OPENNT™: UNIX® Application Portability to Windows NT™ via an Alternative Environment Subsystem

Stephen R. Walli
Softway Systems, Inc.
185 Berry Street, Suite 5514,
San Francisco, CA 94107
stephe@opennt.com

0. Introduction

This paper was originally written for and presented at the USENIX Windows NT Workshop, Seattle, Washington, August 1997. It has been updated to reflect the current architecture and experience with OPENNT.

1. The Problem

Walli's First Law of Applications Portability: Every useful application outlives the platform on which it was developed and deployed.

Application source code portability is one of the cornerstones of most open systems definitions. The intention is that if an application is written to a particular model of source-code portability, it can port relatively easily to any platform that supports the portability model. This model is often based on source code portability standards such as the ISO/IEEE family of POSIX standards [1,2] and ISO/ANSI C[3], and specifications that include these standards such as the Open Group's Single UNIX Specification[4].

The strength of this model is that the investment in the application's development is not lost in re-deployment to new architectures. Re-writing the application to new platforms every five years or so is not an option. There are already too many new applications to be written in the queue without adding the burden of costly re-writes every few years to address newer faster hardware platforms.

Currently businesses are faced with the decision of moving to Microsoft's Windows NT operating system. Windows NT provides a robust environment to solve many business application problems through a host of software development tools, as well as a platform to run the many Win32-based applications that businesses use today. It does this in a price competitive manner with respect to the hardware platforms on which it runs. The problem becomes protecting the huge investment in applications development over the past decade or more in UNIX applications. How does one leverage and protect the existing application base while moving to Windows NT?

2. Alternatives

There are several ways to move existing applications to Windows NT. These range from the expense of a complete re-write of the application to some form of application port. We will briefly look at the pros and cons of the following:

- a complete re-write of the application to the Win32 environment subsystem
- the UNIX emulation library approach to porting the application
- the common library strategy for porting applications
- the Microsoft POSIX subsystem
- the OPENNT subsystem

2.1 A Brief Introduction to the Windows NT Architecture

Before discussing the alternatives, one must understand a little of the overall Windows NT architecture[5]. The Windows NT operating system was structured such that there is a small kernel (not unlike the design of Mach) that is written such that much of the OS is architecture-neutral with the architecture-specific functionality being provided through the Hardware Abstraction Layer (HAL).

Above the kernel sits subsystems that provide functionality to applications. These subsystems are loosely grouped into functional subsystems (e.g. the Security subsystem, the I/O subsystem) that provide basic functionality, and environment subsystems (e.g. Win32 subsystem, the Microsoft POSIX subsystem, the OS/2 subsystem) that present a programmatic and user interface to developers and users. The primary environment subsystem supported by Microsoft, and regarded as the prescribed way to develop applications on Windows NT is the Win32 subsystem. The Win32 subsystem is also responsible for managing the desktop.

Applications programs essentially run as clients of their respective environment subsystem using a variety of techniques to communicate with the subsystem when requesting operating system services. A fast local procedure call (LPC) and shared memory are two of the more common methods of communication. When an application is started, the executable image is inspected to determine which subsystem will provide the application's services.

2.2 Caveat Lector

Prior to discussing the alternatives and the OPENNT subsystem, it is important to note what this paper is not about. Win32 and traditional UNIX applications use different interfaces for accomplishing the same functions (opening files, creating new processes, etc.). These two operating environments also use different philosophies with respect to how applications are structured when looking at such items as the handling of standard I/O streams, the use of file descriptors, consistent return codes, and process creation relationships and control.

This paper does not get into religious or philosophical debate about which OS interface or application architecture may or may not be "better" than the other. Neither does it descend into "marketing" messages about the rationale for providing two different programming and run-time environments on Windows NT. This paper is about solving a very real application programming problem using a unique strategy based on the fundamental design of the Windows NT OS.

2.3 Application Rewrite to Win32

There is a family of simple application programs that are written to ANSI C and its library specifications and will simply re-compile on the Windows NT operating system in the Win32 subsystem space.

Most applications present a greater challenge as they use some form of system resources. Such applications require some amount of rewrite to address the lack of interfaces that directly map the traditional UNIX application interface to which they were written. Some of the re-write may be very straightforward if the use of system services was restricted to simple file operations (*open( ), close( ), read( ), write(*)). If the application depends upon complex signal mechanisms, parent-child relationships (*fork( ), exec(*)), or absolute file semantics (hard links, or the ability to distinguish files by case alone), then the rewrite strategy becomes more expensive as parts of the application must now be re-designed, then re-written.

When one considers a limited number of application programs, this may be an option. If however, one is reviewing a large number of applications, the cost of rewriting can become excessive.

