THE ANDREW SCHULMAN PROGRAMMING SERIES

DOS INTERNALS

INCLUDES MEMORY AND DISK
EXPLORATION UTILITIES

GEOFF CHAPPELL

DOS Internals

Geoff Chappell

Series Editor
Andrew Schulman

Addison-Wesley Publishing Company
Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn
Sydney Singapore Tokyo Madrid San Juan
Paris Seoul Milan Mexico City Taipei

Library of Congress Cataloging-in-Publication Data
Chappell, Geoff.
DOS internals / by Geoff Chappell.
p. cm.
Includes index.
ISBN 0-201-60835-9
1. MS-DOS (Computer file) 2. PC-DOS (Computer file) I. Title.
QA76.76O63C452 1994
005.4'469--dc20 93-46019
CIP

Managing Editor: Amorette Pedersen
Set in 10.5-point Galliard by Benchmark Productions, Inc.

123456789-MA-9897969594
First printing, January 1994

To A.,
You can never be alone
while you can laugh at Buñuel

Preface

Scope

It should be understood from the outset that this book is an analysis primarily of DOS 5 and 6. Constraints on time (and, let it be said, interest) have not always permitted special cases arising from earlier DOS versions to be taken up for discussion.

There are programming examples in this book which insist on having DOS 5 or higher. The most notable case is a library (developed in Chapter 7) which binds device driver initialization to the C run-time library. The code avails itself of the safety feature new to DOS 5 by which device drivers are supplied with an upper bound on the memory they may use during initialization. No doubt some vaguely satisfactory workaround could be contrived to support earlier DOS versions, but the attempt is not made. The decision in this particular case is borderline but my attitude is to not support DOS versions before 5 if the effort would be out of proportion to (and therefore distract from) the didactic point being made at the time.

Going Back

That said, you will find that analysis usually extends backwards to versions 4.01 and 3.30, both of which I have studied in comparable detail. Here again, there are decision boundaries about how much effort to spend over, say, the description of an interface that has changed since one of these versions. An example (from Chapter 16) is the undocumented int 2Fh function 08h through which IO.SYS supports the definition by DRIVER.SYS of extra units for the default block device driver. The structure in which IO.SYS maintains all its knowledge of block device units changed markedly for DOS 4 and, as far as this book is concerned, it must suffice just to note that the structure given in the text applies only from DOS 4 onward. I hope that all differences of any significance have at least been noted in this fashion.

Earlier versions than 3.30 were already uncommon by the time I wrote my first DOS program in 1989. Viewed through my eyes, all versions before 3.30 belong to some dim pre-history on which weak light is thrown occasionally by inspection of software that supports a range of DOS versions. A particularly well-informed source of information about earlier versions (and which has sufficient importance to justify close inspection) is Microsoft Windows, which supports DOS versions from 3.10 onwards.

As far as this book is concerned, DOS 3.10 is an event horizon. Put another way, to anyone concerned with programming DOS at system-level, DOS versions 1.xx, 2.xx, and 3.00 may as well lie in a black hole.

It is as well to offer system-level programmers some guidance over DOS versions while the boundaries and prejudices are being explained. DOS 1.xx appears to have been regarded generally as off-limits for many years: it is well-known that this version has no notion of directories or file handles, let alone support for memory management and program execution. These few words in this paragraph are all that will be said about DOS 1.xx in this book. No account is even allowed for DOS 1.xx in the programming examples, all of which assume that the case will be handled by the C run-time. I cannot resist the facetious remark however, that DOS programmers who cut their work off from the possibility of running under DOS 1.xx are doing themselves out of some small slice of the market: I know at least one person who continues to use (for normal secretarial work, no less) a genuine IBM PC running DOS 1.10 from 320K floppy disks.

For the most part, DOS 2.xx offers all the facilities that will usually be required by programs which work at the level of files and above, i.e., by ordinary, well-behaved applications. There are, however, good grounds for even applications programmers to insist on some form of DOS 3, at least. It may seem a bit excessive to discount a whole major DOS version for the following little reason, but this is what I recommend: In DOS 2.xx, there is apparently no way to fail an int 24h exception. True, it is possible to fiddle with registers on the stack and return to the application rather than to the DOS kernel, but Chapter 5 shows that this technique is suspect even under later DOS versions, primarily because of the difficulty of determining which DOS function has encountered the error, and thence what error indication (if any) should be prepared for the AX and flags registers. This lack of a Fail option complicates critical error handling unreasonably. Supporting DOS 2.xx will often involve implementing a full int 24h handler to be sure that if the user wants to terminate the program, then at least the cleaning up will remain within the program's control. Even with this extra work, however, the quality of error recovery cannot hope to be as good as can be achieved in later versions by the simple expedient of just failing DOS functions that encounter critical errors.

