Web Kiosk Embedded Linux System

Vincent Sanders
Daniel Silverstone

Revision History

Revision 1.00	6th March 2009	VRS, DJAS
	Initial Release.	

Table of Contents

This article describes how to construct a simple Linux-based embedded web kiosk.

1. Introduction

This is the third article in a series demonstrating the fundamental aspects of constructing embedded systems.

This article covers the construction of a web browser with a command shell on the serial console.

This document, and indeed the whole series of articles, assumes a basic understanding of a Linux-based operating system. While discussing concepts and general approaches these concepts are demonstrated with extensive practical examples. All the practical examples are based upon a Debian- or Ubuntu-based distribution.

2. Pushing the limits

So far in this series we have used binaries from the host system or from pre-packaged software. We will now expand this to the inclusion of software built from source.

As will be demonstrated, this involves little more effort than the previous examples in terms of configuration script complexity but will demonstrate the increasing burden of attempting to support additional methods of user interaction.

The primary cause of this increase in complexity comes from the additional kernel driver modules required to make the input and output devices operational. We saw in the previous web server example that drivers were required for each network card we wanted to support. For supporting two network cards (PCI NE2000 for QEMU and e1000e for a ThinkPad laptop) we added three modules.

In this project we need the modules necessary to support the input event system and the drivers to run the hardware. To support a minimal useful set, for mice and keyboards attached via USB HID and PS/2, the count runs to some eleven modules.

This is futher exacerbated by the need to have output using the framebuffer. To support the intelfb driver (for a Thinkpad laptop) and the cirrusfb driver (for QEMU) requires an additional twelve modules.

As these numbers demonstrate, a relatively small increase in supported interfaces rapidly increases the number of drivers required. We are also rapidly approaching the limits of what static module insertion is capable of and would need to start dynamically loading modules according to what hardware is present.

3. Selecting a browser

To have a web kiosk application we obviously need a browser. We could use attempt to run Firefox or a Webkit browser using the X windows system but the dependencies for X and such a browser would make our resulting system huge which is undesirable.

One posibility for improving the situation might be to use a graphics library other than X such as DirectFB with toolkit support. This appears appealing at first glance but still requires a large number of libraries and a lot of software which is not freqently tested and hence will probbaly contain numerous issues we have to solve.

Another possibility is a browser which outputs directly to the Linux framebuffer. This posibility appeals as there is no need for a large toolkit and graphics library and browsers of this type tend to be smaller than their X based siblings.

After some searching, three candidate browsers which run directly on the Linux framebuffer were found:

Zen [http://www.nocrew.org/software/zen/]	A small browser which implements only basic layout. It removed itself from serious consideration as its author clearly states all development has ceased.
Links [http://links.twibright.com/]	The links browser is relatively small and supports HTML 4, however its lack of CSS support and various build issues reduce its appeal.
NetSurf [http://www.netsurf-browser.org/]	The NetSurf browser has several build targets including GTK so it might have been a contender for the DirectFB and GTK type approach. In addition however it supports several framebuffer type display options including the Linux framebuffer. It is in current development and has support for HTML 5 and CSS although it lacks JavaScript support.

Based on the available choices, NetSurf using the Linux framebuffer frontend was selected.

4. Building the browser

Building the NetSurf browser is suprisingly simple. Since the Linux framebuffer port has not been included in an official release it must be built from the project's Subversion repository. This is not generally recommended for embedded systems but there is little other option, at the time of writing.

First the NetSurf development trunk should be checked out using subversion:

```
$ svn co svn://svn.netsurf-browser.org/trunk/netsurf
```

Within the checkout (in the `netsurf/` directory) there is a document called `Docs/BUILDING-Framebuffer` which describes the steps necessary to build the browser.