2.4 Win32-based UNIX Emulation Libraries

There are a number of Win32-based libraries emulating the behavior of UNIX system calls and libraries. The most common include:

- UWIN from AT&T Laboratories
- NuTCracker from Datafocus
- Cygwin32 from Cygnus
- Portage from Consensys (Consensys ceased to distribute the product in late 1997.)

Please see Korn [6] for a comparison of these three packages, and for a complete discussion of the UWIN package.

A set of applications may compile cleanly in this environment, but may not run correctly because system calls don't demonstrate the expected semantics. For example, the semantics for *fork()* and *exec( )* cannot be exactly duplicated with respect to the process attributes inherited by the child process. NuTCracker and Portage leave a process "ghost" around, creating a new process, when an *exec()* call is issued. To obviate the need for exact semantics the UWIN project created a *spawn( )* call that replaces the previous two interfaces with a single interface.

File system semantics are another challenge. The Windows NT file system (NTFS) provides almost complete UNIX file system semantics. Group ownership of files, hard links, and filename case sensitivity is all available from NTFS, but are not available through the Win32 subsystem.

All these emulation systems allow for (indeed may require) the use of Win32 interfaces in the application source code. This is not a feature. In a less experienced development operation it can destroy the application's portability, locking it to the hybrid environment. At best it increases the maintenance cost of the source with an additional level of conditional compilation, and the work may not easily integrate into the existing style of UNIX code.

2.5 Common Porting Libraries

Depending upon the application space there may be a set of libraries that have already been created that are used as the portability layer. The only requirement here is that the library exists on Windows NT. The library layer then becomes the porting exercise. This is a special case of the normal situation where the application source code needs to be ported or re-written. It is an application structuring issue.

2.6 The Microsoft POSIX Subsystem

Windows NT supports multiple environment subsystems by design. One of the original subsystems was the POSIX subsystem. The Microsoft POSIX subsystem is an exact implementation of the ISO/IEEE POSIX.1 standard, which includes the ANSI C library by reference.

Microsoft down-played the POSIX subsystem, essentially pointing out that it was delivered for conformance to NIST FIPS Pub 151-2 [7] requirements for government agency procurement. The development environment was not very supportive, and it was poorly documented. As an exact implementation of POSIX.1, the Microsoft POSIX subsystem was somewhat limited in functionality.

The interesting thing to note about early experiments with the Microsoft POSIX subsystem is that all the "big" things (process semantics, signals, and the file system) behave as expected – it is the little things that are surprising. The *ttyname( )* description in POSIX.1 states that the function returns a pointer to a string containing the pathname of the terminal, but returns a NULL pointer if the file descriptor is invalid or if the pathname cannot be determined. There are no error conditions that are detected[1]. A function that always returns a NULL pointer is a sufficient implementation to fulfill the letter of the standard without being particularly useful to an application.

That said, the Microsoft POSIX subsystem passed the breadth of the NIST FIPS 151-2 certification process. In fact, the original OPENNT Commands & Utilities [8] shipped in March 1996 were built on top of a subsystem that was for the most part the original Microsoft POSIX subsystem. That achievement demonstrated that the environment subsystem architecture of Windows NT was flexible and powerful.

2.7 The OPENNT Subsystem

If the Microsoft POSIX subsystem could be built then why not a complete "UNIX" environment subsystem? There is nothing in the Open Group Single UNIX Specification and its attendant UNIX 95 [9] brand that can not be implemented on top of the Windows NT kernel. As you will see, this also applies to the graphic environment of the MIT X11 project.

The goals when developing OPENNT were:

- To provide a complete porting and runtime environment to migrate applications source code developed on traditional UNIX systems directly to Windows NT. This meant going beyond the standards and specifications (e.g. providing X11R5) as well as providing more than one way to access functionality (e.g. determining the next available master pseudo-terminal).
- To provide true semantics for the system interfaces such that application source code would not need to change to account for "not quite UNIX" semantics.
- To ensure any changes made to the application source code should make it more portable (i.e. follow the standards) rather than less portable (i.e. using Windows NT specific constructs.)
- To ensure performance was not effected by an appreciable amount.

- To ensure that the Windows NT operating system's integrity was not compromised in anyway (e.g. security).
- To integrate the OPENNT subsystem cleanly into the Windows NT world such that it was not "isolated" with respect to such things as data access and application or system management.

After initial investigations into Windows95, it was decided to not pursue a solution in this space. There is no way to provide complete or true POSIX/UNIX semantics on Windows95.

The overall goal was to leverage the Windows NT environment subsystem architecture and design to its logical conclusion, taking it from the general purpose application environment that exists solely in the Win32 subsystem space, and using it to provide an application platform for both Win32 and UNIX-based applications.

