CIS–4020 Lab
Introduction to QNX

Last Revised: December 2, 2009

Introduction

QNX is a commercial micro-kernel based, real time operating system targeting the embedded market. It can also be used as a desktop system, although the support for it in that use is not as extensive as for systems such as Linux or Windows. In this lab you will experiment with QNX in general and with the QNX development tools specifically. Note that as academic users, we have full access to QNX's development suite at no charge.

See the QNX web site at http://www.qnx.com for more information.

QNX Overview

Start QNX in your virtual environment and explore the system a little. Note especially the following items.

1. The QNX terminal gives you a Unix-style shell prompt. You can use this shell to issue ordinary Unix commands and navigate an ordinary Unix file system. Since QNX supports the POSIX API, many Unix tools have been compiled for it.

2. QNX comes with a version of Firefox preinstalled.

3. The default QNX install does not come with any SSH tools. However, there is a project on sourceforge for porting a number of open source tools to QNX. I downloaded and preinstalled a (relatively old) version of SSH for QNX from that project.

4. Although there are no manual pages installed, the QNX help system (accessible via the help viewer tool) is fairly complete. It contains not only reference information but also some good tutorial and overview documents.

5. The QNX development environment is, essentially, Eclipse with the C Development Tools plug-in preinstalled (QNX has added some customizations and branding). There is more on using Eclipse in the next section.

Development Environment

The development environment runs on Windows and cross compiles to the QNX target. If you are programming directly on the QNX system, you will need to instead use a plain text editor and the command line compiler.

The development environment is based on Eclipse, a powerful IDE framework that can be used for many different kinds of development. In this lab you will be using one of the more mature plug-ins: the C Development Tools (CDT). Eclipse by itself is not a compiler. Instead the CDT relies on the services of a

conventional compiler, in this case `gcc`, but presents a user interface that is much more convenient than an ordinary text editor.

The CDT assume your project is built with a makefile and will launch the make utility when asked to do a build. There are two ways this makefile can be created. The CDT can edit the makefile for you so that all you have to do is specify which files are in your project, etc. This option is called a "managed makefile project." Alternatively the CDT can let you edit your own makefile, giving you the ability to use the full power of the make program without any restrictions imposed by the environment. This option is called a "standard makefile project."

For this lab I recommend using standard makefile projects. The managed projects are less suitable for the simple projects we are using, and seem to just add needless complications for our purposes.

Eclipse by default keeps all of its projects under a workspace directory. You can define an "external" project that is kept elsewhere, but for this lab using the workspace directory is probably best. For one thing you can make a tar ball of this directory and copy elsewhere to backup not only your work but also all of Eclipse's settings (which are also stored under the workspace directory as well).

Note also that, unlike some IDEs, Eclipse gives you access to all of your defined projects simultaneously. This can seem confusing at first, but it is often convenient if you want to refer to another project or share files with another project.

Proceed as follows:

1. Load Eclipse and define a standard makefile project named "Hello." Create a new file in this project named "Makefile" and enter the following text.

   ```
   all:   hello

   hello: hello.o
           gcc -g -o hello hello.o

   hello.o: hello.c
           gcc -g -Wall -c hello.c
   ```

   It is necessary for your makefile to define a target "all." Eclipse will build this target by default when it is told to build your project. This allows you to easily build multiple programs at once if necessary. Be sure to indent lines with the TAB key. This is a make requirement.

2. Create `hello.c` and enter the following classic program.

   ```
   #include <stdio.h>

   int main( void )
   {
     printf( "Hello, World!\n" );
     return 0;
   }
   ```

   Build and run this program. *TODO: It is necessary to describe how to use `qconn` to upload the executable to the QNX target system from the Windows development environment. It might also be desirable to talk about how to debug the program via `qconn`.*

3. Create a new project named "Message-Test." Copy the file `qnx-msgtest.c` into the workspace folder for that project (this is the easiest way to add the existing file to the project). Build and test the program.

4. Create a new project named "Pulse-Test." Copy the files `qnx-pulse-server.c` and `qnx-pulse-client.c` into that project. Note that these are two separate programs; adjust your makefile accordingly. Build and test the programs.

1 Qnet Networking

Look up how to start Qnet in the help. Do what is necessary to make that happen. Change the network

configuration of your virtual machine to use direct access to the ethernet interface in your system (instead of, for example, the NAT configuration). Restart your machine and see if you can see the other systems in the lab in your `/net` directory. Demonstrate the pulse client and server working over the network (that is, use your client to send a pulse to a server on a different machine).

Report

There is no report for this lab.