VxWorks Networking in Data Acquisition
Joanne Sirois & Evan Flechsig
Dr. Zalewski CNT 4104
December 16th, 2008
**Table of Contents**

I. Introduction .......................................................................................................................................... 3  
   - Basic Instruction of Workbench ........................................................................................................ 5  
   - Introduction to WindRiver VxWorks Simulator........................................................................ 10  
II. Problem Description ........................................................................................................................... 11  
III. Solution & Design ............................................................................................................................ 12  
   - Socket Client/Server ......................................................................................................................... 12  
IV. Implementation .............................................................................................................................. 14  
Sources ........................................................................................................................................................ 15  
Appendix A .................................................................................................................................................. 16  
Appendix B .................................................................................................................................................. 35
I. Introduction

Do respirators in hospitals or air plane computing systems run off of Windows? To most, the answer to this question is obviously no, but they are indeed run by a computer of some kind. Well, software needs to run on some kind of computer hardware, so what kind of hardware is used for critical operations such as hospital and airplane equipment? They would be run on an embedded system. An embedded system is a type of computer system or computing device that performs a dedicated function and/or is designed for use with a specific embedded software application. These systems are used because of their efficiency (in items such as TV remote, garage door openers, etc), and most often for their reliability (respirators, airliner GPS systems, etc).

However, creating a new physical machine every time a specific program is updated is going to be very costly. This is why embedded operating systems, also known as real-time operating systems, were developed. They provide the reliability of an embedded system, and are easier to revise. Some real-time operating systems include QNX, RTLinux, and VxWorks. We use VxWorks for our project. An example of the embedded single board computer (SBC) running the VxWorks operating is shown in Figure 1 below.
Basic Instruction of Workbench

VxWorks has been developed by Wind River Systems and was designed specifically for use in embedded systems [4]. It is being used by organizations such as National Aeronautics and Space Administration (Mars Reconnaissance Orbiter) [3], Honda (ASIMO robot), Linksys (wireless router), and Boeing (787 airliner), and many others. This operating system supports x86, MIPS, PowerPC, SH-4, ARM, StrongARM, and xScale platforms. VxWorks tasks can be programmed in multiple programming languages, which include Ada, C, C++, and Java. This operating system gives users a unique opportunity to run programs at the kernel level. These programs can be developed using an integrated development environment, WorkBench.
Workbench is based off the Eclipse’s framework\textsuperscript{[5]}, as seen in the familiarities illustrated in Figure 2; it also has its own kernel configurator. Using this tool to develop software for VxWorks is important because a kernel is what gives the hardware the ability to communicate to software applications\textsuperscript{[2]}. The Workbench software runs on a PC Host to develop software for a target computer (SBC83xx Power QUICC II). The SBC83xx is connected to a server via serial cable and Ethernet cable. It uses two connections because the IDE requires the serial to confirm the connection; however the SBC83xx retrieves the program that you write through FTP. A diagram representing this set up can be seen in Figure 3.

![SBC Connectivity Diagram](image)

**Figure 3 - SBC Connectivity Diagram**

The first step in putting the system to work is to check the physical connection of the SBC host. After confirming the physical connection of the SBC to the host, Workbench has to be started, and then the FTP client. A screen shot of the FTP log can be seen in Figure 4. The log shows the certain tasks that it has done, such as accepting proper passwords, finishing transfers, and showing when a user account has been logged on. This is helpful for trouble shooting, as we encountered a situation where a password was mistyped, which we picked up on quickly because it was shown in the FTP log. As an extra check one can attempt to ping the targets IP address in Windows DOS command. Our IP address at the moment is 61.88.32.21.
Now we go back to Workbench. If the terminal tab is not available in the southern window, then enable it by navigating to Window > Show View > Terminal. You should now see the Terminal tab as shown in Figure 3.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:23</td>
<td>Connection accepted from 150.164.238.131</td>
</tr>
<tr>
<td>00:23</td>
<td>Command &quot;USER target&quot; received</td>
</tr>
<tr>
<td>00:23</td>
<td>Command &quot;TYPE I&quot; received</td>
</tr>
<tr>
<td>00:23</td>
<td>Command &quot;PASS&quot; received</td>
</tr>
<tr>
<td>00:23</td>
<td>Command &quot;RETR c:\work\work\works&quot; received</td>
</tr>
<tr>
<td>00:23</td>
<td>RETrieve started on file c:\work\work\works</td>
</tr>
<tr>
<td>00:23</td>
<td>Transfer finished</td>
</tr>
<tr>
<td>00:44</td>
<td>Got file c:\work\work\works successfully</td>
</tr>
<tr>
<td>00:44</td>
<td>QUIT or close - user target logged out</td>
</tr>
<tr>
<td>00:59</td>
<td>Command &quot;USER target&quot; received</td>
</tr>
<tr>
<td>00:59</td>
<td>Command &quot;PASS&quot; received</td>
</tr>
<tr>
<td>00:59</td>
<td>Type set to I</td>
</tr>
<tr>
<td>0:059</td>
<td>Command &quot;RETR c:\work\work\works&quot; received</td>
</tr>
<tr>
<td>0:059</td>
<td>RETrieve started on file c:\work\work\works</td>
</tr>
<tr>
<td>0:059</td>
<td>Transfer finished</td>
</tr>
<tr>
<td>0:059</td>
<td>Got file c:\work\work\works successfully</td>
</tr>
<tr>
<td>0:059</td>
<td>QUIT or close - user target logged out</td>
</tr>
</tbody>
</table>