3. A Short History of OPENNT

In September 1995, Softway Systems, Inc. entered into a long-term agreement with Microsoft to extend the Microsoft POSIX subsystem into a complete traditional UNIX subsystem, capable of branding to the Open Group's UNIX 95 profile brand. Work began immediately to wrap the POSIX subsystem in a shell and utilities environment (delivered March 1996) that conformed to the POSIX.2 Shell and Utilities Execution Environment. An X11R6 server was added (July 1996), along with a telnet service (August 1996). A software development kit (SDK) was added such that developers could begin porting their own applications (September 1996). At that time, the SDK essentially supported the POSIX.1 interfaces, ISO C library, and approximately 60 historical Berkeley library routines from the 4.4BSD-Lite distribution that were required while developing the first POSIX.2 implementation. It provided the additional tools required to develop applications (the RCS version control suite, *ar*, *cc/c89*) and wrapped the Microsoft Visual C/C++ command-line compiler.

Extending the OPENNT subsystem to map more of the kernel interface and adding libraries and utilities has been ongoing. With the release of OPENNT 2.0 (May 1997) and OPENNT 2.1 (Jan 1998), there is now support for:

- POSIX.1, including a nearly complete general terminal interface
- ISO C standard library
- Berkeley sockets
- System V Interprocess Communications facilities (shared memory, semaphores, message queues).
- Memory mapped files
- System V and Berkeley signal interfaces layered onto the POSIX.1 signal semantics.
- traditional curses (ncurses — also supports colour)
- X11R5 clients libraries (Xlib, Xt, Xaw, etc.) and almost all the X11R5 clients
- pseudo-terminals
- OSF/Motif 1.2.4
- cron service
- gcc/g++/g77 and gdb/truss, as well as the "standard" development tools [g]make, rcs, lex, yacc, nm, ar, strip
- telnetd with full concurrent multi-user support.
- inetd, ftpd, rshd, rlogind, rexec
- tape device support
- Perl 5
- the public domain KornShell with full job control, the Tenex csh and 290+ utilities

4. The OPENNT Architecture

The overall architecture of OPENNT consists of the environment subsystem, its mapping to the Windows NT file system (NTFS), its relationship to functional subsystems (e.g. security), and its compiler environment. Issues of integration with the rest of Windows NT and the Win32 subsystem are discussed in the following section.

4.1 The OPENNT Subsystem

The OPENNT subsystem consists of three parts:

- The subsystem itself (PSXSS.EXE) that functionally re-maps the Windows NT kernel and manages such items as process relationships and signal delivery.
- The terminal session manager (POSIX.EXE) that manages the desktop console window for each OPENNT-based session leader.
- The dynamic link library (PSXDLL.DLL) that handles certain system service requests directly, as well as the communication between the subsystem and OPENNT processes.

When an OPENNT application is started for the first time, the Win32-based Program Manager determines that the application belongs to a subsystem other than the Win32 subsystem and a number of events happen:

- If the OPENNT subsystem is not already running, it is started. The subsystem can be configured to start-up at system boot time, but by default is started for the first OPENNT application.
- A terminal session manager is started to manage console window output for the application. A terminal session manager runs for each session. It is the tty device for an OPENNT application.

At that point, application processes run, communicating with the OPENNT subsystem in the same manner as a Win32 application runs and communicates with the Win32 subsystem. Fast Local Procedure Call (LPC) and shared memory are the two primary mechanisms used for communications between the OPENNT subsystem and its processes.

4.2 The Windows NT File System

NTFS was developed for Windows NT to provide a fully functional file system and address the shortcomings of the DOS FAT file system. It provides:

- Recoverability and redundancy of critical disk structures
- Enhanced security consistent with the Windows NT security model
- Data redundancy capabilities to support disk mirroring and striping
- Support for large volumes
- Unicode-based filenames
- Bad cluster remapping
- POSIX file system semantics, including case sensitivity for filenames, hard links, the file change time stamp, and group ownership

File access is mapped from the process to data on the disk via a layered driver model. The NTFS driver is simply another driver layer under the control of the Windows NT executive I/O manager. The NTFS driver can further layer on various fault tolerant drivers, and ultimately layers onto the actual disk driver.

All system service calls for file access come through the I/O manager, regardless of the environment subsystem of origin, i.e. both the Win32 and OPENNT subsystems share a common view of the NTFS. There is no container file system in which POSIX/UNIX file system semantics are emulated. Files are treated as objects within the Windows NT executive, and are managed by the Object manager with respect to object (file) sharing and protection. The Object manager interfaces with the Security Reference Monitor when checking file access permissions.

