unDOS / unWin

DOS and Win32 Emulators

Developer's Guide

Version 4.0
October 2010

by
Alan Moore

Micro Digital

Table of Contents

Overview

unDOS is a simple DOS and BIOS API emulation package. It emulates many of the common DOS int 21h and other services and some of the BIOS int 10h video services as well. Similarly, unWin is a simple Win32 API emulation package. It emulates a small set of Win32 functions. These were individual products in the past but now they are included with smx86 releases. The files are in BSP\X86.

unDOS and unWin provide support for DOS- and Win32-based libraries to run in a real-time, multitasking SMX environment, with little or no modification. The first goal was to support the basic functions of the C run-time library. Another goal was to support graphics libraries that were developed to run in protected mode requiring a DOS extender. Each call to the VGA BIOS or DOS had a high overhead as the DOS extender switched from protected mode to real mode and back. It also would have required writing interrupt handlers for both real mode and protected mode. Using unDOS, this was avoided since the needed services were implemented using its own code and SMX services that run in protected mode, so no switching was needed[1]. This cuts down the typical interrupt latency time dramatically and makes for a simpler system to develop and debug. Still another goal was to support legacy user code that was written for DOS so it would run without DOS and even in protected mode with little modification. The DOStap TSR is provided to help assess the DOS dependency of an existing application while it runs, which is especially important when source code was not available for portions of the code, such as 3rd party libraries.

unWin provides limited Win32 API emulation, enough to provide basic support for the C run-time libraries.

Both provide the option of logging unsupported BIOS, DOS, and Win32 calls made by your program. Contact Micro Digital regarding these unsupported functions.

Note: Information for real mode, 16-bit protected mode, and MetaWINDOW has been removed from this manual. Please refer to a pre-v4 revision for this information.

[1] The one exception is the BIOS int 10h video initialization function, which relies on DPMI 300h. Unfortunately, all graphics cards require different initialization, and the only way to avoid writing a driver for each specific card is to rely on the ROM code resident on the card, which runs in real mode. To run this code, the special DPMI 300h function is used to switch to real mode. Note that this is only an initialization call made once at system startup, not during normal system operation.

Summary of unDOS Services

The following services are supported by unDOS. The 32-bit DOS functions use extended registers for their parameters. Since there is no known standard for these, we have done our best to use an intuitively apparent approach to decide how parameters should be passed in 32-bit registers. This is similar to the format used by DOS extenders such as Borland's PowerPack.

Note that file I/O services are mapped onto smxFile, which is an older file system that is no longer supported. In a future release, it will be mapped it onto smxFS.

Interrupt 10h Services (BIOS)

00h	Set Graphics Mode (by switching to real mode using DPMI 300h)
02h	Set Cursor Position
0Fh	Read Current Video State
10:00h	Set Individual Palette Registers (EGA/VGA)
11h	Character generation has been stubbed out; forces graphics packages to use its own fonts.
12:00h	Return Video Information (EGA/VGA)
1A:00	Read Display Combination Code (VGA)

Interrupt 21h Services (DOS)

0Eh	Set Default Drive
19h	Get Default Drive
1Ah	Set DTA address
25h	Set Interrupt Vector
2Ah	Get Date
2Bh	Set Date
2Ch	Get Time
2Dh	Set Time
2Fh	Get DTA address
30h	Get Version Number (6.20)
35h	Get Interrupt Vector
38h	Get Country information (Set not supported)
39h	Create Directory
3Ah	Remove Directory
3Bh	Change Current Directory
3Ch	Create File with Handle (volume label not supported)
3Dh	Open File with Handle
3Eh	Close File with Handle
3Fh	Read File
40h	Write File
41h	Delete File
42h	Move File Pointer
43:00h	Get File Attributes
44:00h	Get Device Data (for files only, no DOS devices supported)
47h	Get Current Directory
48h	Allocate Memory
49h	Free Allocated Memory
4Ah	Set Memory Block Size (stub)
4Ch	End Program
4Eh	Find First File
4Fh	Find Next File
50h	Set PSP Address
51h	Get PSP Address

56h	Rename File
57h	Get/Set File Date and Time
5Bh	Create New File
62h	Get PSP Address
66:01h	Get Global Code Page (Returns US code 437)

Summary of unWin Services

In addition to the unDOS services listed above, unWin adds the following emulation functions.