Figure 4 – FTP Log

Figure 5 – Terminal Window
You then need to configure the target client. In the Remote Systems View in Workbench, click on “Define a connection to a remote system” if the SBC is not already defined in the list of systems. Chose “WindRiver VxWorks 6.x Target Server Connection” and then click next. Fill in the SBC’s IP address. The screen should be similar to the one shown in Figure 5. Then click OK.

![Target Connection properties](image)

**Figure 6 – Target Connection properties**

Now connect to the target by clicking the green button on the top right of the terminal window or right clicking the target you just configured in the remote systems window. This will
result in uploading the kernel image to the target, which is illustrated through the FTP client (as shown in Figure 3), terminal window (as seen in Figure 6) and the ping command.

![Figure 7 – Terminal of Target via serial connection](image)

You can then implement a program in Workbench. A simple “Hello World” program was executed in Figure 7. To create a new project in Workbench, go File -> New -> Downloadable Kernel Module Project. Choose a project name, hit “Next” until you get to the Build Specs screen, and deselect any build specs without PPC in their name (since this SBC is a PPC architecture) and hit Finish. Then right click the folder and choose New > File. Name the file and code the project. To compile the project right click the project folder and select build, click Generates Includes and the Finish. After the code is compiled you can send the task to the target by right clicking and choose run as kernel task. Alternatively, you can download the program and execute the program through the targets shell, but that will be covered in the socket section of the paper.
Introduction to WindRiver VxWorks Simulator

Another tool used in developing programs for VxWorks is the VxWorks Simulator 6.2. Known as VxSIM, it is a complete prototyping and simulation tool for VxWorks 6 applications. It enables development and testing of significant portions of an application earlier in the development cycle, before hardware is available. The simulator is fully integrated into the Wind River Workbench development environment as a target connection, allowing complete configuration and debugging control through standard interfaces. 4

VxWorks Simulator is a native application that is derived from the VxWorks 6 operating system to accurately implement the sophisticated features of VxWorks 6, including real-time processes, memory protection, file systems, and UNIX-style networking (TCP/IP, rlogin, etc.). The simulator also provides network simulation capabilities that let you create complete simulations of complex networks consisting of multiple IPv4, IPv6, or other protocols, subnets, and routing systems.4

The simulator runs in the host workstation, decreasing the quantity of evaluation hardware typically purchased for early development. It also provides easy access to the host operating system API, so you can use the host facilities and peripherals in your simulation.

Features of VxWorks Simulator 6.2 include4:

• Rapid and precise prototyping of VxWorks 6 applications
• Support for user-level IP network simulation of complex network topologies
• Broadcast and multicast support
• Simulation of packet loss
• Support for simulation of a large number of nodes
• Support for VxWorks file systems
• Integration into Workbench development environment
• Support for broad range of host platforms
• Access to the host operating system API from within the VxWorks simulation
• Support for real-time clock
• VxWorks boot parameter parsing at startup to allow multiple instances of a single image to exist

II. Problem Description

When using automated data acquisitions systems, they need to be monitored frequently for continuation of operation. This is important because data acquisition’s whole purpose is to take reliable readings using specific instruments. If this function is off even slightly, the results could be considered completely worthless. That is why issues have to be resolved as quickly as possible. One solution proposed by E. Desavouret and J. Nogiec 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\[1\].

In their solution, they have a host web server which collects information from a set of data acquisition devices that have their own specialized web servers installed on them. The host server takes the information from the devices and publishes it to a viewable webpage, so that it can be accessed from a client computer with an Internet connection. This is what we’re going to try to emulate, with our SBC box being our DAQ device, and the HP desktop being our host server.