The summary of operations is to install and build the library dependencies, place a configuration makefile fragment in the NetSurf directory and run **make TARGET=framebuffer**.

```
$ sudo apt-get install build-essential libcurl3-dev libxml2-dev
$ sudo apt-get install libmng-dev librsvg2-dev lemon
$ sudo apt-get install re2c libfreetype6-dev ttf-bitstream-vera
$ svn co svn://svn.netsurf-browser.org/trunk/libnsbmp
$ sudo make -C libnsbmp install
$ svn co  svn://svn.netsurf-browser.org/trunk/libnsgif
$ sudo make -C libnsgif install
$ svn://svn.netsurf-browser.org/trunk/libparserutils
$ sudo make -C libparserutils install
$ svn://svn.netsurf-browser.org/trunk/hubbub
$ sudo make -C hubbub install
$ cd netsurf
```

The `Makefile.config.override` should contain the lines:

```
NETSURF_FB_FONTLIB=freetype
NETSURF_FB_FRONTEND=linux
```

This configures the use of the Linux framebuffer frontend and the FreeType 2 library for font handling. Once the compile has completed it should produce an `nsfb-linux` binary.

5. Putting it all together

We will be using the same `mkbusyfs.sh` script from our previous examples, indeed you can continue using the previous installation and simply add configurations as the series progresses.

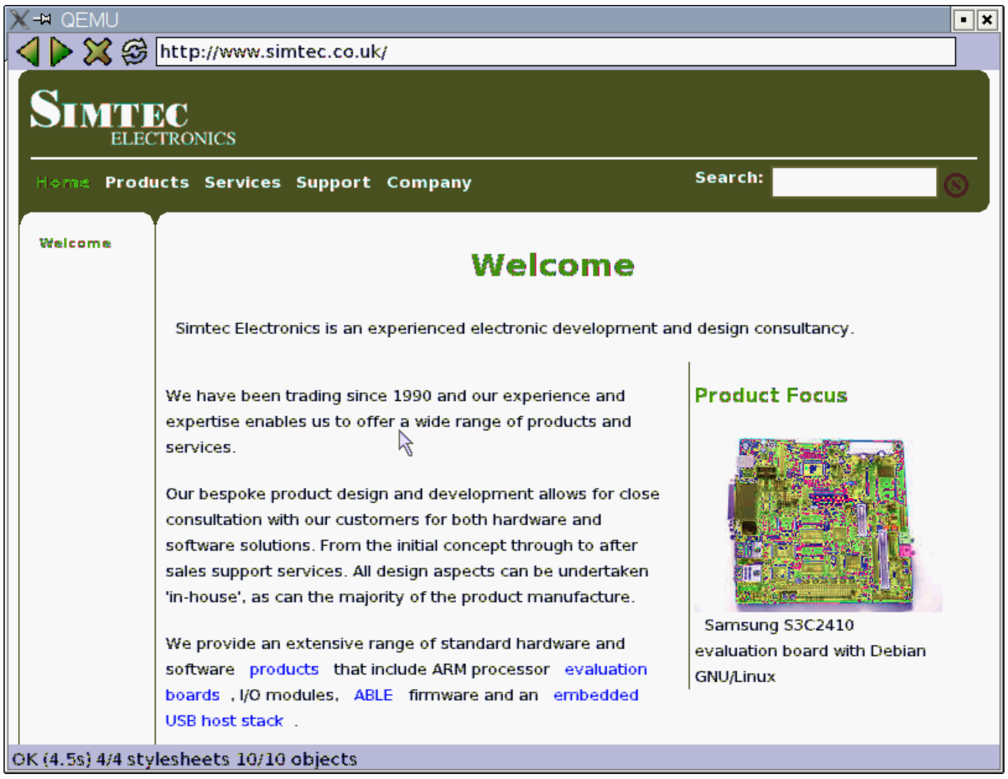
The configuration script [http://www.simtec.co.uk/products/SWLINUX/files/mkbusyfs/webkiosk.sh] for the web kiosk system is straightforward. The only especially interesting item is the number of kernel modules which, as already discussed, has grown considerably.

