Porting Space to Windows

S. de Graaf

Circuits and Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands

Report EWI-ENS 07-02
Nov 20, 2007

Last revision: Jan 18, 2008.

1. INTRODUCTION

A port of Space to the Windows[1] platform is required, because Tanner EDA[2] wants to sell it.

First, the complete Space software was easy ported to Windows using the CYGWIN[3] environment. But using CYGWIN, you must go accord with the terms of the GNU General Public License (GPL).

Second, we have looked for MinGW and UWIN. UWIN (from AT&T) looks to be a good candidate, but was not easy to install for cross compilation. The MinGW[4] environment was directly more successful installed and used. Thus, we are using the MinGW environment for porting the Space software. The MinGW base runtime package is uncopyrighted and the w32api is free to use (see the MinGW licensing terms on site www.mingw.org).

Note that Tanner is only interested in the Space core tools. The Space extractor (space, space3d) and needed programs (makeboxl, makegln, makemesh, makesize, makedela, makesubres, tecc). Futher more some NELSIS (database) programs (mkpr, rmpr, rmdb, dbcat, dblist, getproc, icdman, ...). The graphic programs are not ported (dali, helios, simeye, spock, view3d, Xspace).

1. Windows is a short name for the Microsoft Windows OS (like Windows 98/ME/XP), the most used Operating System on 32-bits Personal Computers (PCs).
2. Tanner EDA, a division of Tanner Research, Inc. Tanner EDA is a leading provider of PC-based EDA software solutions for the design of ICs. Tanner Tools supports Windows XP and Vista platforms.
3. CYGWIN, GNU + Cygnus + Windows, a Linux-like environment for Windows.
4. MinGW, the Minimalist GNU for Windows environment (cross-compiler tools and libraries).

2. THE MINGW ENVIRONMENT

The MinGW[5] environment is delivered with the following tool set:

ar	archive program, to make object libraries
as	assembler
cpp	C macro preprocessor
dlltool	create files needed to build and use DLLs
g++	GNU C++ cross-compiler
gcc	GNU C cross-compiler
ld	linker, combines objects and libraries
nm	list symbols from object files
objdump	display information from object files
ranlib	generate index to archive (library)
strip	discard symbols from object files
windres	manipulate Windows resources

MinGW uses the runtime libraries distributed with the Windows platform. This is called the Win32 API. Documentation about the Windows32 API can be found in Microsoft's on-line library[6] or some other places (see: http://www.mingw.org/docs.shtml#win32api). Note that the C Run-Time library supports almost all UNIX C functions. Note that the UNIX system functions start with a leading underscore, but MinGW delivers an old names interface library.

5. MinGW, the Minimalist GNU for Windows environment.
6. MSDN (Microsoft Developer Network), http://msdn.microsoft.com

3. DIFFERENCES BETWEEN UNIX AND WINDOWS

3.1 File and Directory Path

UNIX uses the slash character ('/') to separate path components (directory names) leading to the file name. Windows uses the backslash character ('\') for this purpose. Absolute path names start with a drive letter followed by a colon (':') and the backslash. For example:

```
C:\WINDOWS\Temp
```

On UNIX, all file systems are mounted to one root ('/') directory. Thus, absolute path names always start with a slash. On Windows, this is split-up, each drive has its own file system. To retrieve the absolute path on Windows, it is a good idea to use the **getcwd** function.

3.2 Network Drives and UNC

The UNC, short for Universal (or Uniform) Naming Convention, specifies a common syntax to describe the location of a network resource, such as a shared file, directory, or printer. The UNC syntax for Windows systems is as follows:

```
\\ComputerName\SharedFolder\Resource
```

The network resource path can locally be associated with a network drive (for example L:\). For UNC, see http://en.wikipedia.org/wiki/Path_%28computing%29.

3.3 File Extensions

The length and characters of directory and file names are not more so restricted, like in MS-DOS. It is common to use file extensions, a dot followed by 3 chars, to specify the file type. For example:

```
.bat   batch program file
.exe   executable program file
.doc   document file of Word
.lnk   link file (shortcut)
```

It is now-a-days more common to use spaces in names, however UNIX has some trouble with spaces (because uses it as separator).