Our main challenge in emulating the works of Desavouret and Nogiec is going to be finding a “customized web server” to install on the VxWorks machine, which is configured to communicate to the HP desktop machine.
III. Solution & Design

In order to execute code remotely, we first need to demonstrate that we can execute a program that uses sockets or any other program that receives information from a remote server. For this, we will run a time client on the VxWorks machine and attempt to retrieve the time from satnet.fgcu.edu. To demonstrate this, we run a sample socket server program on the target, then run the client socket program on a remote server (in this case, satnet.fgcu.edu). Our second demonstration uses VxWorks target as a server.

Socket Client/Server

Like many other socket server-client relationships, this example has the server waiting for some action from the client, to where it does something with this information, and returns an output to the client. In this case, the client can send a message to the server, to where the server prints out this message in the terminal. The client is also given an option for a confirmation message from the server, which is sent to the client if the server successfully received the message. Figure 9 is a screen shot of the targets shell with the programs outputs displayed.

In order to run this demo, you first had to get the binaries of both of the programs, client and server, in the targets memory. This is accomplished by expanding the projects where the programs are located, and then right clicking the binaries for each program after it is built, and then selecting the “Download” option. Work bench will then ask for a destination for these files, which is when the already configured target should be selected.

Once downloaded, the shell must be started for the target in order to invoke the demo server process. The shell gives the user direct access to the target, giving them the ability to launch processes and view directory, among other things. Then, type `sp tcpServer`. “sp” stand for “spawn process”, and “tcpServer” is the name of the binaries of the process we’re starting.

Next, we launch the client by typing `tcpClient targetServer`, you don’t use since we are not starting a process. “tcpClient” is the name of the binaries of the program we’re starting, and “targetServer” is a parameter that will be passed to the program, which is the address of
the machine where the tcpServer process resides. For this demo, the machine running the server process is the local host, which is why we use “127.0.0.1” in the demo.

The next step is to run the server on VxWorks target and the client on another computer in the network, such as satnet.fgcu.edu.

![Figure 8 - Example Socket program](image)
IV. Implementation

What is done in the implementation segment of this project is to show demonstrate communication via sockets. The client side of the relationship will be running on the target waiting for the connection to be established by the client. Once the connection is established, the server will then send a string to the client, which the client will then print out. The server will also print out that the message has been sent to the client. The source code is located in Appendix B. To run the demo, you will need to do the following:

1) Start the FTP server so the target can get the kernel image from the host.
2) Turn on the target.
3) Start “Work bench”, the VxWorks IDE.
4) Connect to a remote server (this will be our client) through SSH.
5) Download the client2.c file to a directory.
6) Download the server binary to the target after it’s been building using the GNU build specification.
7) Compile the client using “gcc –o client2 client2.c” in the directory where the source is located. This will create the object which you will be running later.
8) To run the server program on the target, open up the targets shell and enter messServer “22”. This starts the server and it is now listening on the port specified.
9) Run the client program on the remote client. In our case, we execute ./client2 22
10) The server will then send a message to the client, and the server will show that the message was sent.

We did run into a problem with the server program though running on the target. For some reason, when the shell is closed and reopened, the program will not start correctly. The target then requires a restart in order to successfully run the server application. This might be because there is no official way to end the program.
Sources
Appendix A
SBC 8349E Web Access for VxWorks Development via Wind River WorkBench 3.0
Author: Victor Elci
Editor: Evan Flechsig, Joanne Sirois

Introduction
The SBC 8349E is a single board computer with PowerPC architecture capable of running either a VxWorks or Linux board support packages. The SBC is equipped with 2 serial ports, 2 Ethernet ports and a USB port. Wind River Workbench is an IDE based off of Eclipse which is used to interface with the SBC via a terminal and to program the SBC over an Ethernet connection.

Before developing programs for the SBC, the board must be powered on and have the VxWorks operating system loaded into its memory and before utilizing the Workbench software, the licenses must be working. The instructions following will highlight how to set up and get the SBC running.

Installation and Configuration of the License Server (if not installed already)
Before the Wind River Workbench IDE may be used with floating licenses, the License Server must be installed and configured.

On the Wind River website, you must first login to the licensing section “http://www.windriver.com/licensing/” and add the server's info in the “Manage Hosts” section. Then you must go into the “Manage Licenses” section and assign some licenses to the server. Save the generated license file to the server. I placed my file somewhere in the WindRiver installation directory.

Then install the license server from the CDs. After installation has completed, run “lmtools.exe”
which is typically located in “C:\WindRiver\licadmintools-1.2\license\x86-win32\bin”. On the “Server/License File” tab, make sure “Configuration using Services” is selected. On the “Config Services” tab, set the path to the “lmgrd.exe” file (usually something like “C:\WindRiver\licadmintools-1.2\license\x86-win32\bin\lmgrd.exe”) and the path to the license file that was generated from the website (mine was located in “C:\WindRiver\license\WRSLicense.lic”). On the “System Settings” tab, fill in Computer/Hostname, Username, IP Address, Ethernet Address, and Disk Volume Serial Number if they have not already been filled in. Now, you should be able to start the server on the “Start/Stop/Reread” tab.