The mkbusyfs.sh tool should be used to generate the `webkiosk.gz` which can then be tested with QEMU. The QEMU commandline is slightly different to previous examples as it must redirect the console to a serial port so the video hardware can be used for the graphical framebuffer.

```
qemu -kernel ./vmlinuz-2.6.26-1-686 -initrd webkiosk.gz \
  -append "root=/dev/ram console=ttyS0" -net nic -net user /dev/zero
```

Several issues were experienced with QEMU and only after resorting to installing the latest version from the QEMU Subversion repository was graphical output obtained. As can be seen the output is still not correct, it was not determined whether this was another issue with the emulated video card or the browser.

Figure 1. QEMU video display



Welcome

Simtec Electronics is an experienced electronic development and design consultancy.

We have been trading since 1990 and our experience and expertise enables us to offer a wide range of products and services.

Our bespoke product design and development allows for close consultation with our customers for both hardware and software solutions. From the initial concept through to after sales support services. All design aspects can be undertaken 'in-house', as can the majority of the product manufacture.

We provide an extensive range of standard hardware and software products that include ARM processor evaluation boards , I/O modules, ABLE firmware and an embedded USB host stack .

Product Focus

Samsung S3C2410 evaluation board with Debian GNU/Linux

OK (4.5s) 4/4 stylesheets 10/10 objects

The pre built Kernel [http://www.simtec.co.uk/products/SWLINUX/files/WebKioskEmbeddedSystem/vmlinuz-2.6.26-1-686] and generated output [http://www.simtec.co.uk/products/SWLINUX/files/WebKioskEmbeddedSystem/webkiosk.gz] for an x86 system are available.

6. Putting it on a real system

It was decided to test the system on a real machine. A Lenovo ThinkPad T61 was available. This machine has an Intel i915 chipset which is supported by the intelfb driver resulting in a usable graphical framebuffer.

As in the first article, syslinux is used to make a USB stick bootable. The `syslinux.cfg` used was:

```
default webkiosk
timeout 100
prompt 1

label webkiosk
  kernel VMLINUZ
  append initrd=WEBKIOSK root=/dev/ram video=intelfb vga=0x317
```

The laptop was booted from the USB stick and the web browser started as expected with the correct colours.

7. What's next?

This third example extends the ideas from the previous two articles and produces a useful embedded system. It also demonstrates that as the ways a user interacts with the system become more diverse the system's complexity rises. Finally there is a also an indication that as the target system's hardware becomes diverse (e.g. multiple possible video cards) the number of drivers must also rise and unexpected behaviour is more likely to be experienced as demonstrated by the odd colours from the QEMU video card emulation.

The next step will be to apply the web kiosk example to a specific embedded hardware platform and compare how this alters the systems construction and usage.

8. About the authors

Vincent Sanders	Vincent is the senior software engineer at Simtec Electronics and has extensive experience in the computer industry. He has worked on projects from large fault tolerant systems through accounting software to right down to programmable logic systems. He is an active developer for numerous open source projects including the Linux kernel and is also a Debian developer.
Daniel Silverstone	Daniel is a software engineer at Simtec Electronics and has experience in architecting robust systems. He develops software for a large number of open source projects, contributes to the Linux kernel and is both an Ubuntu and Debian developer.
Simtec Electronics [http://www.simtec.co.uk]	Simtec is a full solutions provider with a proven track record of helping clients with all aspects of a project, from initial concept and design through to manufacturing finished product. With 20 years in the industry, and producing ARM CPU modules since 1992, Simtec's wide experience in embedded systems and the Linux kernel provide a strong base on which to build custom hardware and software solutions, from the smallest of USB devices to the largest complex Linux systems. Simtec's custom-off-the-shelf design service, utilising a range of pre-designed modules of various functions, allows for rapid design and prototype turnaround, reducing time-to-market. Simtec also provide a full software development consultancy with an extensive range of products from boot loaders to full Linux based operating system environments and a range of development boards showcasing Simtec's modular designs.