While the OPENNT subsystem shares a common view of the files within the NTFS, additional functionality is supported. The OPENNT subsystem is able to create hard-links in the file system, something not available via the Win32 subsystem. Win32 subsystem applications (such as the Explorer or File Manager) see two separate files. Case sensitivity in the filename is supported through the OPENNT subsystem, so both *makefile* and *Makefile* can exist, and removing one will not accidentally remove the other.

Advanced features of NTFS are also available to applications running from the OPENNT subsystem. The audit capabilities to track file access success/failure of open, close, read, and write operations are features of NTFS, and available to OPENNT applications. The ability to use additional permission controls with access control lists is also available. These are all file system capabilities that managed by applications that are outside the typical ported application. Windows NT provides the tools to manipulate the audit and advanced security features of NTFS.

4.3 Security, Privileges, and Permissions

Windows NT has obtained its U.S. Department of Defense Orange Book C2 security certification, supporting all required discretionary access controls. One of the goals of OPENNT was to ensure it worked cleanly with the security model on Windows NT, and did not compromise any of its capabilities.

In general, objects (e.g. files) are protected by access control lists (ACL) that are made up of access control entries (ACE). When a user logs onto Windows NT, authentication is provided via username and password and confirmed through the security subsystem. An access token is associated with the successfully logged on user's process and this is used in all actions with objects to determine what access is appropriate given the objects' ACLs.

Several issues arise with respect to OPENNT. The user and group name space is shared. There is no concept of separate user and group databases. Groups can also own objects. This means that a file created on the Win32 side of the house may be owned by a group, and a long listing (*ls*) from the OPENNT side of the house will show a group name as both the owner and the group of the file. This is a little disconcerting for a UNIX system user the first time it is witnessed. Files created from the OPENNT side of the house assign ownership appropriately and consistently with the POSIX standards.

There are no actual permission bits for a file or directory, but rather ACLs are used to map the permission world of UNIX/POSIX. There is an ACE entry for the file owner, file group, and a group named "Everyone" that align with the appropriate permission fields traditionally associated with file permissions in the UNIX world. The ACL can be displayed via the File Manager. Again, files created from the Win32 side of the house will receive an ACL consistent with the Win32 subsystem rules. A long listing from the OPENNT world must map the ACL to permission bits as best it can. Additional levels of security are possible by adding additional ACEs.

There are no */etc/passwd* or */etc/groups* files. All authentication information is kept in the database accessed by the security subsystem. This is one of the two areas that source code changes specific to OPENNT are required for an application that authenticates users. (The other is handling rooted pathnames that expect /usr and /bin to exist in the file hierarchy.) The idea of generating a "shadow" password file was entertained then quickly dropped. The sheer scope and size of some Windows NT domains and the number of users in those domains made start-up time infeasible, and synchronization issues and the maintenance of security would prove too fragile. We provided scripts that allow a system administrator to generate the domain or local password database, such that daily user administrative jobs can still be run against it.

As there is no password file against which to authenticate a user, OPENNT provides a simple functional interface to handle user authentication, as well as a set of *exec( )* functions to execute an application as a particular user. The OPENNT login program executed by the OPENNT telnetd makes use of these functions to setup the shell as a new user. This replaces the *setuid* mechanism.

Windows NT does support an Administrator user with enhanced privileges, as well as an administrators group. There is, however, no root user with all privileges. This has not caused any insurmountable problems in any of the application porting experiences we have had to date, nor have any of the customer base or product beta-testers complained about this lack of a root user id.

4.4 The Compiler Environment

The OPENNT development environment consists of the GNU compilers (*gcc*/*g++*/*g77*), a set of headers, libraries, and shell script compiler drivers for *cc* and *c89* that wrap around the command-line version of the Microsoft Visual C/C++ compiler (*CL.EXE*) and linker (*LINK.EXE*).

The compiler builds object modules that can be linked regardless of the subsystem with which the application will run. The Microsoft linker knows how to stamp the executable appropriately for the "POSIX" subsystem, such that it is correctly passed off to the subsystem when the Win32 subsystem determines it is a "POSIX" binary at application start-up time.

Programs that run as clients of one environment subsystem cannot make calls to interfaces supported by another environment subsystem, so separate libraries are provided to ensure no dependencies to the Win32 world are referenced within the libraries.

Support for *gdb* and *truss* has been provided by implementing a */proc* environment. The MSVC compilers do not generate sufficient information in the object files to be particular good with *gdb*, so compilation with *gcc*/*g++* is the recommended way to get complete *gdb* functionality.

4.5 Performance Issues

The OPENNT subsystem is a peer environment subsystem to the Win32 subsystem. OPENNT processes communicate with the OPENNT subsystem using the same mechanisms as Win32 subsystem processes use to communicate with the Win32 subsystem.