Win32 API emulation functions supported

The Win32 API is partially supported. Below is a list of the functions which are supported or partially implemented.

General limitations and notes (other limitations are noted beside each function)

1. The file I/O functions support only disk files and not console I/O or pipes.
2. Only single byte character set support is currently provided.
3. A TIB (Thread Information Block) is provided to facilitate emulation of Win32 although not all fields are functional. For more information, see the source code (unwin32b.c).
4. Many additional functions are stubbed. These stubs print an error to the screen or trap in the debugger to alert you if they are called.

CloseHandle	
CreateFileA	security attributes, template files, and handles to directories not supported
DeleteCriticalSection	
DeleteFileA	
EnterCriticalSection	
FileTimeToLocalFileTime	not truly implemented; file time reported is same as file time
FileTimeToSystemTime	
FindClose	
FindFirstFileA	
FindNextFileA	
FlushFileBuffers	
GetACP	returns US code page; can be changed to return other country code page
GetCPInfo	returns US code page; can be changed to return other country code page
GetCurrentDirectoryA	
GetCurrentProcess	returns NULL since SMX does not support processes
GetFileAttributesA	
GetFileType	
GetFullPathNameA	
GetLastError	
GetLocalTime	
GetStartupInfoA	not truly implemented; just returns NULL for file handle info fields, to support Borland RTL
GetStdHandle	user must first use SetStdHandle since no standard handles built in
GetTimeZoneInformation	
GetVersionExA	minimal implementation
GetVolumeInformationA	returns fixed volume name, not actual disk volume name
HeapAlloc	
HeapCreate	
HeapDestroy	
HeapFree	

InitializeCriticalSection	
LCMapStringA	upper and lower case ASCII only
LeaveCriticalSection	
MulDiv	
RaiseException	reports exception condition only
ReadFile	
SetFilePointer	
SetHandleCount	
SetLastError	
SetLocalTime	
SetStdHandle	
SetTimeZoneInformation	
TerminateCurrentProcess	returns NULL since SMX does not support processes
TlsAlloc	
TlsFree	
TlsGetValue	
TlsSetValue	
VirtualQuery	partial support; returns min address of own stack
WriteFile	

Structure of unDOS and unWin

unDOS is composed of several C, assembly, and include files which are organized by function and processor mode. The organization of the files is important since you may need to make specific modifications or additions to unDOS/unWin in order to support your application. The files are grouped into the following categories: RTL support, DOS services (int 21h), VideoBIOS services, and Win32 API functions.

Source Files

RTL support is truly spread throughout all of the files. However, the bulk of the RTL-specific support is to be found in urtl.c. This file also serves as a catch-all file to hold code which does not obviously belong in one of the other files. This includes initialization, exit, and logging functions. Much of the code in this file is customized for various compilers so you will find many conditional compile statements.

DOS service support is broken up into a portable C file, emu.c, and a processor mode specific assembly file dosemu32.asm. It contains the actual interrupt handler for int 21h, the vector table for the DOS subfunctions, and the assembly code to handle the subfunctions. Generally, the assembly code calls a C function in emu.c which will performs the meat of the operation or calls smxFile.

Video BIOS services are emulated by vgabio32.asm. It contains the actual video BIOS interrupt handler for int 10h, the subfunction vector jump table, and the subfunction emulation code. All code for video BIOS emulation is written in assembly language.

Win32 API emulation code is contained in the unwin32_.c files. uwin32a.c is provided with smx and stubs off a number of routines needed so our products link successfully and run. uwin32b.c contains most of the Win32 functions, many of which are supported functions. There are also many stubs in this file. All are written in C code and are intended to replace Windows DLLs.

Some other files are present in the directory that are modified versions of files supplied with the particular compiler's RTL.

