Solaris* To Windows* NT Porting Guide

Contents

List of Tables

Introduction

This guide provides information about porting C/C++ applications from the Solaris* operating system to Windows* NT.

This guide is for software engineers, application developers, and system managers who are considering moving software applications from Solaris to Windows NT. It is assumed that you have software development experience and are familiar with the C and C++ programming languages as well as Windows NT and the Solaris operating systems.

Much of the information in this document is available on the web: This document pulls together the information into one location and references links to the Internet.

What is Covered

This guide covers the following areas.

- Porting options and porting libraries
- Backgrounds on porting applications natively using Windows and Solaris tools, including the following topics:
  - Shell basics
  - Berkeley sockets
  - COM Components
  - Fork
  - Data types
  - Graphical model
  - Making
  - Compiling
  - Linking
  - The ISERAN Project

What is Not Covered

This guide does not cover the following topics:

- Instructions for software packages or editors, such as Microsoft Visual Studio* and "vi".
- Instructions for executable creation, such as makefile and Microsoft Visual Studio.

Porting Options

Several options exist for migrating applications from Solaris to Windows NT:

- Run the Solaris applications on Windows NT using the POSIX subsystem
- Use UNIX* porting libraries
- Porting applications from Solaris to Win32 natively

Using the Windows NT POSIX Subsystem

The first option is typically not recommended. The Windows NT POSIX subsystem only supports POSIX 1003.1, which was the only POSIX version standardized when Windows NT was created. Since then, little demand exists for extending the subsystem, primarily due to most applications having been converted to Win32. Also, the 1003.1 system is of limited interest for fully featured applications, because it does not include many capabilities (such as those in 1003.2, network support, and so on). Finally, full-featured applications that run under the Windows NT POSIX subsystem do not have access to Windows NT features available to Win32 applications, such as memory-mapped files, networking, and graphics. Applications such as VI, LS, and GREP are the main targets for the Windows NT POSIX subsystem.

Using UNIX Porting Libraries

Various ISVs support a Solaris API on top of Windows, allowing for multiple platforms with a minimal amount of source divergence. According to DataFocus, "the fastest migration path from (Solaris) to Windows is direct to Win32. … Microsoft wants your (Solaris) applications to run natively on Win32." As noted in the MSDN "Port from UNIX to Win32" paper, "(t)his is an option for some applications. The advantage of these porting libraries is that they minimize the initial porting effort. The main disadvantage, for a competitive software product, is that a native Win32 port of an application will generally be faster and will inevitably have more functionality. It can be awkward for the application to step outside of its (Solaris) shell if it needs to make Win32 calls to get more power from Windows NT."

The following libraries are commercially available.

DataFocus' MKS NuTCracker

http://www.datafocus.com/products/nutc/

"A Powerful (Solaris) and Windows Co-Existence Solution, MKS NuTCRACKER Professional is a complete UNIX development, interoperability, and runtime environment on Windows."

DataFocus' MKS Toolkit

http://www.datafocus.com/products/tk

MKS Toolkit is a comprehensive solution for Solaris to Windows compatibility and Windows scripting. It offers many familiar Solaris system and file utilities, advanced NT security utilities, Windows specific graphical applications and some new utilities.

Interix

http://www.interix.com

Microsoft acquired Softway Systems, Inc. to help ensure customers have reliable interoperability tools between Windows and Solaris. The Interix technologies provide Solaris customers with a migration path to Windows by allowing them to run their Solaris applications on Windows NT while migrating their code to the Win32 API.

*Virtually UN*X*

http://www.itribe.net/virtunix/contributors.html

Virtually UN*X carries ported Solaris applications for Windows NT including NcFTP, an FTP client for Windows 95 and NT. Virtually UN*X's slogan is *"Virtually UN*X is dedicated to helping bring the power of UN*X tools to Windows 95."*

Cygnus Cygwin

http://sourceware.cygnus.com/cygwin/docs.html

Another Win32 porting layer for Solaris applications that is compatible with all Win32 hosts (at the time of this writing, these include Microsoft's Windows NT/95/98 systems). It is a dynamically-linked library (DLL) that provides a large subset of the system calls in common Solaris implementations. The current release includes all POSIX.1/90 calls except for `setuid` and `mkfifo`, all ANSI C standard calls and many common BSD and SVR4 services (including Berkeley sockets).

The Cygwin tools are ports of the popular GNU development tools for Windows NT, 95, and 98. The Cygwin library provides the Solaris system calls and environment. With the tools installed, Win32 console or GUI applications can make use of the standard Microsoft Win32 API and/or the Cygwin API. As a result, it is possible to more easily port many Solaris programs without the need for extensive changes to the source code. This includes configuring and building most of the available GNU software (including the packages included with the Cygwin development tools themselves). Even if the development tools are not needed, the package provides many standard Solaris utilities. Use them from the bash shell (provided) or from the standard Windows command shell.

ATT UWIN

http://www.research.att.com/sw/tools/uwin/

The UWIN package provides a mechanism by which to build and to run Solaris applications on Windows NT, Windows 98, and Windows 95 with few, if any, changes necessary. The UWIN package contains the following three elements:

- Libraries that provide the Solaris Application Programming Interface (API)
- Include files and development tools such as `cc`, `yacc`, `lex`, and `make`.
- Korn Shell and over 200 utilities such as `ls`, `sed`, `cp`, `stty` etc.