Comprehensive benchmarks have not been run, but early informal work has been done using the iozone [10] benchmark, netperf [11] benchmark, and some informal programs to compare CPU bound program throughput. The iozone tests and CPU tests were run on a traditional UNIX system as well as both the Win32 and OPENNT subsystems running on identical hardware. The netperf benchmarks were run between comparable Windows NT machines, and Windows NT and a similar Intel machine running a traditional Berkeley-based system. Informal trials indicate:

- CPU bound applications perform the same between the Win32 and OPENNT subsystems.
- CPU bound applications performed better on OPENNT than a mainstream traditional UNIX system.
- OPENNT and a traditional UNIX platform showed comparable disk performance trends with iozone.
- For small block *read( )* and *write( )* operations, Win32 outperforms OPENNT. For large block *read( )* and *write( )* operations, OPENNT outperforms Win32.
- Socket throughput between Windows NT and a traditional Berkeley system on a local network is virtually the same regardless of whether the performance testing programs are running with the OPENNT or Win32 subsystems.

Performance tuning is an ongoing task and there are a number of projects underway to increase the overall performance of the OPENNT subsystem.

5. Integration with the Win32 World

Walli's Second Law of Applications Portability: Useful applications seldom live in a vacuum.

There are a number of stresses on applications that exist on multiple platforms or have a history of being migrated to new platforms.

- Most operating systems provide functionality beyond that defined in the POSIX family of standards and the Open Group specifications. This additional functionality may be as straight forward as providing the X11 GUI, or as complex as MVS, VMS, and Windows NT, where another entire operating environment is present.

- Applications often contain platform specific source code. This can be to leverage platform specific functionality, or to handle functionality provided differently on different platforms.

- Once deployed the need to share data between applications becomes a necessity, and new applications are created in the space between existing applications.

- The use of non-standard or platform specific tools and functions is sometimes necessary to solve the business problem, indeed this may drive the actual platform purchase.

Determining the balance between protecting the application investment and solving the business problem with a platform's unique attributes becomes a challenge.

Windows NT provides a rich environment of tools and functionality, developed on top of the Win32 subsystem. The model for integration between the Win32 and OPENNT worlds happens at a higher level than the application programming interface. Applications source code is ported directly to Windows NT to run with the OPENNT subsystem. This protects the application investment. Integration with the Win32 world can then take place in a number of ways: NTFS, the Desktop, Win32exec, shared memory, pipes and sockets.

From an end-user's point of view, on a single machine environment they simply have a "desktop" full of applications. They don't care which subsystem the application is communicating with for kernel services anymore than they care in which language the programmer wrote the source code.

5.1 NTFS

The Win32 and OPENNT worlds share a common file system. Files created in one world are seen in the other. All the security and auditing features provided by NTFS and managed through the Win32-based administration tools are available to OPENNT ported applications.

5.2 The Desktop

Windows NT presents either the Windows 3.1 or the Windows 95 desktop. This environment easily supports interaction between the OPENNT and Win32 worlds.

An OPENNT subsystem terminal or tty appears as a Win32 console. This means large windows and screen buffers, scroll bars and cut-and-paste are all available at the user interface. An OPENNT console window behaves very much like a local *xterm*. Cut-and-paste between Win32-based applications and OPENNT applications is flawless. For example, text from a Microsoft Word document can be easily cut then pasted into a *vi* session or OPENNT shell.

Icons can be set up to launch OPENNT applications, included X11-based applications and shell scripts.

Win32 GUI applications launched from an OPENNT shell present their own window as would be expected, and the application can either be run in the foreground (where the shell will wait for it to complete) or the background.

5.3 Win32exec

An early ability demanded by users was to execute Win32-based applications from the OPENNT world. A Win32exec ability was added that allows Win32 GUI and command-line applications to be executed from within an OPENNT process.

There are a number of challenges to overcome in this space. A Win32 GUI application is relatively straight forward, in that there is no I/O to be managed between the subsystems. Once control of the process has been passed to the Win32 subsystem, the OPENNT shell can wait or continue as desired. If a Win32 command-line user interface (CUI) application is run, and its output is to be seen in the shell window, as if from the standard output of a “normal” child process rather than one running as a client of another subsystem, a certain amount of handshaking and re-direction needs to be performed. Standard streams need to be mapped to Win32 handles appropriately, such that I/O can be redirected to other processes (Win32 or OPENNT) in a pipeline, or to files.

The ability to execute Win32 applications allows a number of facilities to be instantly accomplished in a manner most consistent with Windows NT, but with a “UNIX” interface.