3.4 Names are Case Insensitive

In Windows, the following paths specify the same directory:

```
C:\WINDOWS\Temp   C:\WINDOWS\TEMP   C:\windows\temp
```

This is a serious problem for our database format we are using. We can not use the following three cell names:

```
oscil  Oscil  OSCIL
```

3.5 File Name Restrictions

Windows does not permit the use of the following characters in names:

```
\  /  :  *  ?  "  <  >  |
```

These characters have a special meaning, for example used as wildcard or used for I/O redirection. Also, control characters are not intended to be used in names.

3.6 File Stat Functions

Windows has a _stat[7] function, which is simular to the UNIX stat function. The _stat function obtains information about the file or directory specified by path and stores it in the _stat structure pointed to by buffer. The _stat structure has almost the same field names as its UNIX counterpart. That does not mean, that all fields are supported.

```
st_dev;     ID of device containing file -> DRIVE NUMBER (0 = 'A')
st_ino;     inode number                 -> NOT SUPPORTED
st_mode;    protection                   -> ONLY IFDIR/IFREG
st_nlink;   number of hard links         -> NOT SUPPORTED
st_uid;     user ID of owner             -> NOT SUPPORTED
st_gid;     group ID of owner            -> NOT SUPPORTED
st_rdev;    device ID (if special file)  -> NOT SUPPORTED
st_size;    total size, in bytes         -> SUPPORTED (32-bit)
st_blksize; blocksize for filesystem I/O -> NOT SUPPORTED
st_blocks;  number of blocks allocated   -> NOT SUPPORTED
st_atime;   time of last access          -> ONLY VALID ON NTFS
st_mtime;   time of last modification    -> SUPPORTED (till 2038)
st_ctime;   time of last status change   -> ONLY VALID ON NTFS
```

In the Windows environment, you cannot identify (symbolic) links with _stat. And Windows does not support the UNIX **lstat** function. Note that symbolic links (shortcuts) in Windows are identified by the .lnk extension. And what i know, Windows does not have hard-linked files. The Windows **_stat64** function supports 64-bit time/size type. Note that in Visual C++ 2005 the _stat function supports 64-bit time type. Note that Windows has also wide-character versions. The Windows C Run-Time library does also not support *user* and *group* ID functions. Note that the **_stat** function sometimes fails to stat the given path, while the **fopen** or **opendir** function can be successful. Note that the Windows distribution does not contain any links.

3.7 File Protection

The Windows platform uses a very simple file protection scheme. It only supports read-only and read-write permissions. Thus, Windows does not support execute and group/others permissions. Maybe the user can have restricted rights, because he/she has

7. C Run-Time library (Visual C++, http://msdn.microsoft.com/en-us/library/abx4dbyh(VS.80).aspx).

no administrator privilege. Note that for directories, Windows does not support any permissions. Thus, the **_mkdir** function does not have the *mode* argument (the second argument is missing). See also the MS-DOS **attrib** command. Files can have special attributes, like the *system* and *hidden* attribute, which makes them read-only.

3.8 The System Function

The **system** function passes *command* to the command-interpreter, which executes the string as an operating-system command. The function refers to the COMSPEC and PATH environment variables that locate the command-interpreter file (the file named CMD.EXE in Windows NT and later). If command is NULL, the function simply checks to see whether the command-interpreter exists. This function is system depended and therefor not portable. Other commands are used on UNIX platforms.

3.9 Line Termination Characters

The Windows platform uses default other line termination characters than UNIX platforms. The CR/LF combination is used. On UNIX platforms only the *newline* character (i.e. LF, linefeed) is used. The C file read/write functions does automatically the translations. Thus, if writing newline chars to a file, you get (default) CR and LF for each newline character. When reading, default, the CR/LF combination is automatically translated back to one newline character. This means, that you cannot easy *diff* files made under UNIX and Windows. You can also not use the **fseek** function on files opened in the (default) text mode. When you does not want the translations, you must use the file binary access mode. To use that mode, see **fopen** function, add the ’b’ for binary after the ’r’ or ’w’.