The library functions are implemented as functions exported in a DLL (POSIX.DLL). Programs linked with POSIX.DLL run under the WIN32 subsystem instead of the POSIX subsystem. Thus programs can make Solaris library calls or any other WIN32 call as required. A cc command is provided to compile and link programs for UWIN on Windows NT. The cc command calls one of the Microsoft Visual C/C++ 2.X compiler, the Visual C/C++ 4.X compiler, the Visual C/C++ 5.0 compiler, the Visual C/C++ 6.0 compiler, or the Microsoft Tools C compiler to perform the actual compilation and linking. The GNU compiler and development tools are also available for download with UWIN.

Porting Applications Natively

As mentioned above, applications that are ported natively are apt to perform better since one less layer of complexity exists, i.e., native applications are able to work directly with their operating system. So, the recommended route is to port applications natively from Solaris to Windows NT.

WIN32

The Microsoft Win32 application-programming interface (API) provides building blocks used by applications written for Microsoft Windows NT.

The output model for **stdio** does not need to be changed: The Win32 console APIs mimic the **stdio** model and versions of cursors exist that use the Win32 console APIs. Further, a one-to-one mapping exists between C UNIX APIs and Win32 APIs: **open** to **CreateFile**, **read** to **ReadFile**, **write** to **WriteFile**, **ioctl** to **DeviceIOControl**, **close** to **CloseFile**, and so on.

Basic Solaris applications, including many CGI applications, should port easily to Visual C++ running on Windows NT. Functions like **open**, **fopen**, **read**, **write** and others are available in the Visual C++ run-time library.

Microsoft developed Active Directory Service Interfaces (ADSI) to make it easier for developers to write directory-enabled applications using high-level tools such as the Microsoft Visual Basic* programming system, Java*, C, or the Visual C++ development system. This frees developers from having to worry about underlying differences between the different namespaces.

The Win32 API is extensive. Table 1 lists helpful links about the Win32 API.

Table 1. Win32 API Topic Links

Topic (link)	URL
Base Services	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/buildapp/win32api_4mgj.htm
Common Control Library	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/buildapp/win32api_1a7t.htm
Graphics Device Interface	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/buildapp/win32api_9xut.htm
Network Services	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/buildapp/win32api_2o4z.htm
User Interface	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/buildapp/win32api_5rtx.htm
Windows NT Access Control	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/buildapp/win32api_5rtx.htm
Windows Shell	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/buildapp/win32api_4fos.htm
Windows System Information	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/buildapp/win32api_7s8e.htm

Sun* Solaris*

Solaris offers many kinds of "interfaces", such as the programming interface, elements of the user interface, protocols, and rules about naming and the locations of objects in the file system. The programming interface has two major parts: one seen by developers of applications, which is the API, and one seen by developers of system components, such as device drivers and platform support modules, which is the SPI (system programming interface).

Each programming interface to Solaris is also "visible" to the developer at two levels: source level and binary. The acronyms API and SPI indicate the source level-programming interface to the system. The terms Application Binary Interface (ABI) and System Binary Interface (SBI) indicate the binary interfaces corresponding to the respective source level programming interfaces.

For full details on the Sun Solaris API, see the System Interface Guide at the following URL:

http://docs.sun.com/ab2/coll.45.13/SS/@Ab2TocView?Ab2Lang=C&Ab2Enc=iso-8859-1

The programming environment for the C++ compiler, on Sun Solaris, consists of a number of text editing, program development and debugging tools.

Text Editing Tools

There are several text editors available.

- **vi**--The major text editor for source programs is **vi** (pronounced "vee-eye"), the visual display editor. **vi** has considerable power because it offers the capabilities of both a line and a screen editor. **It** also provides several commands specifically for editing programs.
- **textedit**--The **textedit** editor and other editors are available, including **ed** and **ex**.
- **emacs**--For the **emacs** editor, and other editors not from Sun, see the Sun document, Catalyst, a Catalog of Third-Party Software and Hardware.

Program Development Tools

The following is a brief description of the development tools available. See the Programming Utilities[1] and Libraries[2] manual and the Managing the Toolset[3] manual for details.

- **lex**--Generates programs used in simple lexical analysis of text, solves problems by recognizing different strings of characters.
- **yacc**--Takes a description of a syntax and generates a C function to parse the input stream according to that syntax.
- **prof**--Produces an execution profile of the modules in a program.
- **make**--Automatically maintains, updates, and regenerates related programs and files.
- System V make--Describes a version of **make** that is compatible with older versions of the tool.
- **sccs**--Controls access to shared files and keeps a history of changes made to a project.
- **m4**--Processes the macro language.
- **error**--Generates a source code listing with compiler diagnostics printed before the source line that causes the error.
- **gprof**--Profiles by procedure.
- **sbrowser** -- This is a utility with window, icon, mouse, and pointer interface, that functions as a source code and call graph browser, finds occurrences of any symbol. It can find them in all source files, including header files.
- **tcov**--Profiles by statement.
- **vgrind**--Formats program sources.
- Analyzer—Tunes program performance, including memory allocation.
- FileMerge--Merges source files and coordinating source code changes with other developers.
- MakeTool--Builds programs and browses makefiles.
- Manager—Manages and coordinates other programming tools

Debugging

The debugging tools are:

- **dbx**--An interactive symbolic debugger that understands C++ programs.