**Installation of the Workbench IDE (if not installed already)**

After the License Server has been set up, the Workbench IDE must be installed on your workstation.

It is recommended that the “C Development Toolkit” for Eclipse is also installed. Instructions on how to do that can be found in the Workbench help files and have been copied in this documentation and placed after the references and are located on page 8.

**Start the License Server**

If the “lmgrd.exe” process is visible in the task manager, then the license server must be started. The easiest way to do this is to use the “lmtools.exe” which is typically located in “C:\WindRiver\licadmintools-1.2\license\x86-win32\bin”.

On the “Start/Stop/Reread” tab, you can start the license server.

Make sure that the board is properly connected and on

A typical setup consists of the board being connected to the web via a router through an Ethernet cable plugged into port A and a serial connection through its COM1 port. A sample setup is shown below in Figure 1. There are two power buttons. One is a switch located by the power cord. The other is located by the ports on the front.
Make sure the FTP server is running/configured

1. Start up an FTP server on your computer. There should be one that is installed with Workbench if you do not already have one. The following assumes you are using the FTP server installed with Workbench.

2. Go to Security -> Users/Rights to set up an account for the SBC to access the FTP server.

3. Click New User, and input the user name. For example, I entered “target”.

4. Then set the home directory and make sure the VxWorks image is somewhere in that home directory (The VxWorks image should be available off the CD that comes with the board. If that has been lost, then they should also be available off the Wind River website.) For example, I put my VxWorks image in “C:\Temp\VxWorks” and set my home directory as “C:\Temp”

5. If prompted, also set the password. I set mine to “qwerty123” for now.

6. Next go to Logging -> Log Options and check everything except “Winsock Calls”. This will help debugging connection problems later on.

7. Find the IP address of the computer you are working on by typing `ipconfig -all` in the command line. This will be used later when loading the VxWorks image.

Start the Workbench

Configure the target board settings and load the VxWorks image onto it using the terminal (via the serial port)

1. Connect the board to a computer which has Wind River Workbench 3.0 installed on it via a serial cable.
2. Start up the Workbench software.

3. If the terminal tab is not available in the southern window, then enable it by navigating to Window > Show View > Terminal. You should now see the Terminal tab as shown in figure 2.

4. Click the yellow and blue settings icon to change the settings. The settings should be as in

![Figure 3: Terminal Tab](image)

5. Click the yellow and blue settings icon to change the settings. The settings should be as in
6. Click the green connect button to initialize the connection to the SBC.

7. If the connection is successful, you will encounter a boot menu.

**Boot Menu Configuration**

I will explain two ways to load the VxWorks image onto the SBC. The first method involves loading the VxWorks image from the machine the SBC is hooked up to via the serial connection. The three commands you need to know for now are: “?” – prints list of commands, “p” – prints boot parameters, “c” = change boot parameters, “@” – load VxWorks image and boot.

**Boot Menu Configuration w/ Local FTP Server**

1. Enter `?` in the terminal to copy down the available boot devices. The SBC listed mottsec0 and mottsec1 for me.

2. Type `c` in the terminal to modify the boot parameters.
   a) **Boot device** should be mottsec0 or mottsec1 (or another boot device if it is listed)
   b) **Host name** is the name of the computer hosting the VxWorks image
   c) **File name** is the complete file path to the VxWorks image including the file name (example from earlier: “C:\Temp\VxWorks\VxWorks”)
   d) **Inet on Ethernet** is the IP address of the board. If working on your router at home, you can usually arbitrarily set this as an IP address in the range of your router, but you must get assigned an IP address to the board's MAC addresses for usage on campus.
   e) **Inet on Backplane** is not used unless the SBC is mounted on a rack.
   f) **Host Inet** is the IP address of the computer hosting the VxWorks image.
   g) **Gateway Inet** is not used in this scenario, but can be found as the Default Gateway when you do an `ipconfig -all` in the command line. Seems to be needed if the board is run on campus.
   h) **User** is the username of the ftp account you set up.
i) **FTP password** is pretty self-explanatory.

3. The rest of the fields may be left blank. You may name the SBC if you wish though.

Launch VxWorks

1. Type in `@` the terminal to load and boot VxWorks.
2. Once the Boot Menu Configuration has been completed, the SBC will attempt to auto-load the VxWorks image every time it is turned on. This can be circumvented by pressing any key during the countdown.

