VxWorks Real-Time Kernel Connectivity
An Internet Toolkit for Remote Device Testing

Michael Lekon
Instructor: Janusz Zalewski
Fort Myers, Florida
12/4/2009
# Table of Contents

1 Introduction ........................................................................................................................................ 3
  1.1 Overview ..................................................................................................................................... 3
  1.2 Project Status .............................................................................................................................. 4

2 Problem Description .......................................................................................................................... 6

3 Solution ............................................................................................................................................. 8
  3.1 Wind River SBC PowerQuiccII ..................................................................................................... 10
  3.2 Simple WEB Server ..................................................................................................................... 12
  3.3 VxWorks Simulator ....................................................................................................................... 13
  3.4 Web-Based Interface .................................................................................................................... 14

4 Experiments .................................................................................................................................... 17

5 Conclusion ..................................................................................................................................... 19

6 References ...................................................................................................................................... 21
1 Introduction

1.1 Overview

Most consumers are familiar with the function of a modern computer: It allows one or more applications to run in parallel, each acting independently. This seemingly basic functionality is the product of the operating system’s core elements, referred to as the kernel. It’s the kernel which allows multiple applications to run at the same time, even when the system has only a single processor. This is done by dividing the processor’s time between the active applications. Since these time slicers are usually only a few milliseconds, the result is that the applications appear to be running in parallel to the user.

In most operating systems which run on a typical PC, the kernel treats all of the applications equally, giving them equal amounts of processor time. This method is sufficient for everyday Internet browsing or word processing, since the user probably wouldn’t even notice a 100 millisecond delay when typing a sentence. In other situations, however, a 100 millisecond delay might mean the difference between life and death when piloting a super-sonic jet. There is a class of operating systems designed specifically for handling this type of situation essentially in real-time. These real-time operating systems are made to ensure that tasks are completed within their deadline.

One of the more popular real-time operating systems is VxWorks. VxWorks was created by Wind River Systems in 1987 and, thanks to its versatility, has been used in Linksys wireless
routers, Boeing airliners, and even several of the Mars rovers. The ability of VxWorks to handle many applications in a timely and efficient matter makes it ideal for applications which interact with multiple external devices[3].

While the applications which use external devices may be ably managed by the operating system, initiating the applications would require user input to specify any parameters needed to direct the device’s function. In real world situations, it’s not reasonable to assume a user will always be physically present to do this when a device needs testing. Allowing users to interact with the VxWorks computer remotely, however, would provide much more reasonable expectations from the users and could be used to simplify the process overall.

In this document, an Internet-based toolkit is outlined. This toolkit allows users to connect to a centralized server which is connected to one or more VxWorks computer, which are connected to one or more external devices. This centralized server provides an interface through which a device and its parameters may be selected and sent to the appropriate VxWorks computer. The results of this are then returned to the user[2].

1.2 Project Status

This project was started by Joanne Sirois during the Spring 2009 semester [1]. As of November 20, 2009, the following tasks have been accomplished:

- A computer to run VxWorks has been procured (the “target” computer)
- A computer to act as the centralized server has been selected (the “host” computer”)
- The host has been configured for a remote desktop connection to facilitate remote development
- The development environment on the host for development on the target has been configured
- A web server application for the target has been found and compiled
- Communication between the host and target has been established for development purposes
- The code for the CGI of the web server has been corrected and is functioning properly
- The VxSim simulator has been configured to work with a network connection, allowing for much faster testing

Tasks currently being performed include:

- Reworking everything done using the simulator to function on the SBC
- Selecting one or more devices to connect to the target
- Installing and configuring a web server on the host
- Creating a web interface on the host to the target
2 Problem Description

As with any device, data acquisition (DAQ) systems have a chance for failure or erroneous operation. Quick detection of these circumstances occurring is often critical for maintaining productivity and/or safety. A problem arises, though, when these systems are expected to be autonomous. How can we detect and signal errors occurring in the system when there is no human operator? One solution proposed by E. Desavouret and J. Nogiec [2] is to create a type of toolkit which gives technicians the ability to see the status of DAQ devices quickly through any computer with an internet connection.