- **debugger**--A window, icon, mouse, and pointer (WIMP) interface to **dbx**.
- Thread Analyzer—A multi-threaded code analyzer. Available with SPARCworks/ iMPact.

Shell Basics

http://msdn.microsoft.com/library/psdk/shellcc/shell/Shell_basics/Shell_Basics.htm

The shell organizes the Microsoft Windows user interface (UI) objects into a hierarchical structure called the *namespace*. Users interact with the namespace through the shell's graphical UI (GUI) or through an application. Applications interact with the namespace through the shell's application programming interface (API).

The Microsoft Windows shell API consists of a collection of functions, Component Object Model (COM) interfaces, and COM objects. The shell API can be used anywhere in an application, with one important exception: Like all COM-based services, the shell API should not be used within a DLL's DllMain[4] entry point function. Doing so can cause unpredictable behavior.

To use the Microsoft Windows shell API effectively in an application, first read the Shell Namespace[5] section. This document describes the structure of the namespace and how namespace objects are identified.

Berkeley Sockets

Berkeley socket-based applications need few changes to work as Win32 applications. The Windows Sockets interface was designed for portability with BSD sockets. A limited number of instances do exist where Windows Sockets diverted from strict adherence to the Berkeley conventions. For complete details, see the Microsoft MSDN WinSock[6] overview.

In general, a new, unsigned data type, **SOCKET**, exists in Windows since Windows Sockets applications cannot assume that socket descriptors are equivalent to file descriptors as they are in Solaris. Further, since the **SOCKET** type is unsigned, compiling existing source code from a Solaris environment may lead to compiler warnings about signed/unsigned data type mismatches.

COM Components

An issue arises when platforms have different byte ordering. The issue is transparent to any developer using a high-level language. However, when the high-level language is compiled, it is forced into the byte-ordering specification of the platform. As a result, when data is saved on one platform, it may not be easily read on another platform.

Solaris DCOM uses an implementation of distributed computing environment (DCE) remote procedure call (RPC). RPC is equipped for this issue and handles the conversion automatically. There is no need to specify anything in the interface definition or the programming language.

Fortunately, Windows NT supports DCE-compliant RPC, so RPC-based applications are easily usable.

Fork

One of the largest areas of difference is in the process model. Solaris has **fork**; Win32 does not. Depending on the use of **fork** and the code base, Win32 has two APIs that can be used: **CreateProcess** and **CreateThread**. A Solaris application that forks multiple copies of itself can be reworked in Win32 to have either multiple processes or a single process with multiple threads. If multiple processes are used, there are multiple methods of IPC that can be used to communicate between the processes (and perhaps to update the code and data of the new process to be like the parent, if the functionality that **fork** provides is needed). See Creating a New Interprocess (IPC) Message Class[7].

Data Types

The data types supported by the Microsoft Win32 API are used to define function return values, function and message parameters, and structure members. They define the size and meaning of these elements.

See Microsoft Win32 API Data Types[8]. This document contains a table that contains the following types: character, integer, Boolean, pointer, and handle. The character, integer, and Boolean types are common to most C compilers. Most of the pointer-type names begin with a prefix of **P** or **LP**. Handles

refer to a resource that has been loaded into memory.

Graphical Model

Windows and Solaris graphical models are very different. Solaris uses the X Window System GUI, while Windows uses GDI. No simple mapping of the X API to the GDI API exists. However, OpenGL† support is available for migrating Solaris OpenGL-based applications. And X clients and X servers for Windows exist. See Graphics Device Interface for information on GDI.

Making

It is possible to open an external makefile in the Microsoft development environment. A project is created with one file in it, the makefile that is imported, but the project will build and run within the development environment, assuming it is 32-bit compatible. However, to take advantage of wizards and integrated debugging features, it is suggested to create a new project in the development environment, add the existing files to it, set the compile and link settings to match the old makefile, then build and save the up-to-date project. For steps on importing an external makefile see Port a Project Made with (MAKE) to the Development Environment[9].

Compiling

CL.EXE is a 32-bit tool that controls the Microsoft C and C++ compilers and linker. The compilers produce Common Object File Format (COFF) object (.OBJ) files. The linker produces executable (.EXE) files or dynamic-link libraries (DLLs). To compile without linking, use /c on the command line or click the Compile command on the Build menu. To preprocess, use the /P option.

The Sun Solaris C/C++ compiler and linker are invoked with “cc”, and the compiler preprocessor is invoked with “ccfe”.

Components of cc

- ccfe--Performs preprocessing and compilation
- iropt--SPARC, Optimizes for execution time
- cg--SPARC, Performs code generation when optimization is specified
- cg386, cgpa, cgppc--Prepares intermediate code for code generation
- codegen--Performs optimizations and code generation
- fbe, gas--Generates .o file from .s assembly file

Linking

CL.EXE is a 32-bit tool that controls the Microsoft C and C++ compilers and linker. CL automatically invokes the linker after compiling unless the /c option is used. CL passes to the linker the names of .OBJ files created during compiling and the names of any other files specified on the command line.

Options can be set inside or outside of the development environment. Use the /link option to specify linker options on the CL command line. Options that follow the /link option override those in the LINK environment variable. LINK first processes options specified in the LINK environment variable, followed by options in the order they are specified on the command line and in command files. If an option is repeated with different arguments, the last one processed takes precedence.