Making Projects in Workbench *(Hello World Example)*

1. In Workbench, **File -> New -> Downloadable Kernel Module Project.**
2. Enter the project name, for example, “Hello World”
3. Hit next until you get to the Build Specs screen, and deselect any build specs without PPC in their name (since the SBC is PPC architecture).
4. Hit Finish to create the project.
5. Right click the folder of the new project in the Project Explorer and then go to New -> File.
6. Name the file something such as “Hello World.c”.
7. Code to insert is located in the code reference section.

**Compiling**
1. Right-click the project's folder in the Project Explorer once again and select Build Project.
2. Click Generate Includes and then click next, next and finish.

**Connecting to the target (via Ethernet)**
3. In the Remote Systems View, click on “Define a connection to a remote system” if the SBC is not already located in the list of systems. See figure 4.
4. Select “WindRiver VxWorks 6.x Target Server Connection” and then click next.
5. Fill in the SBC's IP address. The screen should be similar to the one shown in figure 5. Then click OK.
Running a program

1. If a previous module has been loaded, it must first be removed from the target. Right-click the module which is currently running on the target and select delete.

3. Right-click the project’s folder in the Project Explorer once more and select Run Kernel Task

4. The entry point must be changed to whatever function you want your program to start execution within the Entry Points pop-up window. Change the entry point to “HelloWorld” in your “HelloWorld” project folder and leave the rest of the fields alone.

5. Click Run and you should see some output in the Terminal window.

Creating and Running VxWorks Projects

This next section will explain how to create and run a sample C project. A more advanced example is included in the Workbench documentation.
1. In Workbench, File -> New -> Downloadable Kernel Module Project.
2. Enter the project name, “HelloWorld”
3. Hit next until you get to the Build Specs screen, and deselect any build specs without PPC in their name (since the SBC is PPC architecture).
4. Hit Finish to create the project.
5. Right click the folder of the new project in the Project Explorer and then go to New -> File.
6. Name the file something such as “HelloWorld.c”.

    void HelloWorld()
    {
        printf("hello world\n");
    }

7. Place the text shown below in Figure 6 into the file you just created.

8. Right-click the project's folder in the Project Explorer once again and select Build Project.
9. Click Generate Includes and then click next, next and finish.
10. Right-click the project's folder in the Project Explorer once more and select Run Kernel Task
11. Change the entry point to HelloWorld in your HelloWorld project folder and leave the rest of the fields alone.
12. Click Run and if everything is working properly, there should be the output for your program in the terminal.

References


Depending on how you obtained the CDT, you might have also received a toolchain with a built-in CDT integration. However, if you downloaded the CDT from an update site, then you will
require a toolchain before you can build and debug any projects.
The base CDT supports integration with the GNU toolchain. This includes GNU's make, gcc compiler, and gdb debugger utilities. If you require a toolchain to build software for your development host, this is the best choice to get started.

Each platform that runs the CDT requires different steps to acquire this toolchain.

**Linux**

All Linux distributions include the GNU toolchain. They may not, however, be installed by default. For instructions about installing the GNU toolchain for Linux, see the instructions for your particular distribution.

**Windows**

For windows, MinGW, and Cygwin are the two main choices for acquiring the GNU toolchain:

- **Cygwin** is a port of the Linux environment to Windows. It provides a compatibility layer in a set of DLLs. These DLLs are *GPL licensed*, making any code that links to them also subject to the GPL. Cygwin, however, does provide the fullest implementation of the GNU toolchain by supporting the GNU libc C runtime library.

- **MinGW** is a port of the GNU toolchain to the Windows platform. The biggest difference over Cygwin is that MinGW uses the Windows C runtime libraries (mscvrt) instead of GNU's libc. As a result, a compatibility layer is not required, thus avoiding the GPL issues with Cygwin. There are differences, though, between the Windows and GNU C runtime libraries that will make writing portable applications more difficult.

However, MinGW provides the best integration support with the CDT due to its direct support for the Windows environment.

The following are instructions and links on how to install the current version of MinGW. Note that these links may become inaccurate over time as new versions of MinGW components are introduced. Please check the [MinGW File Release](#) section for the latest versions.

1. Download and run the MinGW setup program, *MinGW-5.1.3.exe*.
2. Select download and install the MinGW base tools and the g++ compiler. You may select the Current or Candidate version of these tools. You may also install any of the other
available compilers as well.

**Do not install the MinGW Make feature** as the MSYS version of make from step 5 is a more complete implementation of make.

3. The MinGW setup program currently does not install the gdb debugger. To install the debugger, download the file from the following location: `gdb-6.6.tar.bz2`

4. Extract the contents of the file `gdb-6.6.tar.bz2` to the same location where you installed MinGW.

5. If you want to use Makefile projects, download and run the setup program from the following location: `MSYS-1.0.10.exe`. MSYS provides an implementation of make and related command line tools. This is not required for other types of projects with the MinGW toolchain, which use CDT's internal build tools to perform the build.

