Behavior Research Methods, Instruments, & Computers
1992, 24 (3), 497-500

— BOOK REVIEW —

Book review of DOS 5: A developer's guide

PHILLIP L. EMERSON
Cleveland State University, Cleveland, Ohio

DOS 5: A Developer's Guide: Advanced programming guide to DOS
By Al Williams. 1991, Redwood City, CA: M&T Publishing. 914 pp. (with two diskettes of source program code). $39.95.

I am reviewing this book because I like it, as I shall explain. It is organized into three main divisions:

Part	Chapter	Topic
I		Basic MS-DOS Programming
	1	The PC hardware: An Overview
	2	The Application Environment
	3	C and Assembly for DOS
	4	Programs at Last!
	5	DOS Services
	6	ROM BIOS Services
	7	Direct Access Techniques
	8	The Coprocessor
II		Advanced MS-DOS Programming
	9	Building Robust Applications
	10	Graphics Programming
	11	Of Mice and Men
	12	Expanding Horizons: EMS
	13	Device Drivers
	14	TSR Programming
III		Protected Mode Techniques
	15	80386 Protected Mode
	16	Using Extended Memory
	17	80386 Debugging
	18	Accessing 4 Gigabytes in Real Mode
	19	DOS Extenders

This review is from the perspective of a behavioral researcher who has been using MS-DOS with IBM PC compatible equipment in the laboratory for a number of years. Parts I and II cover essentially the same ground as do earlier books of this nature (such as Norton, 1985; Sargent & Shoemaker, 1986), but with some welcome differences. Particularly welcome are the ready-to-compile and ready-to-assemble routines on diskette. In addition, Part III appears to be the first treatment, in a book of this kind, of important advanced features of the 80386, the 80486, and, to some extent, the 80286.

The book is written primarily for people who can do some programming and includes quite a lot of usable source code in C and in assembly language. In that respect, even Parts I and II differ somewhat from earlier treatments of the same topics. These sections of source code are used as examples in the hard-copy text but are also included on two 5.25-in. diskettes. That is fortunate, because some of the programs serve not only as examples but also as directly usable utilities that can be incorporated into applications with little or no modification.

Readers having no prior experience with either C or assembly language programming are not likely to enjoy this book. It clearly is not a good place to start learning to program. On the other hand, the techniques range widely in complexity, so some of them would be easy enough to adapt and use by tinkerers who have only a modest amount of programming experience. The choice of C and assembly language is wise, because these are the languages that would be used in most serious applications. Moreover, some of the applications would be much more difficult and inefficient if programmed in BASIC, for example. Many of the programs can be directly incorporated into some C language applications program, whereas the BASIC demonstration programs in the more chatty books by Goldstein (1989) and Norton (1990) need to be translated if BASIC is not used.

The recommended C compilers are Borland Turbo C Version 2.0 or higher, Microsoft C Version 5.0 or higher, or Mix Power C Version 2.0 or higher. The recommended assemblers are Microsoft MASM Version 5.0 or higher and Borland Turbo Assembler Version 1.0 or higher. An assembler is crucial only for the programs presented in chapters 17 and 19, for some in chapter 18, if Turbo C 1.5 or Microsoft C is used, and for a few other small applications scattered in other chapters.

The objective in all of the example programs is to take control of and exploit the capabilities of the machine and operating system. All examples actually do something and are not merely fabricated for rhetorical purposes. True, the first two programs (in chapter 4) have modest goals: to send escape sequences to the console or printer and to count the number of spaces in a keyboard command line. These are things that a programmer might need to do in real life, and the techniques used are perhaps cleaner

Correspondence should be addressed to the author at the Department of Psychology, Cleveland State University, Cleveland, OH 44115.

and more thoughtful than those that some of us might hatch on the spur of the moment; they illustrate techniques and concepts introduced in the first three chapters.

Chapters 5 and 6 are the necessary encyclopedic reference sections on BIOS and DOS services, but with enough commentary to keep a browsing reader interested for a few pages at a time. These reference sections, of course, do not cover all the minute details, and the book refers readers needing more detailed information to the dense reference book by Microsoft (1991).

In chapter 7, I began to find some things that I really wanted to know, rather than things that I needed to know in order to browse further. Browsers tend to do a lot of skipping on the first pass, expecting certainly the occasional necessity to back up and rebrowse. A browser who has a rebrowser companion can sometimes profit from advice. My advice is to study, for later purposes, the A20.C routine on page 197, in chapter 7. This routine is concerned with memory addressing on an AT above the 1-MB limit of the 8086 processor.