There are multiple pieces of software which are required to make this system function correctly. At each step within the system, certain functionality is required.

At the client:

- The client shall be able to connect to the web toolkit with a computer which has an Internet access and a modern web browser.
- An HTML-based graphical user interface (GUI) shall be provided for all end-user interactions
- A list of devices connected to all targets computers will be presented
- Upon selection of a device, the user shall be directed to the web page corresponding to the specified device
- The user shall be able to select the desired parameters for the chosen device before submitting them for the device to operate on
- A response shall be presented to the user which reflects the results of the requested test

At the host:
• It shall forward user requests to the appropriate target
• Web pages shall be sent to the client as they are requested
• Security shall be implemented by issuing a user name and password to all potential users

At the target:

• The web server shall run continuously
• CGI applications shall be executed
• The resulting text output shall be given to the host
This project will provide an internet-based utility for remotely checking the status of data acquisition (DAQ) devices connected to a computer using the VxWorks real-time operating system developed by Wind River. The basic configuration of this system is shown in Figure 1.

![Conceptual layout of the project's hardware.](image)

The target system (the computer running VxWorks) maintains connections with the internet, a host computer, and a series of DAQ devices. A web server process runs on the target, accepting HTTP requests from clients on the internet. The client then requests data concerning one or more of the DAQ devices. The target queries the specified device(s) and sends the data to the client. The sequence of events when a device request is made is shown in Figure 2, with yellow
Within this figure, the red outlines indicate the surrounded device performs some action while the red arrows indicate the flow of data from the previous device into the next. Once the requested device finishes its task(s), this process works in reverse until it reaches the client that requested it.

The client requests the use of a device from a web interface on the host computer. This is created from a fairly simple set of HTML forms which collect data about which device, which function of that device, and which parameters for that function to use. Figure 3 shows a sample of what one of these forms may look like.

**Figure 2: The sequence of events when a request for a device is made**
3.1 Wind River SBC PowerQuiccII

VxWorks and the web server run on a Wind River SBC PowerQuicc II (Figure 4), a small computer which exists on a single board, which is intended to hasten the development of embedded systems.
Due to the design of VxWorks, a host computer is required to interact with the system. This interaction is accomplished by a RS-232 and network connections between the host and target and the Wind River Workbench software. The workbench is an integrated development environment (IDE) for development on VxWorks-based systems. In addition to the typical text editor, compiler, and linker, the workbench also integrates communication with the SBC, allowing the user to change the boot settings, load and unload various software modules, debug applications remotely, and issue specific commands via a remote shell.

The SBC, while it has ample volatile memory, has very little permanent memory. This means that it cannot maintain a file system of its own, instead using the host’s for accessing required files, including the file containing the VxWorks image (a snapshot of the operating system’s components). Before the system can boot up, this file must be specified using the RS-232 connection and a terminal program (one is available in the workbench) to interact with the boot menu and alter the parameters.
The network connection between the systems is used to transfer files such as the VxWorks image, but an FTP (file transfer protocol) server program (one is provided along with the workbench) must be active on the host to do this. As a result, programs running on the SBC will be able to access the host’s file system (though only the directories specified in the boot parameters) as if it existed locally.

Once the system has booted successfully and a connection has been established from within the workbench, software modules may be downloaded to the memory of the SBC. Executables within these modules may then be run with a command from the shell.

### 3.2 Simple WEB Server

Due to the relatively large differences between VxWorks and more conventional operating systems such as Windows or Linux, the selection of compatible web server applications is rather small. The web server chosen is titled “Simple WEB server” and was written by Jerzy Nogiecz in 1999. As the name suggests, it is a very basic single-threaded web server, providing little more than document retrieval and CGI (common gateway interface) functionality.