**Other Platforms**

The GNU toolchain is supported on all platforms that the CDT supports. For instructions about installing the GNU toolchain on your platform, see your platform vendor.
Code Reference

1. helloworld .......................................................... 9
2. MultiTask1 .......................................................... 10
3. MultiTask2 .......................................................... 12
4. Multitask3 .......................................................... 14

//helloworld.c
//standard “hello world” program

#include < VxWorks.h>

void helloworld()
{
    printf("hello world\n");
}
//MultiTask1.c
//A semaphore is used to synchronize two separate threads.
//Even thread prints even numbers, Odd thread prints odd numbers.
//Without the semaphore, they do not alternate.

#include "VxWorks.h"
#include "semLib.h"
#include "msgQLib.h"
#include "errno.h"
#include "taskLib.h"
#include "msgQLib.h"
#include "signal.h"
#include "timers.h"
#include "fioLib.h"
#include "stdio.h"
#include "limits.h"
#include "hostLib.h"
#include "netShow.h"
#include "netDrv.h"
#include "ftpLib.h"
#include "remLib.h"
#include "limits.h"
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

SEM_ID numSem;

#define TRUE  1;
define FALSE  0;

int taskEven, taskOdd;

void printOdd()
{
    int i=0;
    while(i<50)
    {
        if (semTake(numSem, 10) == OK )
        {
            printf("Number: %d\n", i*2);
            i=i+1;
            semGive(numSem);
        }
    }
}
void printEven()
{
    int i=0;
    while(i<50)
    {
        if (semTake(numSem, 10) == OK )
        {
            printf("Number: %d\n", i*2+1);
            i=i+1;
            semGive(numSem);
        }
    }
}

void multitask()
{
    taskEven = 0;
    taskOdd = 0;

    if((numSem = semBCreate( SEM_Q_FIFO, SEM_FULL)) == NULL)
    {
        printf( "Error creating semaphore\n");
        exit(0);
    }

    semTake(numSem, 10);

    taskSpawn("even",90,0x0100,10000, (FUNCPTR)printOdd,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
    taskSpawn("odd",90,0x0100,10000, (FUNCPTR)printEven,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

    semGive(numSem);
}

//MultiTask2.c
//A semaphore is used to synchronize two separate threads.
//Create thread creates and stores numbers, Read thread prints the data.
//Without the semaphore, data is overwritten.

#include "VxWorks.h"
#include "semLib.h"
SEM_ID createReadSem;

int taskRead, taskWrite;

int data;

void createNum()
{
    int i=0;
    while(i<10)
    {
        if (semTake(createReadSem, 10) == OK )
        {
            data = rand();
            printf("Create: %d\n", data);
            i=i+1;
            semGive(createReadSem);
        }
    }
}

void readNum()
{
    int i=0;
    while(i<10)
{
    if (semTake(createReadSem, 10) == OK )
    {
        printf("Read: %d\n", data);
        i=i+1;
        semGive(createReadSem);
    }
}
}

int main()
{
    taskRead = 0;
    taskWrite = 0;
    data = 0;

    if((createReadSem = semBCreate( SEM_Q_FIFO, SEM_FULL)) == NULL)
    {
        printf("Error creating semaphore\n");
        exit(0);
    }

    semTake(createReadSem, 10);
    taskSpawn("create",90,0x0100,10000, (FUNCPTR)createNum,0,0,0,0,0,0,0,0,0,0,0);
    taskSpawn("read",90,0x0100,10000, (FUNCPTR)readNum,0,0,0,0,0,0,0,0,0,0,0,0);
    semGive(createReadSem);

    return 0;
}

//MultiTask3.c
//A semaphore is used to synchronize two separate threads.
//Even thread prints even numbers, Odd thread prints odd numbers.
//Without the semaphore, they do not alternate.

#include "VxWorks.h"
#include "semLib.h"
#include "msgQLib.h"
#include "errno.h"
#include "taskLib.h"
#include "timers.h"
#include "fioLib.h"
#include "stdio.h"
#include "signal.h"
#include "netDrv.h"
#include "ioLib.h"
#include "hostLib.h"
#include "netShow.h"
#include "ftpLib.h"
#include "remLib.h"
#include "limits.h"
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

SEM_ID numSem;