When it comes to systems-level work, you don't have to be at all adventurous to run into a plethora of old wives' tales. It is unimaginable that anyone competent will ever track down all OEM variants of DOS 2 and analyze them to settle some of these hoary old stories. No doubt the cases which were found important for TSR programming at the time are well established, but they are unlikely to have been covered exhaustively. This is why I describe them as old wives' tales: They will be rough and ready formulations that seemed to get the job done. Handwaving and folklore have no place in this book; at least, that's the intention. The spirit of this book is that programs should be safe and that programming should be sure. Knowledge brings confidence that ambitious projects can be made to work reliably, preferably first time.

To some extent, the question whether to support DOS 2.xx is academic, since I contend that even DOS 3.00 is not a sufficiently reliable platform for adventurous DOS programs. The most glaring difficulty for TSR programming is that there is no means to restore extended error information. It looks as if DOS 3.00 was a trial run preceding

proper support for networking. Internal structures and functionality seem not to have been finalized and, perhaps inevitably, there are omissions.

In Microsoft's scheme of networking, a remote machine's request for file I/O is treated very much like the same request from a pop-up TSR. The desire to support networking appears to have concentrated the mind wonderfully on questions of re-entrancy and the nature of certain system resources. A consequence of good planning for network support is that the DOS kernel has enjoyed remarkable stability from DOS 3.10 onward.

Windows requires DOS 3.10 or higher. This is surely no coincidence. The questions noted above are also the ones of main interest when programming TSRs or more generally any sort of task-switcher. For a large class of programming exercises that lie beyond ordinary well-behaved access at file level, the general self-consistency and uniformity of the DOS programming model in versions 3.10 and higher, coupled with the justification that this version is required by the ubiquitous Windows, suggests that is wholly reasonable for systems-level programmers to insist on having at least DOS 3.10 present when their programs run.

It happens that concerns related to task-switching are not the whole story and that DOS 3.10 might not suffice for some applications. Although the interesting questions of process and resource management seem to have been thought out respectably well for 3.10, it was not until DOS 3.20 that attention was directed to giving DOS applications the means to exercise fine control over disk structure, that is, to call through to device drivers for specialized services like disk formatting. Generic IOCTL (used for the disk duplicating program presented in Chapter 18) is available only from DOS 3.20 onward.

Well, all new DOS versions bring new functionality that applications may use in their dealings with the system (or so one might have supposed until recently) and there will always be reasons to restrict some program's operational domain to some narrow range of probably recent DOS versions. These notes are an attempt to explain my particular feelings on the matter. In fact, close inspection of the programming examples accompanying this book will show that checking the DOS version is the least priority.

Looking Ahead

Some readers will wonder about DOS 6. This is unfortunate, for I think it shows that even those sections of the trade press which concern themselves with technical aspects of DOS programming have given DOS 6 rather more attention than it deserves.

Let there be no mistake: The programming model offered by DOS 6 differs hardly at all from the one provided by DOS 5. The resident portion of IO.SYS and the whole DOS kernel (MSDOS.SYS) are barely changed. As some measure of this, the *most* significant point I can see to make about the DOS 6 kernel is that it fixes a bug by which the DOS 5 kernel, when configured for **dos=high** with BUFFERS in the HMA, would lose track of error conditions when reading sectors into the BUFFERS system. The possibility has not

yet been discounted that the simple oversight responsible for this had already been corrected in later releases of DOS 5.

A graphic illustration of the similarity between DOS 5 and DOS 6 at system level is that a DOS 5 machine may be booted in perfect safety, gaining the facility for multiple configurations, if the DOS 5 IO.SYS is updated to the DOS 6 version *with everything else being left intact.* Not even for DOS versions 3.20 and 3.30 is it possible to boot with IO.SYS from one version and MSDOS.SYS from the other.

Fortunately, there is more to DOS 6 than the system files. Quite a number of substantial applications have been bundled with DOS 6 and these presumably justify incrementing the major version number.

It is the book's overall policy to deal only with DOS features that are somehow intrinsic to the DOS programming model and that will be encountered by ambitious DOS programmers. This serves to excuse no end of analysis of Windows, sometimes reaching far beyond what some may think appropriate in a book on DOS. If the presence of some piece of software may cause the DOS programmer some difficulty and is a matter of sufficient interest to have held my attention for long enough to write about it, then it may very likely appear in the book.