Include Files

There are two primary include files provided for the unDOS user. undos.h is a general include file which can be used by any unDOS user. It includes useful prototypes and constants. Much of this is also available in the standard compiler vendor include files such as dos.h and stdio.h. We recommend using undos.h instead, although the others should work just as well.

unwin32.h is a greatly reduced Win32 include file which has all of the prototypes and declarations needed to use the Win32 API functions supported by unWin. We strongly recommend the use of this include file rather than the Windows include files since it is much smaller, and it is specifically designed to be compatible with SMX include files. **SMX include files are not generally compatible with the standard Windows include files.**

Summary

dosemu32.asm	32-bit assembly source interrupt handler for DOS int 21h
emu.c	C source to implement int 21h services.
undos.h	DOS services emulation include file for user.
unwin32.asm	Win32 API import address table, required by Microsoft C++.
unwin32a.c	Win32 API emulation routines provided with smx.
unwin32b.c	Win32 API emulation routines provided with unWin.
unwin32.h	Win32 API emulation include file for user.
urtl.c	C source code specific to compiler vendor RTL support.
vgabio32.asm	32-bit protected mode assembly source, handler for video int 10h.

unDOS and unWin Services by Category

In this section, unDOS/unWin services are broken down by category and discussed in more detail. References are made to C Standard Run Time Library (RTL) functions which are supported by unDOS/unWin. Although the implementation of these functions varies for each compiler vendor, they generally use this same basic set of DOS and Win32 services.

The RTL functions listed here are in no way intended to represent a complete list of functions supported. You will find that much of your compiler's RTL or your 3rd party library is supported by unDOS/unWin.

File I/O

A primary purpose of unDOS/unWin is to emulate the standard DOS and Win32 File I/O functions. This allows the user to open, close, read, write, and find files on disk from an SMX application. As an incidental benefit, the unDOS functions are performed more quickly than DOS which must often negotiate a long chain of interrupt handlers for the simplest of operations.

unDOS/unWin does not currently support the standard devices: stdin, stdout, stderr, stdaux, and stdprn. As a result, console I/O is not supported.

unDOS/unWin map onto smxFile services. Thus, it shares the same limitations and capabilities as smxFile. For example, only binary mode is supported. Text mode is not properly supported so routines depending on text mode such as fgets() and fputs() may not work properly in certain situations. A future release will map onto smxFS.

C library routines

The following ANSI C standard buffered file I/O routines may work under unDOS.

getc()	fputc()	remove()
fclose()	fread()	rename()
feof()	fseek()	rewind()
ferror()	fsetpos()	setbuf()
fflush()	ftell()	setvbuf()
fgetc()	fwrite()	ungetc()
fgetpos()	isatty()	
fopen()	putc()	

The following UNIX-style standard C library unbuffered or low-level file I/O routines may also work under unDOS:

chdir()	fstat()	read()
close()	getcwd()	stat()
creat()	lseek()	unlink()
fileno()	open()	write()

The following routines are not part of any formal standard but may work under unDOS:

_dos_findfirst()	isatty()	setdisk()
_dos_findnext()	mkdir()	
getdisk()	rmdir()	

Many of your compiler's file I/O C library functions will work under unDOS/unWin since they use the basic DOS/Win32 functions that unDOS/unWin provides. Don't be afraid to experiment!

DOS int 21H subfunctions supported

0Eh	Set Default Drive
19h	Get Default Drive
1Ah	Set DTA address
2Fh	Get DTA address
39h	Create Directory
3Ah	Remove Directory
3Bh	Change Current Directory
3Ch	Create File with Handle (volume label not supported)
3Dh	Open File with Handle
3Eh	Close File with Handle
3Fh	Read File
40h	Write File
41h	Delete File
42h	Move File Pointer
43:00h	Get File Attributes
44:00h	Get Device Data (for files only, no DOS devices supported)
47h	Get Current Directory
4Eh	Find First File
4Fh	Find Next File
56h	Rename File
57h	Get/Set File Date and Time
5Bh	Create New File

Win32 API functions

CloseHandle
CreateFileA
DeleteFileA
FindClose
FindFirstFileA
FindNextFileA
FlushFileBuffers
GetCurrentDirectoryA
GetFileAttributesA
GetFileType
GetFullPathNameA
GetStdHandle
GetVolumeInformationA
ReadFile
SetFilePointer
SetHandleCount
SetStdHandle
WriteFile

Date & Time

These routines allow the user to set and retrieve the date and time. The user is able to initialize the date and time which is updated by smx. The time is stored in the smx global variable stime which maintains the current time and date in the standard ANSI/UNIX format: the number of seconds elapsed since 00:00:00 Jan 1, 1980. The user must set the date and time, reading the hardware clock if one is available.