3.10 Errno Problem

The MinGW compiler uses a macro definition for the global **errno** variable. Therefor it is a problem to use a local variable with the same name.

3.11 Signal Handling Problem

Windows does only support a sub-set of the signals of UNIX platforms. The following signals are not supported:

```
    SIGHUP   SIGQUIT   SIGALRM   SIGCONT   SIGPIPE
```

4. The Windows User Environment

When starting to work on a Windows platform, the user gets its own environment (if set-up). Its own "Desktop" and "My Documents" folder. There is no need for a home directory ala UNIX, when not working in a command line window. When starting CMD, the Windows command-interpreter, you can enter MS-DOS commands. For example:

```
chdir - change (or display) the current directory (or: cd)
copy  - copy file(s)
date  - change/show current system date
dir   - list directory contents
del   - delete file(s) (or: erase)
echo  - turn echo mode on/off or echo a message
exit  - exit the command-interpreter
mkdir - make a new directory (or: md)
more  - list contents of a file (.. lines each time)
path  - set or show the search path
ren   - rename a file or directory (or: rename)
rmdir - remove directory (must be empty) (or: rd)
set   - set or show the environment variables
type  - list contents of a file
```

Enter the **set** command to show the specified environment variables. As can be seen, there is probably no HOME environment variable found. Instead, there are HOMEDRIVE and HOMEPATH. Note that you cannot go to your home directory ala UNIX. To go to the home directory enter:

```
%HOMEDRIVE%
cd %HOMEPATH%
```

Note that you can only change to another drive by giving the drive letter followed by a colon command. You cannot change your drive with the **chdir** (change directory) command. The Windows environment uses also the PATH variable, but the command-interpreter looks first for program files in the current directory before trying the search path. Note that each directory in the environment variable PATH is separated by a semicolon (;). UNIX is using the colon (:) as separator, but that’s not possible for Windows (because the colon is used after a drive letter).

5. Missing Functions

5.1 The fork Function

The **fork** and **vfork** functions are missing. The **spawn** function is used to solve this problem. Note that the **spawn** function looks for spaces in the argv-list. Care must be taken, because argv[0] can contain an absolute executable path. When the path contains spaces, you must put double quotes around argv[0], else it is split. I have chosen to remove the absolute path and to use only the tail of it (the program name part).

5.2 The sbrk Function

The **sbrk** function is used to get information about how many memory is allocated by the program. The problem is solved, by looking to the addresses **malloc** gives back and the last allocated byte size.

5.3 The link Function

This function was sometimes used to rename a file. We fixed the problem by using the **rename** function instead. Note that a file not always can be moved in this way (for example to another file system). In that case a copy and unlink operation must be performed.

5.4 The rand48 Functions

Added the 48-bit random functions to "libstd". Also on the Cygwin platform we had trouble with the random functions. To be sure, that they always work the same way, we can better use our own functions.

5.5 The sleep Function

The UNIX **sleep** function was not found on the Windows platform. We use the **Sleep** function, which argument must be specified in milli seconds.

5.6 The times Functions

Implemented own **times** function in "libstd". Added also **gettimeval** to replace the **gettimeofday** function.

6. Shell Scripts

UNIX depended shell scripts cannot be used for Windows. Some TCL scripts, which are interpreted by the **cacdtcl** program are implemented as a Windows batch file.
For example the **tabs** program uses the following batch file (tabs.bat):

```
@echo off
cacdtcl %ICDPATH%\share\src\space\utilities\tabs\tabs --no-color %*
```

There are also batch files made for:

```
circuit   layout   tabs-verify
```

7. Third Party Libraries

Some third party libraries must be used for some purposes. Very important is, that they can be compiled with the Mingw cross-compiler.

7.1 The libcrypto Library

We use the MD5 and RSA functions of this library.

7.2 The libgmp and libmpfr Library

This multi precision libraries are used in our unigreen library.

7.3 The libpthread Library

I don’t know of we really use this library. Maybe only the include files. Our **libstd** is thinking of threads.

7.4 The libtcl Library

For our **cacdtcl** shell program, we need a TCL 8.3 or 8.4 library.