Thus, it is one thing to examine HIMEM.SYS as the standard XMS server and to consider how its behavior is driven by the installation of other software such as the task-switchers for the MS-DOS Shell and Windows 3.1 standard mode. Throughout DOS's twilight years (and it could be a surprisingly long twilight), the XMS seems likely to be an important feature of DOS programming. It would be quite another thing however, were this book to study RAMDRIVE.SYS just because it happens to be an XMS client.

Similar arguments apply to DoubleSpace. I am aware that this disk compression software bundled with DOS 6 has generated enormous interest and that there is sore need for sober analysis. While Chapter 3 describes the interface between IO.SYS and DBLSPACE.BIN is some detail, this book is not the place for a discussion of DoubleSpace itself. Though DBLSPACE.BIN is billed as a new system file, this role arises only as the means to secure its presence on bootable floppy disks, and that IO.SYS should know about the file in order to enable compressed drives as early as possible. In programming terms however, DBLSPACE.BIN is far from fundamental: By hooking into DOS's device driver communications, it presents the contents of a file (the DoubleSpace Compressed Volume File, or CVF) as a DOS drive. Thus, DBLSPACE uses the structures and interfaces DOS already supports for drive management. These you will find discussed in Chapter 16, since they need to be understood by any DOS programmer intending to work close-in with standard disk drive storage on PCs. As far as this is concerned, the only special consequence of DoubleSpace's distribution as a component of the standard DOS package is that users might reasonably have expected rather more in the way of safety and compatibility. That Microsoft can have fallen short in these factors when releasing the version supplied with the DOS 6.20 upgrade must count to the industry's undying shame.

Notations and Conventions

Inevitably, in a book whose programs are mostly written in C but whose author is arguably not a C programmer, a few things should be explained from the outset.

All C source for this book is intended for compilation under Microsoft C 6.0. Perhaps it need not be mentioned here (since the realization will follow from just a cursory glance at the source code), but the compiler is used very much as a tool for high-level assembly language. For the few modules written in assembly language, the expected assembler is MASM 6.0.

There will undoubtedly be some notational changes required for porting the C source code to one of the competing C compilers for DOS, but I expect this process to be fairly mechanical. Some important cases exist, however, where the code depends strongly on labels that are probably peculiar to the Microsoft C run-time library. This applies to the two programs presented in Chapter 5, for reasons that are completely unrelated. In one, it is a simply irresistable convenience to reach deep within Microsoft's implementation of the `spawn` family of functions in order to replace their common core but to keep their ability to do such things as to follow the PATH environment variable when searching for a program. This gives the book's programming example the power of a real program rather than an instructive gadget. Nonetheless, it is the instruction which is the program's claim to existence and readers should be prepared to do their own possibly heavy adaptation if they want to port the program to another compiler. It is hoped that the dependencies will be flagged well enough that the task should not be odious.

This point is strongest in Chapter 6, which develops a scheme for resident programming in C. The object is not to develop a universal scheme but to see how problems posed by the compiler at hand may be resolved through combining an understanding of DOS's behavior with the will to overcome difficulties. For a different compiler, there may be different problems, but again, all I can hope is to have marked out the course sufficiently clearly.

Standard Macros and Symbols

Among this book's header files is one which contains symbols and so forth, that the author prefers to have handy when programming in C:

```
/* standard.h */

#ifndef     _STANDARD_INC
#define     _STANDARD_INC

/* Logical operators and values */

#define     AND     &&
#define     NOT     !
```

```
#define     OR      ||

#define     FALSE   0
#define     TRUE    1               // for consistency with TRUE = NOT FALSE

#define     OFF     0
#define     ON      1

#define     CLEAR   0
#define     SET     1

/* Convenient data types */

typedef unsigned char       BYTE;
typedef unsigned int        WORD;
typedef unsigned long       DWORD;

typedef signed char         SBYTE;
typedef signed int SWORD;
typedef signed long         SDWORD;

typedef unsigned char       CHAR;

typedef int                 BOOL;

/* Macro for generating a far pointer from segment and offset */

#define MK_FP(seg,off) (((_segment) (seg)) :> ((void _based (void) *) (off)))

/* Macros to decompose 16-bit and 32-bit objects into high and low
    components and to reconstitute them */

#define HIGH(x) ((BYTE) ((x) >> 8))
#define LOW(x)      ((BYTE) (x))

#define MK_WORD(high,low)  (((WORD) (high) << 8) | (low))

#define HIGHWORD(x) ((WORD) ((x) >> 16))
#define LOWWORD(x)  ((WORD) (x))

#define MK_DWORD(high,low) (((DWORD) (high) << 16) | (low))

/* Macros for directing the compiler to use current segment register
    values rather than generate relocatable references */
```

```
#define     CODESEG              _based (_segname ("_CODE"))
#define     CONSTSEG    _based (_segname ("_CONST"))
#define     DATASEG              _based (_segname ("_DATA"))
#define     STACKSEG    _based (_segname ("_STACK"))

/* Macro for NULL in case using STDLIB.H would be inappropriate */

#ifndef NULL
#define NULL ((void *) 0)
#endif

#endif
```