The ANSI C routines time() and clock() have been replaced with simpler functions which directly access corresponding smx global variables.

C library routines

The following ANSI C standard time and date routines may work under unDOS:

asctime()	difftime()	mktime()
clock()	gmtime()	time()
ctime()	localtime()	

The following routines are not part of any formal standard but may work under unDOS:

_dos_getdate()	_dos_setdate()
_dos_gettime()	_dos_settime()

Most of your compiler's time and date C library functions will work under unDOS since they use the basic DOS functions which unDOS provides. The only ones which won't work are the ones which use the BIOS time interrupt such as Borland's biostime().

DOS int 21H subfunctions supported

2Ah	Get Date
2Bh	Set Date
2Ch	Get Time
2Dh	Set Time

Win32API functions

FileTimeToLocalFileTime
FileTimeToSystemTime
GetLocalTime
GetTimeZoneInformation
SetLocalTime
SetTimeZoneInformation

Process Control (PSP and TIB)

Many of the ANSI C standard Process Control functions are supported by unDOS. This includes the various types of application exit routines: exit(), abort(), and the atexit() function stack.. The End Program DOS subfunction (4Ch) is also supported.

Most of the UNIX and OS/2 standard process control routines such as cwait(), wait(), raise(), signal(), exec() and spawn() are not supported by unDOS. Many of these are specific to the way DOS and UNIX handle processes and are not relevant to the SMX multitasking environment. For task control you should use the smx API directly.

The Program Segment Prefix (PSP) is passed by DOS to every executable when it is loaded and executed by DOS. Many DOS-compatible libraries expect to have access to the PSP in order to access the DOS environment variables. unDOS provides a PSP which is currently non-reentrant; i.e. there is only one PSP block which is shared by all tasks accessing unDOS. In the PSP there is provided an environment string in standard DOS format. The environment is initialized in the unDOS source code and the user may modify it according to the needs of their application.

The user may access environment variables by using the standard C routine getenv() or by accessing the PSP via DOS interrupts. The user may also create his own PSP and change the PSP pointer. This could be a means of making unDOS reentrant much as DOS is made reentrant for multitasking operating systems such as OS/2 and Windows.

The standard routine put_env() has been supplied to allow the user to dynamically alter the environment. The routines put_env() and get_env() work together on their own copy of the environment. Changes made to the environment via put_env() will not be reflected in the copy of the environment which can be accessed via the PSP. The environment string pointed to by the PSP is a copy of the original environment string which was compiled into the program.

The other fields of the PSP are not currently supported by unDOS.

C library routines

The following ANSI C standard process control routines have been provided by unDOS to replace the routines in your C library.

abort()	exit()	putenv()
atexit()	getenv()	

DOS int 21H subfunctions supported

4Ch	End Program
50h	Set PSP Address
51h	Get PSP Address
62h	Get PSP Address

Win32 API functions

Win32 does not support the PSP. Instead, Win32 has something called a TIB (Thread Information Block) which is allocated for each thread. unWin treats the entire SMX application as a single thread and provides a single TIB which it uses to emulate Win32's TIB. Not all fields of the TIB are initialized, see unwin32b.c for details.

unWin supports the Win32 critical section construct using smx semaphores. This is used by multi-thread libraries to make their functions reentrant:

DeleteCriticalSection
EnterCriticalSection
InitializeCriticalSection
LeaveCriticalSection

Currently, unWin supports use of the Borland 32-bit multi-thread RTL (CW32MT.LIB) and will soon support the Microsoft 32-bit multi-thread RTL (LIBCMT.LIB) . Linking the multithread library is enabled by uncommenting the "mt" macro in pro.mak.

The advantage of the multi-thread library is that it uses these critical section locks so that it is reentrant. In other words, the multithread RTL has been modified expressly for a preemptive multi-tasking environment such as SMX. There is no need for the application code to test a semaphore prior to making C library calls when this library is used, as is needed for other non-multi-thread libraries.