The age of this code combined with the relatively new compiler of the workbench has created numerous problems whose causes have been difficult to pinpoint. Simply getting the web server into full working order has taken several months. While there are still security and efficiency concerns (CGI programs run uncontrolled within the same thread as the web server, for instance), its current state is sufficient for a proof-of-concept demonstration.

The CGI functionality of this web server operates differently than most others. First, since the CGI programs run as a function within the web server application and the web server was written in C, all CGI programs must also be written in C. Another consequence of this is that, the
program must be represented by a function, rather than a binary file. In order for the target to be aware of a function, it must be present in the global system table which is updated whenever a new module is downloaded to the target. Currently, a second VxWorks module is under development which contains all CGI programs (functions) which are downloaded to the target alongside the web server.

3.3 VxWorks Simulator

Included with the Wind River workbench, is the VxWorks simulator (Figure 5). This simulator creates a virtual machine running VxWorks within the host. After creating a VxWorks image designed to work with the simulator, boot parameters can be set, and it may be started from the Workbench as if it were a physical device. Since there is no need to work with the actual hardware when using a simulator, this makes development faster and easier.

![Figure 5: VxWorks simulator shell window](image-url)
A VxWorks simulator cannot maintain a network connection on its own. Instead, the VxSim network daemon must be installed an running when starting a simulator which requires network access. By default, a private network within the host is created for the simulators, meaning they are visible only to the host. This simulates a portion of the security that would be implemented on the hardware. Users cannot directly interact with the targets, meaning that undesirable requests can be filtered entirely from the host.

Currently, two simulators are configured to represent two target computers. When both of these are started, and a network interface is available (from the VxSim network daemon), the web server module can be downloaded and run. These two simulators have been used in the demonstrations seen throughout this document.

### 3.4 Web-Based Interface

To allow users to access the system, the host must maintain a web server application and web site of its own. The host computer runs the Windows XP operating system, which allows a wider variety of web servers to be used with it. The Apache 2.2 web server has been installed and configured on the host. It is currently set up such that the HTTP daemon must be started manually after logging in. This is to reduce the risk of conflict over port 80 with other users’ applications.

The web site is currently very simple. It consists of a main page with two links to pages, one for each simulated target (Figure #6). On each target’s page is a list of theoretical devices connected to them. When the user selects a device and clicks the “Submit” button, a PHP script on the host accesses the desired CGI program (function) on the private network to which the simulators are connected, receives the result, and passes it through to the user.
The disadvantage of using VxWorks simulators in this way is that they cannot directly communicate with any of the devices this system was meant to work with. Until hardware implementation is possible, several CGI programs represented a theoretical device have been developed to simply demonstrate the process of accessing them. Figure #7 shows sample output from a theoretical boiler monitoring system. This display was generated by a CGI programming running on a simulator.
Boiler System Analysis:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feed Pump #1 Availability</td>
<td>Online</td>
</tr>
<tr>
<td>Feed Pump #2 Availability</td>
<td>Online</td>
</tr>
<tr>
<td>Feed Pump #3 Availability</td>
<td>Online</td>
</tr>
<tr>
<td>Feed Pump #4 Availability</td>
<td>OFFLINE</td>
</tr>
<tr>
<td>Feed Pump #1 Flow (lbs/hr)</td>
<td>3500.00000000</td>
</tr>
<tr>
<td>Feed Pump #2 Flow (lbs/hr)</td>
<td>2710.00000000</td>
</tr>
<tr>
<td>Feed Pump #3 Flow (lbs/hr)</td>
<td>0.00000000</td>
</tr>
<tr>
<td>Feed Pump #4 Flow (lbs/hr)</td>
<td>0.00000000</td>
</tr>
<tr>
<td>Boiler Tank Water Level</td>
<td>11342 / 18500</td>
</tr>
<tr>
<td>Rate of Steam Flow (lbs/hr)</td>
<td>6221.600000</td>
</tr>
<tr>
<td>Up Time (hour:min:sec)</td>
<td>13:14:32</td>
</tr>
</tbody>
</table>