#define TRUE  1;
#define FALSE 0;

int taskEven, taskOdd;
int max=-1;

void printOdd()
{
    int i=0;
    while(max!=-1)
    {
        if (max==-1)
            {
            continue;
        }

        if (semTake(numSem, 10) == OK)
            {
            printf("Number: %d\n", i*2);
            i=i+1;
            semGive(numSem);
        }

        if (i>=max/2)
        {
            break;
        }
    }
}
void printEven()
{
    int i=0;
    while(max!=-1)
    {
        if (max==-1)
        {
            continue;
        }
        if (semTake(numSem, 10) == OK && max!=-1)
        {
            printf("Number: %d\n", i*2+1);
            i=i+1;
            semGive(numSem);
        }
        if (i>=max/2)
        {
            break;
        }
    }
}

void multitask()
{
    taskEven = 0;
    taskOdd = 0;

    printf("Input max number: ");

    scanf("%d", &max);

    if((numSem = semBCreate( SEM_Q_FIFO, SEM_FULL)) == NULL)
    {
        printf( "Error creating semaphore\n");
        exit(0);
    }

    semTake(numSem, 10);

    taskSpawn("even",90,0x00100,10000, (FUNCPT)printOdd,0,0,0,0,0,0,0,0,0,0,0);
taskSpawn("odd", 90, 0x0100, 10000, (FUNCPT)printEven, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

semGive(numSem);
Appendix B

//Filename: client.c
#include <iostream>
#include <cstdlib>
using namespace std;

#include <stdio.h> /* perror() */
#include <stdlib.h> /* atoi() */
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h> /* read() */
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

//Structure stores the data about given quadratic equation.
typedef struct{
    double a,b,c;
    double disc;
    double root1, root2;
    bool discisNeg; // If true, quadratic is not solvable
    bool aisZero; // If true, equation is not quadratic
}dsQuadratic;

// Prints the solution
void printQuadSoln(dsQuadratic *quadInfo){
    printf("For the quadratic equation (%g)x^2 + (%g)x + (%g) = 0\n", quadInfo->a, quadInfo->b, quadInfo->c);
    if(quadInfo->aisZero){
        cout << "\n\nThere are no roots. ";
    }
    else if(quadInfo->discisNeg){
        cout << "\n\nThere are no roots. ";
    }
    else{
        printf("\nRoot1 = %g\nRoot2 = %g\n\n", quadInfo->root1, quadInfo->root2);
    }
}

int main(int argc, char *argv[]){
    int clientSocket,
    remotePort,
    status = 0;
    struct hostent *hostPtr = NULL;
    struct sockaddr_in serverName = { 0 }; char buffer[256] = "\n    char *remoteHost = NULL;
remoteHost = argv[1];
remotePort = atoi(argv[2]);

if((clientSocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP))==-1){
    exit(1);
}

hostPtr = gethostbyname(remoteHost);
if (NULL == hostPtr){
    hostPtr = gethostbyaddr(remoteHost, strlen(remoteHost), AF_INET);
    if (NULL == hostPtr){
        perror("Error resolving server address");
        exit(1);
    }
}

serverName.sin_family = AF_INET;
serverName.sin_port = htons(remotePort);
memcpy(&serverName.sin_addr,hostPtr->h_addr,hostPtr->h_length);
if(connect(clientSocket, (struct sockaddr*)
&serverName,sizeof(serverName))==-1){
    perror("connect()");
    exit(1);
}

if(read(clientSocket, buffer, sizeof(buffer))!= - 1){
    cout<<"Server said "<<buffer<<endl;
}else{
    perror("read");
    exit(1);
}

/*
dsQuadratic *p_qInfo = new dsQuadratic;
// Retrieve the quadratic coefficients from the client user
cout << "\nPLease input the coefficients." << endl << endl;
cout << "\ta = "; cin >> p_qInfo->a;
cout << "\tb = "; cin >> p_qInfo->b;
cout << "\tc = "; cin >> p_qInfo->c;
cout << endl;

// Write data to the server
if(write(clientSocket, (void*)p_qInfo, sizeof(dsQuadratic))==-1){
    perror("Writing to Server");
    exit(1);
}

// Read data from the server
if(read(clientSocket, p_qInfo, sizeof(dsQuadratic))==-1){
    perror("Reading from Server");
    exit(1);
}else{
    // Display the solution for the user given coefficients
    printQuadSoln(p_qInfo);
}*/
```c
close(clientSocket);
return 0;
}

/*File name: server.c*/
#include "vxWorks.h"
/**include <cstdlib>*/
#include <stdio.h>   /*
#include <stdlib.h>  /* exit() */
#include <string.h>  /* memset(), memcpy() */
#include <sys/types.h>
#include <sys/socket.h>   /* socket(), bind(),
                     listen(), accept() */
#include <sched.h> // clone
#include <arpa/inet.h> // inet_ntoa
#include <netdb.h> // gethostbyname
#include <unistd.h>  /* fork(), write(), close() */
#include <math.h>

typedef struct {
    int slaveSocket;
    struct sockaddr_in clientName;
} client_struct;

typedef struct{
    double a,b,c;
    double disc;
    double root1, root2;
    int discisNeg; /* If true, quadratic is not solvable*/
    int aisZero; /* If true, equation is not quadratic*/
}dsQuadratic;

dsQuadratic* calcRoots(dsQuadratic*);
/**
* constants
*/
const char MESSAGE[] = "How are you?\n";
const int BACK_LOG = 5;
double ROOTS[2];
STATUS messServer(char *argv){
    int serverSocket = 0,
    on = 0,
    port = 0,
    status = 0,
    childPid = 0;
    dsQuadratic *p_qInfo;
    int nRead;

    struct hostent *hostPtr = NULL;
    struct sockaddr_in serverName = { 0 };
```
if (argv == NULL){
    fprintf(stderr, "Usage: %s <port>\n", argv[0]);
    return(ERROR);
}