For example, the *lp* utility becomes a simple shell script wrapper around the Win32 *PRINT.EXE* command. All the functions available on the network printing system are instantly available in a manner most appropriate to the architecture, while providing an interface most appropriate to a traditional UNIX environment. Likewise, *useradd* and *userdel*, traditional UNIX commands to manage users, become simple scripts wrapped around *NET.EXE*. A variation of this solution was used to support the *mt* command for tape device management. Instead of adding a lot of non-standard tape device interfaces to the OPENNT subsystem, the *mt* command was written as a simple Win32-based command-line utility and installed in the *bin* directory. The shell executes *mt* as it would any other command.

5.4 OPENNT Sockets and Winsock

Applications can communicate in client/server fashion using sockets and the TCP/IP protocol. Client/server applications can be built to use either the Win32 GUI for client code, written on Winsock in the Win32 subsystem (run on Windows NT and Windows 95), or maintain the traditional UNIX client code with its X11 or simple character interface written using sockets. Server code is maintained as traditional UNIX code running on the OPENNT subsystem. The long term flexibility of the solution is completely in the developer's hands.

There are a number of examples of the strength and flexibility of this solution in our own product space.

- Early on we ported the Apache web server directly into the OPENNT environment. The Apache server source code plus all the existing Perl, CGI, HTML scripting from the UNIX environment comes directly into the

Windows NT world on the OPENNT subsystem. The Microsoft Internet Explorer and Netscape browser are Win32 "clients" that can then be used to connect to the web server.

- The OPENNT X11R6 server is a Win32 application. All of the OPENNT X11R5 clients are OPENNT subsystem applications.

- The telnet daemon shipped with OPENNT is a direct port of a traditional UNIX telnetd running on the OPENNT subsystem. Any telnet client, local or remote, Win32-based or UNIX-based, can connect to it. (As a further example of mixing and matching in the environment to best express functionality while protecting the application source, the OPENNT subsystem telnetd further runs as a Windows NT service, controlled by either the OPENNT command-line *service* utility or the Win32 GUI service control applet in the Windows NT Control Panel.)

5.5 Shared Memory and Memory Mapped Files

The Win32 subsystem supports the concept of memory mapped files. The Windows NT kernel supports shared memory sections, but these are not directly manifested to the programmer. OPENNT supports SVID shared memory and memory mapped files. A very high bandwidth connection can be created between Win32 and OPENNT processes using memory mapped files. With the next release of OPENNT (June 1998) a developer will be able to Win32 map an OPENNT shared memory section as a file as well.

5.6 Win32 GUIs and UNIX Applications [12]

A number of customers have raised the question of how to create a Win32 (Win95) GUI for a UNIX application ported to Windows NT. This is an evolutionary step beyond simply porting the application directly to Windows NT on the OPENNT subsystem, and the customer has already decided that new work will be undertaken.

The environment subsystem architecture prevents client applications of one subsystem calling DLLs from another subsystem, so the obvious step of simply embedding Win32 GUI calls into the UNIX application source base is unavailable. Modifying the application source to embed Win32 GUI interfaces is a very limiting solution and prevents the redeployment of the application again without substantial work.

Three different methods of wrapping a Win32 GUI on UNIX applications have been used with OPENNT depending upon the nature and domain of the application:

- If the application is (or lends itself to) a client server architecture, then the server side is directly ported to Windows NT on the OPENNT subsystem maintaining the application development investment here, and the client is written to Win32. The client has been written using Visual Basic or MSVC/C++ and Winsock. The client is exportable to any "Windows" system.
- Two different customers have written web-based Java front-ends to the UNIX back-end on OPENNT. This has the added benefit that as long as the developer is careful, they have a completely portable graphic front-end to match their portable UNIX application back-end.
- We have experimented in-house with the Win32-based Tcl/Tk provided free from Sun Microsystems. The Win32 implementation is a complete Tcl/Tk environment but the Tk widget set displays with a Win32 look-and-feel. The UNIX application code runs as a ported OPENNT application. The Tcl/Tk graphic front-end is portable to any UNIX environment supporting Tcl/Tk, including the OPENNT Tcl/Tk environment running with the familiar X11 look-and-feel on Windows NT.

While the first solution adds a new dimension to the applications architecture with respect to the Windows world of both Windows NT and Windows95, the latter two solutions provide graphic front-ends with a broad range of deployment opportunities and flexibility.

5.7 UNIX Daemons and Windows NT Services [13][14]

An early customer demand from OPENNT was a better telnet daemon than was commercially available. To port a traditional telnetd to OPENNT required pseudo terminals be added to the subsystem. Once this was accomplished, the Berkeley telnetd ported directly to OPENNT.

Merely having an OPENNT telnet daemon did not completely fulfill the customer requirement. On Windows NT, services are the functional equivalent to UNIX daemons, and there are a number of tools available (both graphic and command line) for managing Windows NT services.