Manual Pump Control

<table>
<thead>
<tr>
<th>Pump #0</th>
<th>ON</th>
<th>OFF</th>
<th>NO CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump #1</td>
<td>ON</td>
<td>OFF</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>Pump #2</td>
<td>ON</td>
<td>OFF</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>Pump #3</td>
<td>ON</td>
<td>OFF</td>
<td>NO CHANGE</td>
</tr>
</tbody>
</table>

Submit Query

Figure 7: Sample CGI output from a theoretical boiler monitoring system.
4 Experiments

Essential to the project is the ability to trigger the execution of programs on the target. The CGI of the web server now provides this after multiple modifications to the original code. To experiment with this newly added functionality a simple CGI program was written and downloaded to the SBC, the code of which is provided below.

```c
int quadratic(void){
    float a, b, c, r1, r2, disc;
    char* data = getenv("QUERY_STRING");
    sscanf(data,"a=%f&b=%f&c=%f", &a, &b, &c);
    disc = b*b-4*a*c;

    printf("<html>");
    printf("<head><title>CGI<title></head>");
    printf("<body>");
    printf("discriminant = %f<br />", disc);
    printf("a=%f<br />b=%f<br />c=%f<br /><br />", a, b, c);

    if(disc >= 0) {
        r1 = (-b + sqrt(disc)) / (2*a);
        r2 = (-b - sqrt(disc)) / (2*a);
        printf("root 1 = %f<br />");
        printf("root 2 = %f<br />");
    }
    else
        printf("Non-real roots");

    printf("</body>");
    printf("</html>");
    return 0;
}
```

This program takes 3 parameters, a, b, and c from the QUERY_STRING variable and uses the quadratic equation to calculate the two roots (if they exist). With this function available to VxWorks and the web server running, it can be called with parameters passed by the GET method. The result is shown below (Figure #8) with randomly chosen parameters.
Figure 8: Demonstration of the web server's CGI capabilities

discriminant = 41.000000
a = -2.000000
b = -1.000000
c = 5.000000

root 1 = 1.850781
root 2 = 1.350781
5 Conclusion

While the currently implemented system provides communications to and from a VxWorks system, providing the extensible platform for device testing that is desired will require much more work.

The target web server has, by far, been the largest hurdle in getting the system to its current functionality. The code for CGI functionality in the Simple WEB Server, originally, did not function in the slightest degree, despite this application supposedly being used successfully in the past. Whether this was caused by differences in the C standard used or a product of the Workbench compiler’s VxWorks-specific nuances, I can’t be certain, but even now with basic functionality restored, there is ample room for improvement. Simply being able to run CGI programs from a separate file would go a long way to make the whole system more extensible.

Despite repeated attempts to compile and implement a more recent, better maintained web server (K Lone and GoAhead servers primarily), they have presented even more problems than the Simple WEB Server. One of these other web servers would be ideal, but their limited documentation and little online support may simply cause more trouble than they’re worth.

The lack of any actual external devices as well as hardware difficulties has also dampened progress. The SBC provides only a single USB, 2 serial and 2 Ethernet ports. In order to utilize the advantages of a VxWorks-based system, the capability to connect more devices must be present. Also, while the CGI functionality has worked on the simulators, it has been less successful when used on the SBC. This is due to a VxWorks image which does not provide all of the operating system functions required by the code. Despite compiling and downloading new images in multiple configurations, none except the outdated image will function properly.
In addition to the issues with the image, the VxWorks computer seems to have had a hardware failure or catastrophic error in its permanent memory. This forced development to focus on the web interface which, while essential to the system, doesn’t require much effort to provide the necessary functionality. Configuring the devices and writing the CGI program to interface with them was expected to be the most difficult aspect of this project.
6 References