port = atoi(argv);

/*
 * create a server socket
 * 1) PF_INET: this socket uses IP protocol
 * 2) SOCK_STREAM: the communication will be two-way and connection-based
 * 3) IPPROTO_TCP: uses TCP/IP protocol
 */
if((serverSocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP))==-1){
    perror("socket()");
    return(ERROR);
}

/*
 * set an option for socket
 * 1) serverSocket: specifies which server socket to set options for
 * 2) SOL_SOCKET: the option will be set on the socket level, not other levels, i.e. TCP level
 * 3) SO_REUSEADDR: when this program ends, can the socket be reused immediately
 * 4) (const char*)&on: turn on SO_REUSEADDR
 * 5) sizeof(on): size of the option
 */
on = 1;
if(setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR,
(const char *) &on, sizeof(on))==-1){
    perror("setsockopt(...,SO_REUSEADDR,...)");
}

/*
 * when connection is closed, there is a need to linger to ensure all data is transmitted, so turn this on also
 */

struct linger linger = { 0 };

linger.l_onoff = 1; /*enabling lingering*/
linger.l_linger = 30;/*! linger 30 seconds when the connection is closed*/

/*
 * set an option for the socket
 * 1) serverSocket: specifies which server socket to set options for
 * 2) SOL_SOCKET: the option will be set on the socket level, not other levels, i.e. TCP level
 * 3) SO_LINGER: when connection is closed, should the socket linger for a while?
 */
4) (const char*) &linger: turn on lingering
5) sizeof(linger): size of the option

if(setsockopt(serverSocket, SOL_SOCKET, SO_LINGER,(const char *)
&linger, sizeof(linger))==-1){
    perror("setsockopt(...,SO_LINGER,...)\n");
}

hostPtr = hostGetByName("localhost");
if (hostPtr==NULL){
    perror("gethostbyname()\n");
    return(ERROR);
}

/* zero out serverName*/
memset(&serverName, 0, sizeof(serverName));
/* set the ip address*/
serverName.sin_addr.s_addr = htonl (INADDR_ANY);
serverName.sin_len = (u_char) sizeof(serverName);

/* bind a name to a socket*/
if(bind(serverSocket, (struct sockaddr *) &serverName,
    sizeof(serverName))==-1){
    perror("bind()\n");
    return(ERROR);
}

/* BACK_LOG defines the max length of pending */
/* connection may grow to */
if(listen(serverSocket, BACK_LOG)==-1){
    perror("listen()\n");
    return(ERROR);
}

printf("Listening at port %d ... \n", port);
int i = 1;
FOREVER{
    client_struct clientReq;
    int clientLength = sizeof(clientReq.clientName);
    /* zero out the clientName*/
    memset(&clientReq.clientName, 0,
        sizeof(clientReq.clientName));
/* accept an incoming request to service*/
clientReq.slaveSocket = accept(serverSocket,
    (struct sockaddr *) &clientReq.clientName,
    (socklen_t*)&clientLength);

printf ("INCOMING CLIENT MESSAGE %d\n", i);
i++;

    if(write(clientReq.slaveSocket, (void*)MESSAGE,
       sizeof(MESSAGE))==-1){
        perror("Writing to client: ");
    }
    close(clientReq.slaveSocket);

    if (nRead == ERROR)                 /* error from read() */
        perror ("read");
    close(clientReq.slaveSocket);
}

return (OK);

}

dsQuadratic* calcRoots(dsQuadratic* info){
   double a = info->a;
   double b = info->b;
   double c = info->c;

   double disc;

   if(a == 0){
      info->aisZero = 1;
   }
   disc = b*b - 4*a*c;
   if(disc < 0){
      info->discisNeg = 1;
   }
   if(a!=0 && disc!=0){
      info->root1 = (-b + sqrt(disc))/(2*a);
      info->root2 = (-b - sqrt(disc))/(2*a);
   }
   return info;
}