The Sun Solaris C/C++ compiler and linker is invoked with “cc”, the stand-alone linker is invoked with “ld”.

Components of cc

- tdb_link--Performs template instantiation of out-of-date templates and invokes the linker
- ld--Performs link editing
- cdlink--Performs post-link operations to handle static constructors and destructors

Mapping of Compile and Link Options

Table 2 lists all of the Solaris compile and link options and maps them to the Win32 compile and link options. Note that the Win32 link options are only referred in this table, and that the descriptions exist in Table 3.

Table 2. Mapping Compile and Link Options

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
-a	Prepares object code for coverage analysis using tcov		
-bsdmalloc	Uses the faster malloc from the libbsdmalloc.a		
		/C	Preserves comments during preprocessing
-B[dynamic|static]	Specifies whether library bindings are “dynamic”(shared) or “static” (not shared). This option and argument are passed to the linker, “ld”	see /LD	
-c	Directs the cc driver compile without linking	/c	Compiles without linking
-cg[89|92]	SPARC macros for archives, chips, and cache		
+d	Prevents expansion of inline functions. Also turned on with –g	/Ob	Controls inline expansion
-dalign	SPARC, Generates double-word load and store instructions		
-d[y|n]	-dy Specifies dynamic linking, -dn specifies static linking. This option and argument are passed to the linker, “ld”	see /LD	
-D*name*[=*def*]	Defines macro symbols to the preprocessor	/D*name*[= | **#** [{*string* | *number*}]]	Defines constants and macros
-dryrun	Does not execute the commands constructed by the compilation driver, only shows them		
-E	Run only the preprocessor on C++ files and send the result to standard output	/E	Copies preprocessor output to standard output
		/EH{s|a}[c][-]	Specifies the model of exception handling
		/EP	Copies preprocessor output to standard output
+e[0|1]	+e0 suppresses the generation of virtual tables, +e1 creates virtual tables for all defined classes with virtual functions		
-fast	Selects a combination of compilation options for optimum execution speed	/Ox	Uses maximum optimization (/Ob1gity /Gs)
-fnofma	Avoids producing fused multiply-add and fused multiply-subtract instructions	see /Op	

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
-fnonstd	Causes nonstandard initialization of floating-point arithmetic hardware	see /Op	
-fns	Turns on the SPARC non-standard floating-point (fp) mode		
-fround=*r*	Sets the IEEE rounding mode		
-fsimple	Optimizes mathematically equivalent expressions		
-fstore	Converts the value of a fp expression to the type on the left hand side	see /Op	
-ftrap=*t*	Sets the IEEE 754 trapping mode		
		/F *number*	Sets stack size
		/FA[*c*|*s*]	Creates a listing file
		/Fa	Sets listing file name
		/Fd*filename*	Renames program database file
		/FD	Generate file dependencies
See –o		/Fe*filename*	Renames the executable file
		/Fi*filename*	Preprocesses the specified include file
-flags	Displays a brief description of each compiler option	/HELP	Lists the compiler options
		/Fm[*filename*]	Creates a map file
		/Fo*filename*	Creates an object file
		/Fp*filename*	Specifies a precompiled header file name
		/FR[*filename*] /Fr[*filename*]	Generate browser files
-386		/G3	Optimizes code to favor the 386 processor. Phased out in Visual C++ 5.0, the compiler will ignore this option.
-486		/G4	Optimizes code to favor the 486 processor. Phased out in Visual C++ 5.0, the compiler will ignore this option.
-pentium		/G5	Optimizes code to favor the Pentium® processor
		/G6	Optimizes code to favor the Pentium Pro processor.

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
		/GB	Optimizes code to favor the Pentium processor. Blends optimizations for the 80386 (/G3), 80486 (/G4), Pentium (/G5), and Pentium Pro (/G6) options.
		/GA	Optimizes code for Windows application
		/GD	Optimizes code for Windows DLL
-g	Instructs the compiler and linker to prepare the file or program for debugging	See compile options: /LDd and /Zi See link options for: /DEBUG	
-g0	Instructs the compiler and linker to prepare the file or program for debugging, but not to disable inlining	See compile options: /LDd and /Zi See link options for: /DEBUG	
-G	Specifies shared library bindings. This option is passed to the linker, “ld”	see /LD	
		/Gd	Uses the **__cdecl** calling convention
		/Ge	Activates stack probes
		/GF /Gf	Enable string pooling
		/Gh	Calls hook function, **__penter**
		/Gi	Enables incremental compilation
		/Gm	Enables minimal rebuild
		/GR	Enables run-time type information (RTTI)
		/Gr	Uses the **__fastcall** calling convention
		/Gs*size*	Controls stack probes
		/GT	Supports fiber safety for data allocated using static thread-local storage.
		/GX[–]	Enables synchronous exception handling
		/Gy	Enables function-level linking
		/GZ	Catch release-build errors in debug build
		/Gz	Uses the **__stdcall** calling convention