Standard Makefile Inclusions

Likewise, all the book's makefiles include the following standard provisions, if only to hide awkward things like ensuring that compiler options don't sneak in from environment variables.

```
# standard.mak

# This is an NMAKE include file for projects containing assembler and C.

# Pre-defined inference rules present some problem when compiling C source
# files with the /Fa option to generate assembly listings. Once the .ASM
# file exists, NMAKE assembles the old listing rather than compile the new
# C source even if the inference rule contains an explicit .C dependence.
# Fix this by reversing the order of .ASM and .C in a new .SUFFIXES list.

.SUFFIXES :
.SUFFIXES : .c .asm

# Utilities not known to NMAKE by default:

LIBRARIAN = lib
LINKER = link
MAPSYM = mapsym

# Favored switches for the assembler, compiler and linker:

ASWITCHES = /Cp
CSWITCHES = /Gs /Oas /W3 /Zpl
LSWITCHES = /noi

# Generate listings unless "LISTING=OFF" is given on the NMAKE command
```

```
# line or precedes the inclusion of this file.

!IFNDEF LISTING
LISTING = ON
!ENDIF
!IF "$(LISTING)" == "ON"
ALIST = /Fl /Sg
CLIST = /Fa
LLIST = /m
!ENDIF

AFLAGS = $(ASWITCHES) $(ALIST)
CFLAGS = $(CSWITCHES) $(CLIST)
LFLAGS = $(LSWITCHES) $(LLIST)

ASSEMBLE = $(AS) /c $(AFLAGS)
COMPILE = $(CC) /c $(CFLAGS)
LINK = $(LINKER) $(LFLAGS)

# Pseudo-targets:

ALL : CLEAR TARGETS

# The first pseudo-target is a possibly over-cautious insistence on
# clearing flags passed to the assembler, compiler or linker through
# environment variables.

CLEAR :
  set $(AS)=
  set $(CC)=
  set $(LINKER)=
```

Header Files

Heavy use is made of a collection of header files defining structures, etc., for key elements of DOS programming. It is hoped that the use of long identifiers makes it easy enough to follow source without having to reprint these many header files in the text of the book or even as an appendix, especially when many of the structures concerned are presented in tabular form anyway. All include files appear on the accompanying disk.

In the inclusions at the top of each C module, it is assumed that header files from this collection are available on a path defined in the INCLUDE environment variable. The filenames are therefore enclosed in angle-brackets. For clarity, standard header files and those specific to this book are presented in two separate blocks of inclusions.

Nomenclature

In general, structure elements and public data are named with a prefix to describe the data type, following the so-called "Hungarian" scheme presented in the Windows SDK. Given the author's clear preference for unsigned quantities, only some of the prefixes given in the Windows SDK are used here:

- b int (two bytes) taking Boolean values
- c char (single byte)
- dw dword (four bytes)
- lp far pointer
- p pointer (typically a 16-bit offset, given the predominance of small memory model)
- w word (two bytes)

All the book's source code, whether for C or assembly language, is case-dependent. If a structure element, function or public label has a compound name, then each element begins with an upper case letter for readability, as in SomeFunctionName or dwLongStructureElement. Macros and other equates are usually given in capitals only, using underscores if they seem required for readability, as in SOME_MACRO. Names for automatic data and many static variables often consist of lower case letters only: since their scope is severely restricted, their readability (if not also the name itself) is perhaps not so important.

Typeface Conventions

Some attempt is made at using different typefaces to emphasise the technical role of a word or phrase in the book's main text.

When DOS commands such as **fdisk /status** are used mid-sentence, they appear in bold. The same applies to configuration statements like **multitrack=off** or **RemMediaCD ChngTrack=on** (which are permissable in CONFIG.SYS and SYSTEM.INI respectively) and, less commonly, to keywords from programming languages.

Filenames and program names are given in upper case when they are not in the context of a command. Thus, text might speak of the WIN.COM program that starts Windows but of running the command **win ::** to see Windows crash.

