CDEV-NT: PORTING THE CONTROL DEVICE INTERFACE TO WINDOWS NT

W. Akers, TJNAF; J. Chen, TJNAF; C. Watson, TJNAF

Abstract

In recent years the rapid increase in processing power of the personal computer has made it a significant competitor to high-end workstations for accelerator control and experimental physics applications. When the decreasing price of the PC and the availability of inexpensive commercial software is also considered, NT becomes a very attractive alternative to traditional UNIX systems. In order to simplify the integration of Personal Computers into our operating environment, we have ported the Control Device (CDEV) Interface to Windows NT. By supporting CDEV on this platform, we can provide routine access to our existing control system. Additionally, CDEV allows us to create an interface from our UNIX workstations to Windows NT applications (such as databases) that are significantly less expensive on the PC. This paper details the pitfalls that we encountered during the software migration and will provide a direct comparison between the performance of CDEV applications on UNIX and NT. Particular attention is paid to network performance, which represents most of the overhead of this transition.

1 MOTIVATIONS

1.1 A Common Interface

One of our primary goals at Jefferson Lab has been to develop and use a consistent programming interface whenever possible. CDEV is one of the products of this effort. The CDEV core provides a standard interface that can be wrapped around any control system device making it accessible through a standard API. CDEV extensions provide a set of network interface classes that simplify the development of client/server applications to extend the control system.

As Windows NT machines become more prominent in the control room, concerns have been raised that they might fragment our software development efforts and create unnecessary distinctions between programmers who are developing similar products on different platforms. The extension of CDEV to Windows NT is aimed at reducing this problem by providing a common interface to the control system on both platforms.

1.2 Portable Applications

In addition to making the transition between platforms easier for software developers, another goal is to make the software as portable as possible. Much of the incompatibility between platforms exists in the internal representation of data and the techniques used to move the data between machines. CDEV provides an API that conceals communications details from the application, making it possible to write client/server programs that can be compiled and run on either platform without modification.

2 THE DEVELOPMENT ENVIRONMENT

2.1 Compiler/Linker Selection

One of the first considerations in porting CDEV to Windows NT was to determine which compiler should be used. We examined the venerable GNU C++ compiler, which offered the benefit of having command line parameters that would be consistent between architectures. Several factors dissuaded us from this decision.

When porting CDEV to different UNIX architectures we have traditionally used the native compiler for that platform and have provided the GNU compilation as an alternative. Additionally, the cost of compilers and good quality development tools for Windows NT is relatively low, making the use of GNU tools atypical on this platform. For these reasons, we opted to use the Visual C++ environment for software development.

2.2 Selecting the Build Environment

The selection of the build environment was a more difficult decision than the selection of the compiler. GNU's gmake is commonly used on most UNIX platforms and has been the standard for building the CDEV distribution from the first release. However, building and installing gmake under Windows NT proved to be an annoyance and led us to consider using Microsoft's build utilities.

When performing a build under Windows NT, a developer has the option of using the Visual C++'s system of project files or using Microsoft's nmake. We found that the project files used to perform a build in Visual C++ had many built in restrictions that limited our ability to use our existing directory hierarchy. Many of these "default options" could not be readily overridden, and we abandoned the project files in favor of performing the build with Microsoft's nmake.

The syntax of commands used by nmake is very different from that used by gmake. These differences required us to create a separate makefile for Windows NT. A consolidated Makefile is still present for all of the UNIX platforms, and a new Nmakefile has been added to perform the build for Windows.

3 PITFALLS OF PORTING

3.1 *Dynamic Link Libraries vs. UNIX Shared Libraries*

CDEV is designed to associate a device/message pair with a control system object and then dynamically load a library (the cdevService) that acts as the interface between CDEV and the control system. In UNIX these are called shared objects, in Windows NT they are called Dynamic Link Libraries or DLLs. The most difficult part of porting from UNIX to Windows NT was in discovering the vaguely documented differences between UNIX shared objects and Windows NT's DLLs.

Our first and most significant discovery was that DLLs and Windows applications do not share file descriptors. This made it impossible to directly pass file pointers that were created in the application into a CDEV library function for I/O processing. One solution to this problem was to convert any file descriptor to an osfHandle prior to passing it to the DLL and then have the DLL convert it to a file descriptor before using it. Besides being an inelegant solution, this approach produced source code that was bafflingly complex and non-portable.

