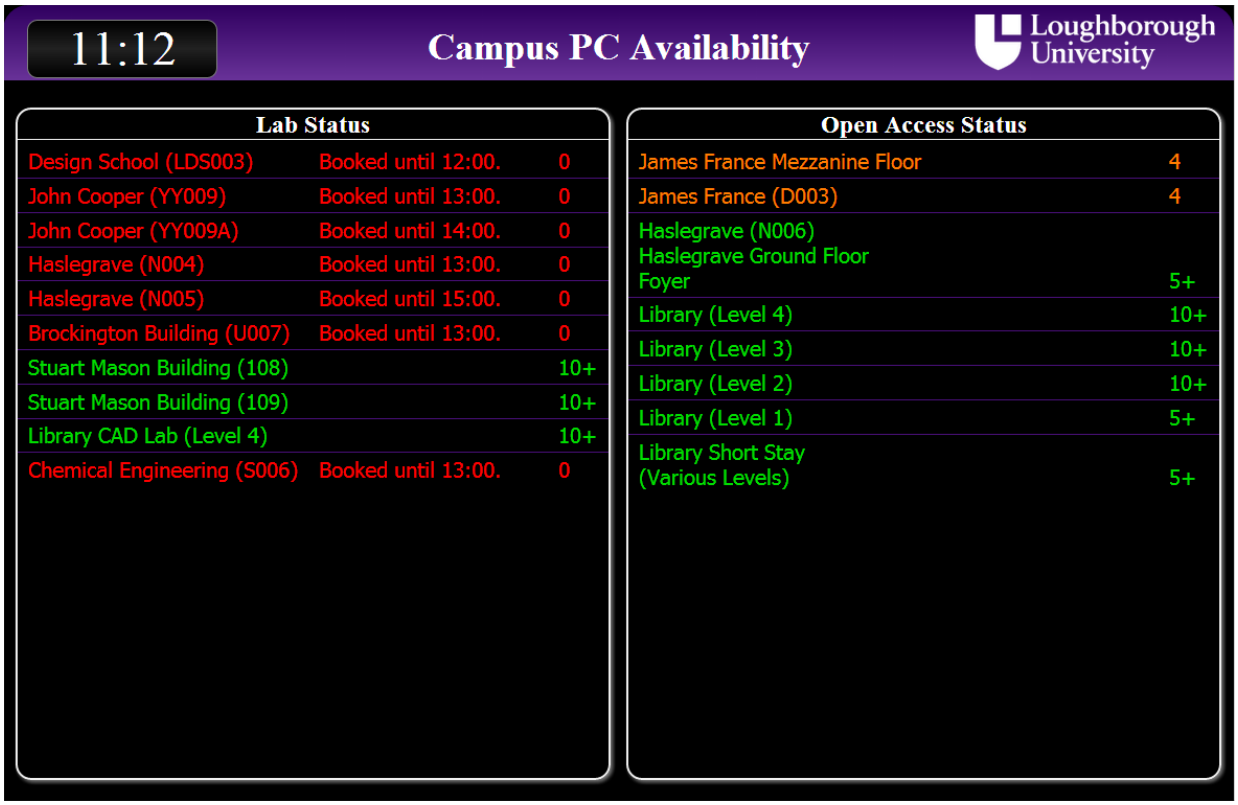

Figure 1: PC Availability Screen Shot

The second page developed lists room bookings from the Library's resource booking system (WUBS). As the specific resources to be displayed would vary on a screen by screen basis it was decided to go for a single dynamic page that would pull its configuration parameters from its query string. The parameters available for the finished page were:

- resources; a comma separated list of resource ids
- columns; the number of columns to display the resources in
- level; the floor of the library the display will be shown on.

The code that does the main work of the page has two phases, a start-up phase and a main loop. The start-up phase goes through the following steps:

1. It parses the page's query string and extracts the parameters.
2. It builds a class name from the level parameter and adds this to its body element. This makes it very easy to base the page's style on the level, which will keep the display screens consistent with the Library's existing floor colour coding scheme.
3. It prepares a series of place holder DIV elements, one for each resource to be displayed.
4. It calls a WUBS API to obtain the names of all the resources, which it stores in an array.
5. It passes control to the main loop.