Function names that might ordinarily appear in programming code retain the use of a fixed pitch typeface when extracted into the main text, as happens for instance when a point is made about the `_chain_intr` function in the Microsoft C library. Fixed pitch is also used when taking several lines to show the contents of a text file or to present the results of running a program.

Acknowledgments

It seems customary in books that delve into DOS or Windows programming to thank a list of software developers who have helped with information, including some who must remain nameless. This is not the case here, but since this is my first book, please bear with me while I thank those who have played key parts in bringing me to a study of DOS programming, in addition to thanking those with a closer connection to the book itself.

Although he may not remember it, Professor Jonathan Kaye of the Department of Linguistics at the School of Oriental and African Studies (SOAS) played an instrumental role in seeding my over-indulgent interest in programming way back in 1989.

Without the students living at the International Hall of Residence at the University of London, some of whom used to insist on misbehaving with a few communal PCs (and now just have a canny knack of doing strange things with their own PCs), I might never have found the motivation to uncover the truth behind DOS's inner workings, or at least to put my findings into practice. While this book was being written, many of the Hall's students and staff (especially its Warden, Donald Mann) found time for words of encouragement. They probably think their efforts were not even noticed, so gloomy was my outlook at some points in this book's history.

Major Reg Cross and his wife Mary Cross have been consistently concerned for my well-being. Reg suggested many years ago that I had the right sort of mind for computing. I'm not sure I yet agree with him or that I've taken up the subject in quite the way he had in mind, but I know he'll take it in his usual good humor if I name him as the one to blame for all this.

Neither Miles Dennis nor Martyn Lovell is primarily a PC programmer, but it was invaluable having them at hand in the first few months to read over rough work for the book and to assure me that my mental flotsam was surviving its translation process. My respect for their technical abilities is no faint praise, as readers of this book will quickly appreciate. Miles also has been a consistently concerned and supportive friend. And I'm not saying that just because I've had near to permanent loan of his nice fast modem.

Had Miles and Martyn not insisted, I might never have joined CIX, a system for conferencing and email which first put me in touch with professionals who were interested in technical aspects of DOS programming. Immediately, I was encouraged to make something of my DOS study, which just a little earlier I had thought to abandon. I should like to single out David Jewell, author of the forthcoming book *Polishing Windows*, who has been particularly helpful by way of encouragement and always ready to act as a common-sense sounding board (until he too succumbed to this degenerative condition called book-writing).

Dan O'Brien, also met through CIX, may not have known what he was starting when he gave me the opportunity to review the book *Undocumented DOS* for *.EXE* magazine in April, 1991. On the other hand, he may have had a pretty good inkling that the review would be critical. This book is very much Andrew Schulman's Revenge. It is a double-edged blade however; Andrew has surely had many "heart attacks" while wondering how to cope with his author's curious approach to writing a book. He has taken a lot of trouble to see that the results actually appear in print, going so far as to write the introductions and links that I simply didn't do.

Ralf Brown was kind enough to read my scrappy manuscript and make sense of it. This can't have been easy, as you the reader are about to discover, in spite of the changes and second-thoughts prompted by Ralf's analysis.

At Benchmark Productions and Addison-Wesley, there are a host of people I have never seen but whose work is responsible for transforming my manuscript into the book you have before you. Those I know about are Abby Cooper, Ann Lane, Amy Pedersen, Chris Platt, and Andrew Williams. I have at least talked to Chris Williams and Phil Sutherland. If you like this book, please write to Phil at Addison-Wesley: in doing this, you may help restore them from the depths of despair to which I've no doubt reduced them.

Professor John Taylor, Director of the Centre for Neural Networks at King's College, London has been very kind to keep in touch and keep me in mind despite the ever-lengthening time that this book has prevented me from engaging in useful research (not to mention writing up results).

Special personal thanks must be extended to long-standing friends Brad McCusker, Anne Tully, and Lorna Gibb, not for any particular connection with the book, but for consistently showing far more faith in me than can be reasonably justified and for just about managing to counter-balance my persistent sense of abject failure. My parents, Myles and Angela Chappell, in Bundaberg, Australia, have been truly supportive despite my great distance from home. Without their assistance at vital moments, this book would certainly have been aborted. Please recommend this book heartily to all and sundry so that I can visit my parents and take my friends out to dinner.

When I was just a teenager, a kind, gracious gentleman by the name of Fred Hendricksen took a large sum of money from his own pocket for me to buy books. Now I've written one, Uncle Fred.

Geoff Chappell
(geoffc@cix.compulink.co.uk)
London

Contents