After further discussions with Microsoft, we learned that we could link our applications and libraries with the shared version of the C standard libraries. This would provide file and memory operations to both the applications and CDEV libraries from a common source, allowing file descriptors and certain memory blocks to be exchanged seamlessly. While this approach will fail if the application and library are linked with different versions of the C standard library, it provided the most efficient and portable solution to this problem.

3.2 *Networking*

The socket API in Windows NT is remarkably similar to that used in UNIX and much of the code was reused without modification. Many of the incompatibilities that existed were caused by function name differences between the two platforms. To create a map between function names in Windows NT and similar functions on UNIX, a single file, cdevPlatforms.h, was added to the CDEV Generic Server distribution. This file also maps error codes returned by socket operations in Windows NT to the enumerated values that are commonly used in UNIX.

The use of pipes in CDEV posed a more difficult problem. In the UNIX environment, a pipe is functionally the same as a file descriptor or a socket. In Windows NT, however, a pipe is a software entity created by the application that emulates a file descriptor. Because of this, pipes that are generated by Windows NT cannot be passed to the *select* system call. We addressed this problem by writing a new *pipe* function that creates two UDP sockets and connects them to one another. These two file descriptors are then returned to the caller as a socket pair.

The *select* system call proved to have other inconsistencies that had to be addressed. When the user tells CDEV to pend for a specific time interval, the cdevSystem object will collect all of the file descriptors from the underlying services and will *select* on them for the specified interval. In UNIX, if no file descriptors are returned by the services, the *select* function will sleep until the time interval expires. Conversely, the Windows NT implementation of *select* is designed to fail and return immediately if no file descriptors are specified. In order to make the behavior consistent between the two platforms, we added a cdevSelect function that detects this condition on Windows NT and explicitly calls *sleep*.

3.3 *Portable Data Interchange*

All of the data interchange in CDEV is performed using Sun's External Data Representation (XDR). XDR is used to encode all data prior to transmission and to decode data that is received. Because XDR libraries are not provided with Windows NT, we have included Sun's External Data Representation source code with our distribution. These modules are conditionally compiled when performing a build on Windows NT.