Interrupt Vector Management

unDOS supports the get and set interrupt vector management functions. unDOS also provides replacements for the commonly used interrupt generation functions int86(), int86x(), and int386(). Currently these functions only handle the interrupt numbers 10h and 21h (video and DOS).

C library routines

The following routines are not part of any formal standard, but they may work under unDOS:

_dos_getvect()	int86()	int386()
_dos_setvect()	int86x()	

Most of your compiler's interrupt vector management C library functions will work under unDOS since they use the basic DOS functions which unDOS provides.

DOS int 21H subfunctions supported

25h	Set Interrupt Vector
35h	Get Interrupt Vector

Win32 API functions

It is not possible to interact with interrupt vectors through the standard Win32 API.

Internationalization

unDOS provides basic support for the internationalization features of DOS. The information is initialized to United States settings and can be changed in the source code to whatever country the user desires. The more advanced features which allow changes to the fonts used and such are not supported at this time. We recommend use of graphics interfaces such as MGL to provide fonts for other languages.

DOS int 21H subfunctions supported

38h	Get Country information (Set not supported)
66:01h	Get Global Code Page (Returns US code 437)

Memory Management

unDOS/unWin provides basic DOS like memory management using the smx heap. Set Memory Block Size is generally used by startup code to free up memory for the heap by cutting down the amount of memory allocated for the executable program. This does not apply to the SMX environment so this function has been stubbed off; it returns an error code to the caller.

smx provides a generous supply of heap management routines including many of the ANSI C standard routines. Most applications or libraries will access the heap through these routines and will never use the DOS subfunctions.

C library routines

The following ANSI C standard memory management routines are supported by smx and unDOS:

calloc()	malloc()
free()	realloc()

Most of your compiler's memory management C library functions work under smx/unDOS.

DOS int 21H subfunctions supported

48h	Allocate Memory
49h	Free Allocated Memory
4Ah	Set Memory Block Size (stub)

Win32 API functions

Three types of memory are commonly utilized by libraries: Heap, Thread Local Storage, and virtual memory. unDOS does not properly support Virtual Memory; it only passes back the address of the stack in VirtualQuery — specifically to support Borland's Multi-Thread 32-bit library. Thread Local Storage is emulated more thoroughly although there is only one set of local storage for the entire application, which again is considered a single Win32 thread.

HeapAlloc
HeapCreate
HeapDestroy
HeapFree
TlsAlloc
TlsFree
TlsGetValue
TlsSetValue
VirtualQuery

Miscellaneous

These miscellaneous DOS functions are provided primarily for compatibility with DOS-dependent libraries and applications that might use them.

unDOS emulates the DOS version function, returning version 6.20 as the current version.

DOS int 21H subfunctions supported

30h	Get Version Number (6.20)

Win32 API functions

GetCPInfo
GetCurrentProcess
GetVersionExA
LCMapStringA
MulDiv
RaiseException
TerminateProcess

The last error is tracked by unWin internally. Standard Win32 error codes are used. GetStartupInfo simply sets file handles field to NULL which is needed by Borland's Multi-Threaded RTL.

GetLastError
GetStartupInfoA
SetLastError

Video BIOS Services

unDOS provides some basic Video BIOS int 10h services. These are provided because the video BIOS is not accessible in protected mode. These services assume the presence of a standard VGA interface to a video card. The video card must have a standard VGA BIOS present and it must have been initialized already. The initialization of the VGA BIOS is generally done during the boot sequence of your computer.

These routines utilize the BIOS system RAM variables stored at absolute address 400h to 500h.

Some of these functions are considered to be "setup" functions which are very dependent on the video card such as set graphics mode. The VGA BIOS Emulator switches to virtual 86 mode, turns off interrupts and then calls upon the standard VGA BIOS to perform the function. This could mean that interrupts will be turned off for a long time so these functions should only be called at the beginning and end of the application, rarely and at non-critical times.

Alternatively, we provide two small DOS programs which will switch the system into and out of video mode. Grafmode.exe may be executed before pmEasy to set your video into VGA mode. Textmode.exe may be executed after exiting pmEasy to restore your video to standard text mode.

The functions supported here are attuned to what is required by graphics mode interface packages such as MetaWINDOW and MGL which are supported by unDOS. Features specific to text mode features are not generally supported.

