CIS–4020 Lab
QNX Serial Device Driver

Last Revised: November 2, 2008

Introduction

In this lab you will write a QNX device driver for the serial port in your machine. Because QNX is based on a microkernel, your driver can be written as an ordinary user-mode process. This simplifies its construction considerably as compared to the case with drivers for monolithic kernels like Linux. In addition to interacting with the hardware you will also need to provide support for applications that wish to read the port using the standard POSIX functions. To do this you will need to use the QNX resource manager framework.

See the QNX web site at http://www.qnx.com for more information about QNX and about the specific techniques and functions you will need for this lab (QNX has the system documentation on-line). You can also read the QNX documentation using the help version in your QNX session.

Serial Driver: Part 1

Please refer to the datasheet on the 16550 UART for detailed technical information about the serial port hardware in PC class machines. The COM1 port is located at a base address of 0x3F8 and IRQ an value of 4. The COM2 port is located at a base address of 0x2F8 and an IRQ value of 3. You can hard code these values into your program. You only need to support one serial port; you can choose which one.

Also, in the interest of time, you only need to support reading characters from the port. Extending your driver to allow writing as well would obviously be highly desirable in the real world. However, doing so is not as trivial as you might think, and it doesn't add enough educational value to this lab to be worth doing here.

The sample program `qnx-interrupt.c`, taken largely from the QNX documentation, demonstrates the basics of interacting with a hardware device that generates interrupts. You will absolutely want to enable the use of interrupts on the device; polling the hardware is unacceptable in a multitasking system. In the hardware control thread configure the port to use 9600 baud, no parity, eight data bits, and one stop bit. In general it is better to accept this information from the command line and allow the user to specify the serial parameters, but for now you can hard code these parameters into your program.

Your first program should simply print the characters that it receives. To test your program follow the steps below.

1. Terminate the QNX serial port driver. This is straight forward because the driver is a user-mode process.
2. Restart the QNX serial port driver on only one serial port—the port you are *not* using for your driver. Be sure the QNX driver is using the same line parameters (baud rate, etc) as your driver and be sure that it is not using flow control. You will need to review the documentation for the

QNX driver to find out precisely what options you should use.

3. Connect the two serial ports on your machine together using a null modem.
4. Copy a text file to the name representing the QNX driver and observe what characters your driver receives.

Initially you can ignore flow control. If you find there are problems doing that (lost characters) you should implement RTS/CTS flow control in your driver (don't forget to restart the QNX driver with RTS/CTS flow control enabled and be sure the null modem has RTS and CTS crossed over).

Serial Driver: Part 2

Integrate the material in the sample program `qnx-manager.c` into your program. This material provides a skeleton for a simple resource manager that supports reading and writing to a resource by multiple processes[1]. You can leave the write handling alone for now, but you should implement support for the read message. Note that the resource manager code executes in a different thread from the hardware control thread. Initially pass only a single character at a time to the reader regardless of how many characters are requested. You will need to use POSIX thread synchronization primitives (condition variable) to coordinate the threads.

You should now be able to use the standard POSIX `cat` utility to read the name associated with your driver and extract characters from the serial port.

Serial Driver: Part 3

Modify the producer-consumer code you developed for an earlier lab to implement a generous buffer between the hardware control thread and the resource manager thread. Unlike in the earlier lab the consumer in this lab will want to remove more than one item at a time. Modify your earlier code accordingly. If you have implemented flow control, you will want to trigger changes in the flow control lines based on the state of the buffer. For example, hold off the remote side of the connection when the buffer is 3/4 full and restart the remote side of the connection when the buffer drains to 1/4 full.

[1] It might make more sense to allow only a single process at a time to read the port, but that involves extra work with handling open and close messages.

Report

Write a report that describes your program and the test results you obtained. If you did not complete all the steps above, comment on how far you got and on what you would do next to continue the project if you had more time. Turn in your report and a commented listing of your software. As always lab partners can turn in the same listing but should write separate reports.