One can run a traditional daemon as a process in the background, however, OPENNT more properly supports running OPENNT ported daemons as Windows NT services. To the service manager, the daemon appears to be a service. It correctly maps the Windows NT start and stop messages to the appropriate signals the daemon understands.

Once *telnetd* was built, *inetd* was quick to follow and as of Spring 1998, OPENNT supports all the traditional internet daemons (*inetd*, *telnetd*, *ntalkd*, *ftpd*, *tftpd*, *rlogind*, *rshd*, *fingerd*, *rexecd*) configured through a traditional *inetd.conf* file in the */etc* directory.

One of the interesting aspects of supporting the daemons is to ensure they correctly integrate with the Windows NT security model. The use of *setuid()* is restricted in use. For an application like *telnetd*, it is relatively straight forward to modify *login* to correctly exec the shell with the *execasuser* call after fetching the user's password. For remote shell operations where the user does not want to continually use their password, OPENNT supports a *.password* file that makes use of NTFS ACL security to ensure the security of the system. It is recommended that *inetd* is run as a completely unprivileged account to further ensure the security of the environment.

6. Porting Experiences

Literally millions of lines of complex UNIX application code has been ported to Windows NT with OPENNT by a large base of commercial and government organizations. (As of Spring, 1998 there were also 22 independent software vendors shipping products on top of OPENNT.) The following sub-sections discuss porting experiences with various packages of software that the Softway Systems developers have ported over time.

6.1 4.4BSD-Lite

Much of the original utility base for the early OPENNT commands and utilities came straight off the 4.4BSD-Lite distribution. The general flow was to copy the source distribution over to a working directory, use a simple template makefile, and begin simple compile-edit cycles until the utility built. The first thing that always needed to be changed was to change *<sys/param.h>* to *<sys/types.h>* and *<limits.h>* as that is the "standard" way to get the types and limits on a POSIX.1 based system. (A skeletal *<sys/param.h>* has since been added to the SDK.) Before sockets and memory mapped files were implemented, any reference to them needed to be stepped around. These functions and macros have since been "turned on" again. Symbolic link code was also avoided. Once linked, testing and conformance work could begin.

6.2 GNU Source

GNU source was used for a number of utilities in the original packages (*RCS*, *diffutils*, *binutils*). The challenge here has always been *configure* scripts. Originally, the MSVC command-line compiler always outputs the name of the module being compiled which confuses *configure* scripts terribly (and there is no way to turn this output off). *Configure* scripts also make liberal use of argument order to the compiler that while common practice is not "standard". The early c89/cc script complained about this.

Configure scripts worked much better once *gcc*/*g++* was ported. In porting this compiler environment, it was setup such that *ld* actually generates proper Windows NT portable executable format (PEF) binaries. *Gmake* has also been ported so configure scripts that rely on it have fewer problems.

Configure aside, the source code itself often just builds. Before *configure* support was added, the fastest way to port a GNU tool to OPENNT was to toss away the configure script, copy the makefile template and configuration header template, and quickly hand-tune them turning on anything that said POSIX or ANSI C. A fast inspection was often enough to get it right. The source code typically built at that point. For example, the only changes required in the 17000+ lines of *gawk* was to change *<varargs.h>* to *<stdarg.h>* in two places.

Configure still has problems in some areas. There are some scripts (e.g. *bash*) where headers appear to be scanned for prototypes which misses functions defined as macros, and thus *configure* doesn't. It took on the order of two hours to port *bash* (Gnu's Born Again Shell). The primary problems were stepping around the GNU *malloc* environment in the build, and catching and fixing a circular macro definition associated with signal support.

6.3 Perl5

The Perl5 distribution was one of the early tools built and made available off the Tool Warehouse section of the OPENNT web page. The distribution is 78,796 lines of code. Running the configure scripts was the only way to determine how to best build Perl5 because of the number of options and permutations available.

The configure script was modified as follows:

- Correct compilation argument order problems (mentioned above).
- Point to the location of the OPENNT header files in *$OPENNT_ROOT/usr/include*.

The configure script produces a set of scripts that further generate the makefiles and configuration-based header files. While the overall configure script could not be run from start to finish due to problems in the environment, it ran well enough to produce the individual sub-scripts which ran correctly, producing a set of headers and makefiles.

The only source changes made stepped around some non-portable use of the user database fields *pw_gecos* and *pw_passwd*. Perl5 passes all the test cases in the test suite shipped with the distribution, with the single exception of the setuid test.

6.4 Apache

Apache, the public domain web server, is 45,726 lines of code. It was ported as an experiment early in the alpha test cycle of OPENNT 2.0.