Figure 2: Room availability screen shot - level 1

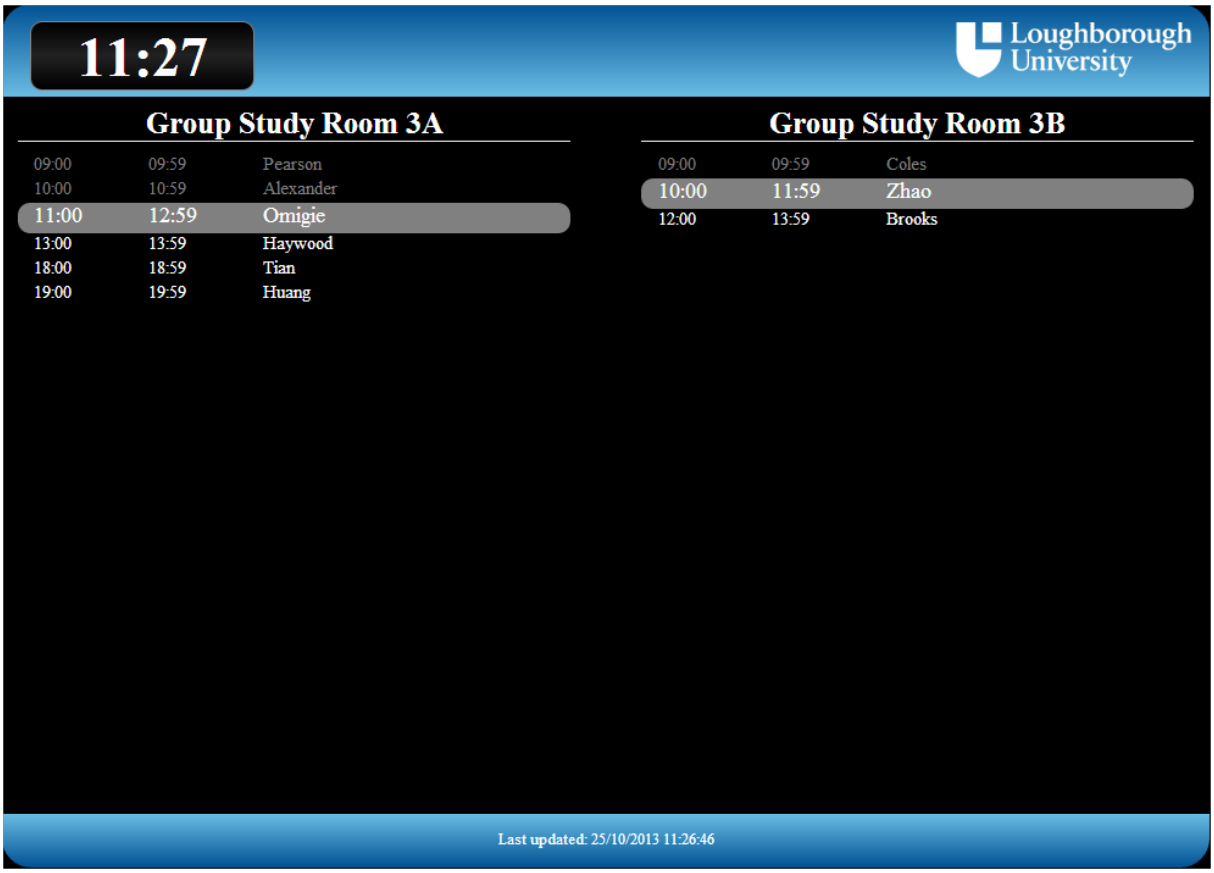

Figure 3: Room availability screen shot - level 3

The main loop consists of the following steps:

1. It calls a WUBS API to retrieve a list of the current day's bookings for each resource.
2. The booking details for each resource are then processed to merge continuous bookings by the same users.
3. It decides on the display method to be used. If there are more resources to be displayed than twice the number of columns then a reduced display is chosen, otherwise a full display is used.
4. If a full display is chosen then a table is produced for each resource detailing, for each booking, its start and end times, the surname of the user who has the resource booked and if the resource is a seminar room the title of the scheduled event.
5. If a reduced display is chosen then only 3 bookings will be shown for each resource. The 3 bookings shown consist of the previous booking, the current booking and the next booking. If there isn't a previous or current booking then more future bookings are shown.
6. The "Last Updated" time in the page's footer is updated with the current time.
7. It waits for 2 minutes then starts the loop again.

After reviewing both the IT Lab Availability and Resource Availability pages it was decided that it would be useful for Library users to have the current time displayed. This was achieved by simply adding a DIV element to hold the time and styling it via CSS [8] to fit in with the rest of the page. Then a small JavaScript function was added that is called every few seconds to update it the DIV with the current time.

Configuring the Raspberry Pi's for digital signage.

There were two obvious methods available for configuring the Raspberry Pi's for use as digital signage. One method used by Central Connecticut State University's Burritt Library [9] is to start with a standard Raspbian OS and then customise it in a number of ways, including automatically logging in, automatically open a URL in a browser in kiosk mode and disabling screen savers and screen blanking. The other method was to use Screenly [10] on the Raspberry Pi. This is an application designed to turn a Raspberry Pi into a configurable display sign. To simplify installation there is a version of Raspbian available with Screenly preinstalled and the common additional configurations made for it to run effectively as a digital sign. The content to be displayed on the screen is configurable through a web interface. Both methods would achieve the desired results but using Screenly would be both quicker to deploy and easier to maintain in the future. One area in which the Screenly image does require extra work is in securing it, as the web interface doesn't require any authentication to use.