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
-help	Displays a brief description of each compiler option	/HELP	Lists the compiler options
-h*name*	Names a shared dynamic library and provides a way to have versions of them	See link options: /DLL, /VERSION	
-H	Prints the path name of each include file		
		/H*number*	Restricts the length of external (public) names
-I	Causes linker to ignore any LD_LIBRARY_PATH setting	See link options: /LIBPATH, /NODEFAULTLIB, /IMPLIB	
-inline=*rlst*	Inlines the routines specified in the *rlst* list		
-I*pathname*	Adds *pathname* to list of directories to search for include files	/I*directory*	Searches a directory for include files
		/J	Changes the default **char** type
-keeptmp	Retains temporary files created during compilation		
-KPIC, -Kpic	Produces position-independent code		
-l*lib*	Specifies additional libraries for linking with object files	See link option: /LIBPATH	
-libmieee	Causes libm to return values in the spirit of IEEE 754		
-libmil	Inlines some library routines for faster execution		
-L*dir*	Adds *dir* to the list of directories to be searched by the linker for libraries that contain object-library routines	See link option: /LIBPATH	
See –B, -G, -dy		/LD	Creates a dynamic-link library.
See -g		/LDd	Creates a debug dynamic-link library
		/link *option*	Passes the specified option to LINK
-migration	Displays contents of the C++ Migration Guide		
-misalign	SPARC, Notifies the compiler when the data in the program is not properly aligned		

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
		/MD	Creates a multithreaded DLL, using MSVCRT.LIB
		/MDd	Creates a debug multithreaded DLL, using MSVCRTD.LIB
		/ML	Creates a single-threaded executable file, using LIBC.LIB
		/MLd	Creates a debug single-threaded executable file, using LIBCD.LIB
-mt	Compiles and links a multithreaded program	/MT	Creates a multithreaded executable file, using LIBCMT.LIB
		/MTd	Creates a debug multithreaded executable file, using LIBCMTD.LIB
-M*mapfile*	Directive to use the specified mapfile. This option is passed to the linker, “ld”	See link option: /MAP	
-native	Generates code for the machine that is performing the compilation	See /Ox	
-nocx	Does not link to –lcx		
-noex	Does not generate code to supports C++ exceptions		
-nofstore	Does not convert the value of a fp expression to the type on the left-hand side of an assignment		
-nolib	Does not link with any default system or library	See linker options: /IMPLIB, /LIBPATH, /NODEFAULTLIB	
-nolibmil	Resets –fast so that it does not include inline templates		
		/nologo	Suppresses display of sign-on banner
-noqueue	If no license, will return without queuing the request to compile		
-norunpath	Does not build the path for shared libraries into the executable		
		/O1	Creates small code
		/O2	Creates fast code
		/Oa	Assumes no aliasing
+d	Prevents expansion of inline functions. Also turned on with –g	/Ob	Controls inline expansion
		/Od	Disables optimization
-o1	Minimal postpass assembly-level optimization		

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
-o2	o1+ Basic local and global optimization	/Og	Uses global optimizations
-o3	o2 and optimizes references and definitions for external variables		
-o4	-03 and does automatic inlining	/Oi	Generates intrinsic functions
		/Op	Improves floating-point consistency
		/Os	Favors small code
		/Ot	Favors fast code
		/Ow	Assumes aliasing across function calls
-o5, See /fast	Highest level of optimization	/Ox	Uses maximum optimization (/Ob1gity /Gs)
		/Oy	Omits frame pointer
		/Qlf	Generates additional debugging information for kernel-mode device drivers.
		/QI0f	Performs Pentium 0x0f erratum fix
		/QIfdiv[–]	Performs Pentium FDIV erratum fix
-o *filename*	Sets the name of the output file or the executable to *filename*	/Fe*filename*	Renames the executable file
+p	Disallows anachronistic constructs		
-p	Prepares object code to collect data for profiling with prof	See link option: /PROFILE	
-pentium		/G5	Optimizes code to favor the Pentium processor
-pg	Prepares object code to collect data for profiling with gprof	See link option: /PROFILE	
-pta	Instantiate a whole template class, creates a ".o" file for each member of a class		
-pti*path*	Includes the path name of the template source		
-pto	Instantiates generated templates and makes them static		
-ptr*database-path*	Specifies directory of primary and secondary build repositories (templates)		
-ptv	Turns on verbose mode	See link option: /VERBOSE	
-P	Writes preprocessor output to a ".i" file	/P	Writes preprocessor output to a file

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
-PIC, -pic	Produces position-independent code		
-qp	Prepares object code to collect data for profiling with prof	See link option: /PROFILE	
[-Qoption|-qoption] *prog opt*	Passes the option to the phase	See link option: /OPT	
[-Qproduce|qproduce] *sourcetype*	Produce source code of the type *sourcetype*		
-readme	Displays the contents of the readme file		
-R	Merges the data segment with the text segment		
-R *pathname*	Used to create a list of directories to be used during library searches	See link option: /LIBPATH	
-s	Remove all symbol information from output executable files. This option is passed to the linker, “ld”		
-sb	Generate extra symbol table information into a database		
-sbfast	Runs only the ccfe phase to generate the extra symbol table information into a database		
-S	Compile and output an assembly source file		
-temp=*dir*	Sets the name of the directory for temporary files		
-time	Report execution times for various compilation passes		
		/Tc*filename*	
		/TC	Specifies a C source file
		/Tp*filename*	
		/TP	Specifies a C++ source file
-unroll=*n*	Specifies loop unrolling activity dependent upon *n*		
		/u	Removes all predefined macros
-U*name*	Removes a predefined macro	/U*symbol*	Removes a predefined macro
-v	Prints command line for each compilation pass		
-V	Prints names and version numbers of invoked programs		
		/V	Sets the version string
		/vd{0|1}	Suppresses or enables hidden vtordisp class members