BIOS int 10H subfunctions supported

00h	Set Graphics Mode (by switching to real mode using DPMI 300h)
02h	Set Cursor Position
0Fh	Read Current Video State
10:00h	Set Individual Palette Registers (EGA/VGA)
11h	Character generation has been stubbed out; forces graphics packages to use its own fonts.
12:00h	Return Video Information (EGA/VGA)
1A:00	Read Display Combination Code (VGA)

Unsupported Functions

In this section we discuss the functions which are not emulated by unDOS. The first thing to do is determine what functions are being called which are not supported by unDOS. After this, simply stub out or support the functions in question. Contact Micro Digital for help in this regard; possibly the functions you need have been supported by the time you need them. Otherwise, the helpful people at Micro Digital may be willing to do the work for you or at least point you in the right direction.

The following methods are suggested as ways to determine which unsupported function is being called:

Method 1 (logging)

Simply turn on the unDOS/unWin logging option in undos.h and recompile your application. You may select from 3 options:

1. LOGOPTION 0: No special action is taken when an unsupported function is called. Usually, a general error code will be returned by unDOS/unWin depending on the function called.
2. LOGOPTION 1: A message will be written to the display. This is especially useful when running in text mode. These messages are sometimes lost in the shuffle during a change to graphics video mode. The message may also be lost if a crash occurs.
3. LOGOPTION 2: A message is written to the display and each time a call occurs a message is appended to the logfile (“dos.log” as declared in urtl.c). This is more reliable if file I/O is working properly. This is not especially helpful if the call in question is being made before the file system is initialized. The primary benefit here is a list of the calls which your application is making that are not currently supported by unDOS. Currently, only DOS functions are logged, not Win32.

This is by far the most gracious method and should cover most cases.

Method 2 (break on unsupported call)

If the logging approach does not work, then you should run your application under your debugger. Set breakpoints on the following unDOS functions:

- Unsupp_DOS
- Unsupp_Win32

If you reach one of these breakpoints, simply step through the routine to return to the function that called it to determine what DOS or Win32 function is being called and who is calling it in your application.

For unsupported Win32 functions, you may wish to leave HALT_OPTION set to 1 in undos.h during development, which will automatically stop the debugger in the stubbed Win32 routine if one is called.

Method 3 (unstubbed Win32 call)

If you are in 32-bit protected mode and you are experiencing a GPF and none of the above steps were effective then your application is probably making a call to the Win32 API to a function which has not been stubbed out in unwin32_.c. This may happen if you are linking the Win32 import library or using a compiler which automatically links the DLL import information in so that non-stubbed calls do not create link errors. In the former case, do not link import libraries. In the latter, there are two approaches which may be used:

1. Laboriously step through your code with a debugger until you can track down the call to Win32 which is causing the GPF.
2. Search through your source code for your application and libraries to determine which Win32 calls are being used and compare this list of calls to the functions supported and stubbed off in the unwin32_.c files.

Method 4 (unsupported interrupt)

It is possible that your application will attempt to call BIOS and other specialized interrupt handlers which are completely unsupported by unDOS (i.e. those other than int 10h or int 21h). In this case, you will most likely see a message from pmEasy indicating that an unsupported interrupt has been called.

Glossary

DPMI Short for “DOS Protected Mode Interface”. This is a standardized user interface for DOS programs running in protected mode and is generally supported by DOS extenders. unDOS combined with pmEasy provides support for the most commonly used DPMI functions. See the pmEasy documentation for more details about the DPMI server.

GUI Short for “Graphical User Interface”. These are much friendlier interfaces and are becoming more and more popular even in the traditionally user unfriendly world of embedded, real time systems. The need for GUI support on the SMX platform has provided inspiration for the development of unDOS.

RTL Short for “Run Time Library”. This is the library generally supplied by your C/C++ compiler vendor which supports the standard C run time library functions. A major goal of unDOS and unWin is to support the functions in these libraries so that the SMX user may use standard functions.

3rd Party Libraries These are C/C++ libraries which are sold as separate products from the C/C++ compiler, i.e. a “third party” vendor. These libraries generally represent a great deal of labor and can save a developer man years of development effort. They are generally written to run on platforms which are widely used such as DOS, Windows, and Win32. One of the most useful products in this category is a GUI.

Index