4 PERFORMANCE BENCHMARKS

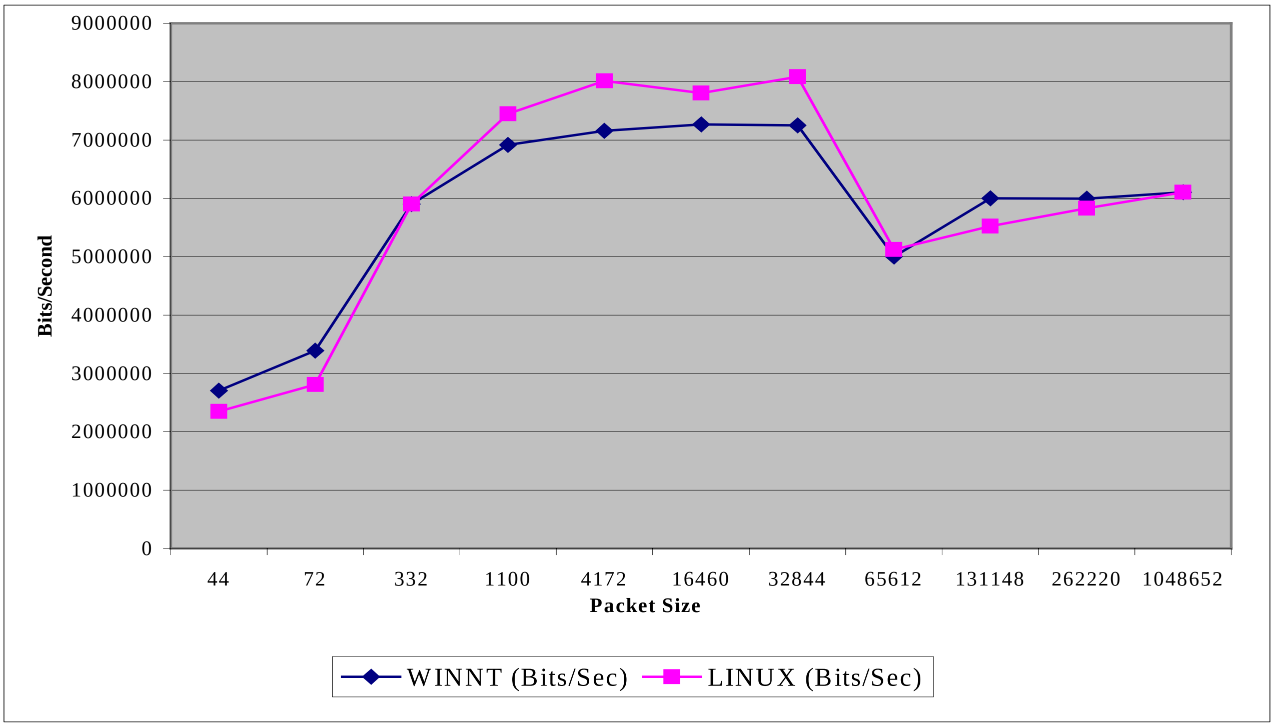
4.1 *Platforms*

In order to make the comparison between the Window and Unix versions of CDEV as fair as possible, we decided to perform our benchmarks on the same two systems. The client system is a 133 MHz IBM ThinkPad and the server is a 450 MHz DELL Dimension. Both systems are configurable to run either Windows NT or Red Hat Linux and are connected using 10 Mbit Ethernet.

4.2 *Performance Test Application*

The test application measures the performance of communications between a CDEV Generic Server and Client running on separate hosts. The application used for this test is provided with the CDEV distribution in the /extensions/cdevGenericServer/tests/Performance directory.

Because most CDEV services are developed using the client/server framework, our test focuses on throughput between a client and server. The test begins by transmitting very small (44 byte) packets between the client and server, and calculating the time required to complete a round trip. The size of each packet is gradually increased until the packet size is one megabyte.



As illustrated by the chart, the performance between the same systems running Linux and Windows NT versions of CDEV is consistent. For packets whose size is at the very high and very low end of the scale, Windows NT outperforms Linux slightly. However, Linux shows significantly better performance in the transmission of mid-range packets.

The dip that is seen when transmitting packets that are 64 kilobytes is related to the buffering scheme that is used by the CDEV Generic Server. One goal of the design is to ensure that a server is not adversely impacted by a misbehaving or slow client. This is done by buffering small packets and automatically forcing flushes when the data transmission size reaches a certain threshold. Future releases of the CDEV Generic Server will use a non-event driven scheme that will result in more consistent performance at higher packet sizes.

5 CONCLUSIONS

The performance difference between CDEV applications compiled on Windows NT and Linux is negligible and should not impact the decision of which platform to use. The greatest benefit provided by Windows NT is the reasonable price and availability of good quality commercial software. The developer can use CDEV to collect information from the control system and then plug this information into products such as Microsoft Excel or Access for analysis and reporting. Using the CDEV interface, developers will be able to write macros for these commercially available spreadsheets and databases that allow them to collect and analyze data in real-time.

6 FUTURE DIRECTIONS

Jefferson Lab in association with CERN is developing an extended CDEV interface that is written entirely in Java. CDEV Java currently accesses the control system through the CDEV Gateway and native Java method calls, and provides mechanisms for attaching to JDBC compliant databases directly.

CDEV extensions, such as the CDEV Generic Server and CDEV Gateway, are continuing to be refined to improve performance and reliability. Upcoming development on these products will attempt to improve network throughput and connection management by replacing the current select-driven, socket management model with a thread oriented approach.

Version 1.7 of the CDEV distribution can be obtained from the CDEV web page at http:\\www.jlab.org\cdev.

REFERENCES

[1] ‘An Object-Oriented Class Library for Developing Device Control Applications’, J. Chen, W. Akers, G. Heyes, D. Wu, C. Watson, ICALEPCS 1995 Proceedings.

[2] ‘The CDEV Generic Server’, W. Akers, J. Chen, C. Watson, S. Witherspoon, J. Van Zeijts, B. Bowling, ICALEPCS 1997 Proceedings.

[3] ‘A Portable Accelerator Control Toolkit’, C. Watson, Particle Accelerator Conference 1997 Proceedings.