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
		/vmb	Uses best base for pointers to members
		/vmg	Uses full generality for pointers to members
		/vmm	Declares multiple inheritance
		/vms	Declares single inheritance
		/vmv	Declares virtual inheritance
+w	Generates additional warnings	/W*level*	Sets warning level
-w	Disables all warnings	/w	Disables all warnings
-x601	Optimizes for PowerPC 601 processor	See link option: /MACHINE	
-x603	Optimizes for PowerPC 603 processor	See link option: /MACHINE	
-x604	Optimizes for PowerPC 604 processor	See link option: /MACHINE	
-xa	Prepares object code for coverage analysis using tcov		
-xar	Creates archive libraries		
-xarch=*a*	Limits the set of instructions the compiler may use		
-xcache=*c*	SPARC Defines cache properties for the optimizer	N/A	
-xcg[89|92]	SPARC macros for archives, chips, and cache	N/A	
-xchip=*c*	SPARC, Specifies the target processor for use by the optimizer	See link option: /MACHINE	
-xF	Enables reordering of functions. If followed by an Analyzer run, a map file is created.	See link option: /ORDER	
-xildoff	SPARC, Turns off the incremental linker	N/A	
-xildon	SPARC, Turns on the incremental linker	N/A	
-xinline=*rlst*	Inlines the routines specified in the *rlst* list		
-xlibmieee	Causes libm to return values in the spirit of IEEE 754		
-xlibmil	Inlines some library routines for faster execution	See /Ob	
-xlibmopt	SPARC, Uses an optimized math routine library	N/A	

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
-xlicinfo	Displays information on the licensing system	N/A	
-xM, -xM1	Outputs makefile dependency information	N/A	
-xMerge	Merges data and text segments	See link option: /MERGE	
-xnolib	Does not link with any default system or library	See link options: /IMPLIB, /LIBPATH, /NODEFAULTLIB	
-xnolibmopt	SPARC, Resets *–fast* and does not use math routine library	N/A	
-xo*level*	Same as –o*level*, above	See corresponding solutions described above for -o*level*	
-xpg	Prepares object code to collect data for profiling with gprof		
-xprofile=[collect|use|tcov]	Collects data for profile or Uses a profile	See link option: /PROFILE	
-xregs=*r*	Specifies register usage for the generated code	See link options: /LARGEADDRESSAWARE, /STACK	
-xs	Disables Auto-Read for dbx		
-xsafe=mem	Allows the compiler to assume no memory-base traps occur. Grants permission for speculative loads		
-xsb	Generate extra symbol table information into a database		
-xsbfast	Runs only the ccfe phase to generate the extra symbol table information into a database		
-xspace	Does no optimization that increases code size		
-xtarget=*t*	Specifies target system for instruction set and optimization	Not specifically supported, see “/G” options	
-xtime	Report execution times for various compilation passes		
-xunroll=*n*	Specifies loop unrolling activity dependent upon *n*		
-xwe	Converts all warnings to errors		
-Xm	Allows use of the character “$” in identifier names	N/A	
		/X	Ignores the standard include directory
		/Yc[*filename*]	Creates a precompiled header file

Solaris Option	Solaris Purpose	Win32 Option	Win32 Purpose
		/Yd	Places complete debugging information in all object files
		/Yu[*filename*]	Uses a precompiled header file during build
		/YX	Automates precompiled header
-Ztha	Prepares code for the Thread Analyzer		
-ztext	Forces fatal error for relocations remain against non-writable allocable sections		
		/Z7	Generates C 7.0–compatible debugging information
		/Za	Disables language extensions
		/Zd	Generates line numbers
		/Ze	Enables language extensions
		/Zg	Generates function prototypes
		/Zi	Generates complete debugging information
		/ZI	Includes debug information in a program database compatible with Edit and Continue
		/Zl	Removes default library name from .OBJ file
		/Zm*number*	Sets the compiler's memory allocation limit
		/Zn	Turns off SBRPACK for .SBR files
		/Zp*n*	Packs structure members
		/Zs	Checks syntax only

Table 3. Win32 Link Options

Win32 Link Option	Action
/ALIGN:*number*	Specifies the alignment of each section
/BASE:{*address* | @*filename*,*key*}	Sets a base address for the program
/COMMENT:["]*comment*["]	Inserts a *comment* string into header
/DEBUG	Creates debugging information
/DEBUGTYPE:CV /DEBUGTYPE:COFF /DEBUGTYPE:BOTH	Creates particular formats of debugging information
/DEF:*filename*	Passes a module-definition (.DEF) file to the linker
/DEFAULTLIB:*library*	Searches specified library when resolving external references