Testing the Raspberry Pi as a display screen

To test how well the Raspberry Pi would handle running a display screen for a long period of time, one was fitted with an SD card loaded with Screenly. The web interface was used to configure it to display the upcoming bookings for a number of resources, and it was left displaying to a monitor for the next week. During this period the Raspberry Pi proved to be a stable platform for service delivery. The hardware, while being warm to the touch, never got too hot.

Securing Pi's and Screenly

There were a number of areas that the Raspberry Pis needed to have secured before they could be used to drive the display screens in public. The first thing that stood out as needing to be tackled was the issue of default passwords. The Screenly image had two users as standard (root and pi), both of which had a default password. These needed changing for both users to limit shell access to only legitimate administrators. Changing these passwords was a simple case of logging into the Pi using SSH and using the passwd command for both the pi and root users.

After securing the user login details, securing the web interface was investigated. After a quick look at the web interfaces' underlying Python code it became obvious that retro fitting secure authentication into the code would be a lot of work to both implement and maintain. The alternative method to secure the web interface that was used instead was to stop the web interface listening on all of the Raspberry Pi's network interfaces, instead setting it to only accept connections on the localhost interface. While this change would stop anyone from accessing the web interface from the network it would still be possible for administrators to access the admin interface by using a ssh tunnel [11] to forward the web interface's port from the Raspberry Pi's localhost interface to port 8080 on their localhost interface. Then they can simply point their web browser at http://localhost:8080/ and access the web interface.

The physical security of the Raspberry Pi had to be tackled next. Because it would have to be located close to the screens it would be driving then each Raspberry Pi would have to be physically secured. There is a large choice of cases available for the Raspberry Pi. The Cyntech [12] case was chosen because it supported an SD card cover [13] which would protect the SD card from being easily damaged, and also stop it being easily removed or replaced.

After selecting the case, the other physical security issue tackled was the placing. Each Raspberry Pi was located above the false ceiling, where data and power were available. The video signal was carried over a HDMI cable, run down through trunking inside the wall to the screens. This keeps the Raspberry Pi out of sight and reach of most users.

Display screens in use

In total, 6 Raspberry Pis were configured and deployed as display screens. One shows the current availability of PCs in IT labs around the campus, and the other five show the bookings for a variety of resources covering seminar rooms, meeting rooms, group study rooms and bookable booths.

Figure 4: Display screen in use for Group Study Rooms

Conclusion

The following conclusions were drawn from investigating these potential uses for Raspberry Pis in an academic Library environment:

- Raspberry Pis present a significant cost saving over traditional devices.
- They can display traditional HTML pages well enough to appear responsive to user input.
- When dealing with larger quantities of data, in an AJAX style web page, they can become less responsive.
- If memory usage goes above the available physical RAM on the device, there is a markedly noticeable drop in the performance of the device as it moves to use swap space.
- Raspberry Pis make excellent devices to drive display screens.

Another aspect that became obvious during the investigation into replacing the OPAC machines with Raspberry Pis is that there is a very rapid level of development of the underlying device drivers still taking place for the Raspberry Pi. As more of the Raspberry Pi's device drivers move over to using its available hardware the overall performance of the device improves. This investigation into the Raspberry Pi has shown that it is a very useful device and is capable of replacing a number of more expensive devices in the academic library environment.

References

[1] Dorey Brian, 2012. Raspberry Pi Solar Data Logger. BrianDorey.com.

[2] Akerman Dave, 2013. Ted Bull Stratos: Babbage's leap of faith. Raspberry Pi.

[3] Auld Chad, 2013. If you build it they will feed! Manifold.

[4] Knight Jon, Cooper Jason, Brewerton Gary. Redeveloping the Loughborough Online Reading List System. July 2012, Ariadne, Issue 69.

[5] Upton, Eben. Wayland Preview, May 2013. RaspberyPi.org.

[6] Cooper Jason, Brewerton Gary. Developing a Prototype Library WebApp for Mobile Devices. July 2013, Ariadne. Issue 71.

[7] jQuery Foundation. jQuery: The Write Less, Do More, JavaScript Library. jQuery.org.

[8] Bos Bert. Cascading StyleSheets. W3C.

[9] Iglesias Edward, Schlegel Arianna. Using a Raspberry Pi as a Versatile and Inexpensive Display Device. July 2013, Code4Lib. Issue 21.

[10] WireLoad Inc. Screenly Open Source Edition.

[11] Wikipedia. Tunneling protocol.

[12] ModMyPi. Cyntech Blackberry Raspberry Pi Case. ModMyPi.com.

[13] ModMyPi. Cyntech Case SD Card Cover Black. ModMyPi.com.