Initially,

- symbolic links, *setsid( )* and *setpgrp( )* were stepped around.
- Calls to *mktemp( )* were changed to use *getenv( )* of *$TMPDIR* instead of hard-coding path names.
- Uses of *chown( )* were avoided as they violated the *chown( )* functionality mandated by POSIX.1 in association with the standard option *POSIX_CHOWN_RESTRICTED*.
- A *crypt( )* routine (taken from 4.4BSD-Lite) was required.

The following stanza (reformatted for publication) was added to the configuration header for OPENNT.

```
#elif defined(OPENNT)
#define S_ISLNK(m) (0)
#define  bzero(a,b) memset(a,0,b)
#define
     USE_FCNTL_SERIALIZED_ACCEPT
#undef   HAS_GMTOFF
#define  NO_KILLPG
#define  NO_SETSID
#define  JMP_BUF sigjmp_buf
#include <sys/time.h>
#define  getwd(d)
     getcwd(d,MAX_STRING_LEN)
#define  SIGURG SIGUSR1
```

This stanza was about the same size as any other system specific stanza, and has since been updated to handle the new functionality provided since the alpha test version of OPENNT 2.0.

6.5 xv

The *xv* utility is the popular X11-based tool for browsing and manipulating graphic images (e.g. GIF and JPEG). It contains a full file browser as part of the utility. The distribution is approximately 83,600 lines of code. A few trivial source changes were required to:

- step around non-portable use of *mknod( )*
- avoid a use of *endpwent( )*
- add a #define to properly use *strerror( )*

xmkmf was used to generate the appropriate Makefile and *xv* was made.

7. Summary

It has become too expensive to continually rewrite applications to move them from system to system, and source code portability is an important tool to protect existing applications investments. Our experience clearly demonstrates that the Windows NT architecture of alternative environment subsystems provides a way to accomplish this. OPENNT provides the facilities of a traditional UNIX system on Windows NT, such that existing applications developed on traditional UNIX systems can be directly brought to Windows NT, rebuilt, and deployed. Using the environment subsystem architecture of Windows NT, a peer environment to the Win32 world exists, and is integrated to that world in a manner most logical to both.

Up-to-date information about OPENNT can be found at: http://www.OpenNT.com

8. References

1. ISO/IEC 9945-1:1990, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language], IEEE Standards, NJ, ISBN 1-55937-061-0
2. ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities, IEEE Standards, NJ, ISBN 1-55937-255-9
3. ISO/IEC 9899:1990, Programming Languages—C,
4. X/Open CAE Specification:
   - System Interfaces and Headers, Issue 4, Release 2
   - Release 2 Commands and Utilities, Issue 4,
   - System Interface Definitions, Issue 4, Release 2

   X/Open Company Ltd., 1994
5. Custer, Helen, Inside Windows NT, 1993, Microsoft Press, Redmond WA (ISBN 1-55615-481-X)
6. Korn, David, *Porting UNIX to Windows NT*, Proceedings of the USENIX 1997 Annual Technical Conference, pp. 43-57, 1997
7. National Institute of Standards and Technology, Federal Information Processing Standards Publication 151-2, *Portable Operating System Interface (POSIX) - System Application Program Interface [C Language*], 12 May, 1993.
8. OPENNT Commands & Utilities, Release 1.0, Softway Systems Inc., March 1996.
9. Walli, Stephen R., Go Solo: How to Implement and Go Solo with the Single UNIX Specification, Prentice Hall, Englewood Cliffs, NJ, 1995
10. The iozone benchmark available from this site includes source code and documentation for several I/O benchmarks, 9 May, 1997 http://www.cs.umbc.edu/~elm/Ftp/iobenchmarks
11. netperf-1.7.1 Netperf is a benchmark for measuring networking performance. It focuses on bulk data transfer and request/response performance using TCP or UDP and the Berkeley Sockets interface. It is maintained and supported by the IND Networking: 19 March 1997 http://hpux.dsi.unimi.it/hppd/hpux/Networking/Admin/netperf-1.7.1/
12. Walli, Stephen R., *Win32 Tcl/Tk GUIs on UNIX Apps on Windows NT*, ;login Special Issue on Windows NT, November 1997.
13. Ruddock, Rodney and Walli, Stephen R., *inetd and telnetd on Windows NT with OPENNT*, ;login Special Issue on Windows NT, November 1997.
14. McMullen, John, and Ruddock, Rodney *Technical Note #9: inetd and the Daemon Package*, Softway Systems, Inc., 12 Feb, 1998.

9. Trademarks

OPENNT is a trademark of Softway Systems, Inc. Windows NT is a trademark of Microsoft Corporation. UNIX is a registered trademark of The Open Group. All other trademarks belong to their respective holders.