Win32 Link Option	Action
/DELAY	Controls the delayed loading of DLLs
/DELAYLOAD	Causes the delayed loading of the specified DLL
/DLL	Builds a DLL
/DRIVER[:UPONLY]	Creates a Windows NT kernel mode driver
/ENTRY:*function*	Sets the starting address
/EXETYPE:DYNAMIC	Builds a virtual device driver
/EXPORT	Exports a function
/FIXED[:NO]	Creates a program that can be loaded only at its preferred base address
/FORCE[:{MULTIPLE|UNRESOLVED}]	Forces link to complete in spite of unresolved or multiply defined symbols
/GPSIZE:#	Specifies the size of communal variables for MIPS and Alpha platforms
/HEAP:*reserve*[,*commit*]	Sets the size of the heap in bytes
/IMPLIB:*filename*	Overrides the default import library name
/INCLUDE:*symbol*	Forces symbol references
/INCREMENTAL:{YES|NO}	Controls incremental linking
/LARGEADDRESSAWARE	Tells the compiler that the application supports addresses larger than two gigabytes.
/LIBPATH:*path*	Allows the user to override the environmental library path
/LINK50COMPAT	Generates import libraries in Visual C++ Version 5.0 format
/MACHINE:{IX86|ALPHA|ARM|MIPS|MIPSR41XX|PPC|SH3|SH4}	Specifies the target platform
/MAP	Creates a map file
/MAPINFO:{EXPORTS|FIXUPS|LINES}	Includes the specified information in the map file
/MERGE:*from=to*	Combines sections
/NODEFAULTLIB[:*library*]	Ignores all (or specified) default libraries when resolving external references
/NOENTRY	Creates a resource-only DLL
/NOLOGO	Suppresses startup banner
/OPT:{REF|NOREF|ICF[,*iterations*]|NOICF}	Controls LINK optimizations
/ORDER:@*filename*	Places COMDATs into the image in a predetermined order
/OUT:*filename*	Specifies the output file name
/PDB:*filename*	Creates a program database (.PDB) file
/PDBTYPE:{con[solidate]|sept[ypes]}	Specifies where to store the Program Database (PDB) debug type information.
/PROFILE	Enables profiling (creates a mapfile)
/RELEASE	Sets the checksum in the .EXE header
/SECTION:*name*,*attributes*	Overrides the attributes of a section
/STACK:*reserve*[,*commit*]	Sets the size of the stack in bytes
/STUB:*filename*	Attaches an MS-DOS† stub program to a Win32 program

Win32 Link Option	Action
/SUBSYSTEM:{CONSOLE|WINDOWS|NATIVE|POSIX|WINDOWSCE} [,*major*[.*minor*]]	Tells the operating system how to run the .EXE file
/SWAPRUN:{NET|CD}	Tells the operating system to copy the linker output to a swap file before running it
/VERBOSE[:LIB]	Prints linker progress messages
/VERSION:*major*[.*minor*]	Assigns a version number
/VXD	Creates a virtual device driver (VxD)
/WARN[:*level*]	Specifies warning level
/WS:AGGRESSIVE	Aggressively trim process memory

The ISERAN Project

http://www.iseran.com/

"The Iseran Project is comprised of skilled individuals who fund their mountaineering holidays through software development." The project includes an FAQ on Win32[10] programming and porting which contains the following information.

Porting

Porting is not as difficult as it has been made out. However, the Win32 API:

- houses many slightly different APIs -TextOut, ExTextOut and TabbedTextOut and PolyTextOut, for example
- split the NT Executive objects and all other objects in a system: GDI objects, Windows and sockets
- grew to match as the platform quickly evolved
- key strength is legacy support.

Migration Tips

- Don't develop for the Posix subsystem, instead, use the Win32 API with the C/C++ run time libraries.
- NT5's object model is not too different from (Solaris's) 'everything is a file' world. Just use WaitForMultipleObjects() instead of select().
- Message queues are not waitable objects; MsgWaitForMultipleObjects() is a way of including queues.
- Use WSASelect(), since sockets are not waitable in Win95, only in NT and Win98 with Winsock 2.
- When opening files with fopen(), use the "b" argument to indicate binary mode. Otherwise crlf translation will catch you out.
- Do not assume case sensitivity in naming files.
- Forward slashes in path names are usually ok.
- The C++ compiler should support STL, although some patches may be needed.
- There is no fork() and no real equivalent. Somewhere in the NT executive it -and a process hierarchy- are supported, but they don't make it as far as the Win32 subsystem. Threads and CreateProcess() are what are available.
- signal() may not work well even if it is in the run time library.
- When moving from a workstation environment, note that the byte order for x86 CPUs is different: use htons() and htonl() and the inverses when doing socket work.
- No direct equivalent of a **setuid** bit for running programs.

"Windows NT + GNU UN*X=Free" by James Fudge for WinPlanet

www.winplanet.com/winplanet/reports/573/1/

In a discussion between James Fudge and Chris Szurgot of Virtually Un*x, Mr. Szurgot stated that the hardest part of setting up most ported UNIX applications involves:

- "Getting the headers right. Although winsock.h was easy to find, some of the more esoteric headers had to be built from scratch. (FTP, & TELNET info mainly.)
- Screen & Keyboard Input/Ouput. In order to do NcFTP, I had to also port Curses, but normal DOS screen IO was horribly inefficient, so curses had to be ported to use Win32 Console mode function. Fortunately, somebody had already ported Curses to dos, so I just built on that. …
- Removing fork().
- Otherwise, they were relatively simple."

References and Other Related Topics

For more information, refer to the URLs listed in

Table 4. Related Topics

Topic (link)	URL
Port from UNIX† to Win32†	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/devprods/vs6/visualc/vccore/_core_port_from_unix_to_win32.htm
Win32 Topics	http://msdn.microsoft.com/library/devprods/vs6/visualc/vccore/_core_win32_topics.htm
Overview of the Win32 API	http://msdn.microsoft.com/library/psdk/buildapp/win32api_7f8p.htm
Application Integration	http://msdn.microsoft.com/library/backgrnd/html/appint.htm
Build Reliable and Scalable N-tier Applications that Run on Both Windows NT and UNIX --MSJ, December 1998	http://msdn.microsoft.com/library/periodic/period98/ntUnix.htm
From One Code Base to Many Platforms Using Visual C++†	http://msdn.microsoft.com/library/backgrnd/html/msdn_fromone.htm
Microsoft Windows NT and UNIX Interoperability	http://msdn.microsoft.com/library/backgrnd/html/WindowsNT_UNIX_Interop.htm
The Microsoft Windows NT Platform Enterprise Interoperability with UNIX	http://msdn.microsoft.com/library/backgrnd/html/ntsunixinterop.htm
Write portable code for UNIX	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/devprods/vs6/visualc/vccore/_core_write_portable_code_for_unix.htm
Write portable code for Windows	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/devprods/vs6/visualc/vccore/_core_write_portable_code_for_windows.htm

Topic (link)	URL
Building COM Components on UNIX	http://msdn.microsoft.com/library/techart/msdn_unixcom.htm
Deviation from Berkeley Sockets	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/winsock/ovrvw3_9xma.htm
Microsoft Visual C++	http://msdn.microsoft.com/library/devprods/vs6/visualc/vccore/addpfhm.htm
Microsoft Visual C++ Compiling and Linking	http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/devprods/vs6/visualc/vccore/comphm.htm
Microsoft Visual C++ Compiling and Linking Reference	http://msdn.microsoft.com/library/devprods/vs6/visualc/vccore/_core_compiler_reference.htm
Sun Solaris 8 Documentation	http://docs.sun.com/ab2/@TopicBrowse?topic=Sol8_C;Ab2Enc=iso-8859-1;Ab2Lang=C
Sun Solaris C++ Programming Environment	http://docs.sun.com/ab2/@LegacyPageView?toc=SUNWab_32_3%3A%2Fsafedir%2Fspace3%2Fcoll1%2FSUNWspro%2FSPROabcpl%2Ftoc%2FCPPPUG%3A1306;bt=C%2B%2B+4.1+User%27s+Guide;ps=ps%2FSUNWab_32_3%2FCPPPUG%2F07.Programming_Environment&Ab2Lang=C&Ab2Enc=iso-8859-1
Solaris Compiler and Linker documentation	http://docs.sun.com/ab2/@LegacyPageView?toc=SUNWab_32_3%3A%2Fsafedir%2Fspace3%2Fcoll1%2FSUNWspro%2FSPROabcpl%2Ftoc%2FCPPPUG%3A1039;bt=C%2B%2B+4.1+User%27s+Guide;ps=ps%2FSUNWab_32_3%2FCPPPUG%2F02.The_Compiler&Ab2Lang=C&Ab2Enc=iso-8859-1
Sun Solaris Managing the Toolset	http://docs.sun.com/ab2/@LegacyTocView?toc=SUNWab_44_2%3A%2Fsafedir%2Fspace3%2Fcoll1%2FSUNWspro%2FSPROabrm%2Ftoc%2FTOOLSET%3ATOOLSET;bt=Managing+the+Toolset&Ab2Lang=C&Ab2Enc=iso-8859-1
Sun Solaris 2.6 Software Developer Collection Vol 1	http://docs.sun.com/ab2/coll.45.4/@Ab2CollView?subject=driver&Ab2Lang=C&Ab2Enc=iso-8859-1
Sun Solaris 8 Software Developer Collection	http://docs.sun.com/ab2/coll.45.13/@Ab2CollView?subject=driver&Ab2Lang=C&Ab2Enc=iso-8859-1

Footnotes

[1] http://docs.sun.com/ab2/coll.45.4/PROGUTILS/@Ab2TocView?Ab2Lang=C&Ab2Enc=iso-8859-1

[2] http://docs.sun.com/ab2/coll.45.4/LLM/@Ab2TocView?Ab2Lang=C&Ab2Enc=iso-8859-1

[3] http://docs.sun.com/ab2/@LegacyTocView?toc=SUNWab_44_2%3A%2Fsafedir%2Fspace3%2Fcoll1%2FSUNWspro%2FSPROabrm%2Ftoc%2FTOOLSET%3ATOOLSET;bt=Managing+the+Toolset&Ab2Lang=C&Ab2Enc=iso-8859-1

[4] http://msdn.microsoft.com/library/psdk/winbase/dll_8asu.htm

[5] http://msdn.microsoft.com/library/psdk/shellcc/shell/Shell_basics/Namespace.htm

[6] http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/winsock/ovrvw3_9xma.htm

[7] http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/mapi/start_7yk3.htm

[8] http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/psdkref/type_8uk3.htm

[9] http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/devprods/vs6/visualc/vccore/_core_port_a_project_made_with_nmake_to_the_developer_studio_environment.htm

[10] http://www.iseran.com/Win32/FAQ/section0.html