Elsewhere in chapter 7, casual browsing is appropriate according to one's interests. Among the topics are some that have always been of interest to researchers using the computer to control real-time experiments, such as (1) writing text to screen memory, (2) interrupts, soft and hard, (3) direct keyboard access, (4) interacting with DOS memory allocation, (5) timing and sound generation, (6) joy sticks, and (7) parallel and serial interfaces. There is no coverage of less standard add-on devices, however, such as AD and DA converters. I found the TONE.ASM and TONETEST.C programs of interest, not so much for what they actually do (produce tones on the PC beeper speaker), but more for the suggestions in the accompanying commentary. One suggestion is that the tone frequency generator (channel 2 of the 8253 or 8254 timer) can freely be used for anything that the programmer desires, without interfering with any operating system functions, except that those familiar beeps may be eliminated or modified. Furthermore, channel 2 of the timer chip can be used as a real-time timer, programmable to count at almost any rate between about 20 Hz and 2 MHz. The programming is essentially the same as that for channel 0 on the same 8253 or 8254 chip, but users of timing routines on channel 2 need not worry about disturbing the IBM BIOS real-time clock, as must those using channel 0 (see Dlhopolsky, 1988; Emerson, 1988).

Chapter 8 is about programming the numeric coprocessor (80×87). C compilers usually handle the coprocessor chip reasonably well (but see Emerson, 1989, for some variations among compilers), so there would seem to be little motivation for a behavioral researcher to program the coprocessor directly.

My comments on Part II (chapters 9-14) are brief because these chapters seem more relevant for commercial programming than for research applications. However, there is good general advice for any programmer in chapter 9, whereas chapter 12 begins to get into important topics concerning access to memory above the 1-MB limit. Chapter 12 illustrates the use of expanded memory boards on any 8088 or 80×86 with the Lotus-Intel-Microsoft specification (LIM, EMS, or just EMS). This specification provided the earliest standard method of breaking the 1-MB barrier with MS-DOS, and it still is used widely for backward compatibility with older machines and old software on newer machines such as the 80386 that do not require it. An EMS board contains the memory chips for the extra memory but also contains processing hardware that looks like an external device to the main processor (e.g., 8088). Thus, programs that use EMS do not address EMS memory locations directly but must temporarily give up access to some part of lower memory to provide address space to EMS. Given an 80386, the other method (extended memory) of obtaining access to memory above 1 MB has fewer limitations, which is one of the main subjects of Part III of this book.

From my point of view, Part III alone is worth the price of the book. It explains methods of allocating and accessing extended memory on the 80286, 80386, and 80486. Some of us are old enough to remember how the DEC PDP-9 and PDP-15 overcame the PDP-8 problem of paged memory addressing; a similar method was used in the design of the 80386 to overcome the problems of the segmented structure of 8086 memory. However, MS-DOS was designed for the 8086 and not for the advanced features of the 80386, except for a few that were added in later versions of MS-DOS. Part III shows how, by astute usage of those few added features, to make use of some of the advanced capabilities of the 80386 even while operating with MS-DOS.

Chapter 15, the first chapter in Part III, probably should be browsed at least cursorily for an introduction to the terminology and logic used in the applications in the later chapters. In chapter 16, an extremely useful set of small routines is introduced that provides a means for C programmers to allocate and access upper memory. Their prototypes are shown in the top part of Table 1. The last four of them are analogous to the C library routines alloc(), realloc(), memcopy(), and free(), but can operate on extended memory above the 1-MB barrier. With extmemcopy(), either the source or destination arrays, or both, can be either in upper or lower memory. The type LPTR is specified as unsigned long in a typedef statement in a header file; it means "linear pointer," which is different from the ordinary segmented pointers used in standard MS-DOS. The user must use a couple of small macros or routines to convert between linear and segmented far pointers.

The main limitation of the routines presented in chapter 16 is that to be operated on by ordinary programs or transferred from lower to upper memory while lower memory is used for something else, the upper memory array must be transferred to low memory in chunks (of 128 K maximum length). Also, this method works to access only up to 16 MB, whereas the 80386 in principle can access 4 GB. However, this method works with the 80286 as well as with the 80386 and 80486.

Table 1
Prototypes of Routines Presented in Chapters 16 and 18 for Extended Memory Access

```
Chapter 16, for 80268, 80386, 80486
unsigned ext_size(void); /* get size of extended memory */
LPTR ext_alloc(unsigned length) /* allocate a block of memory, of size length, in K */
LPTR ext_realloc(unsigned length); /* reallocate a block */
int extmemcopy(LPTR dst,LPTR src,unsigned wc); /* copy an array of wc words */
int ext_free(int flag); /* free memory when done */
Chapter 18, for 80386, 80486
void extend_seg(void); /* set 80386 seg regs to 4 GB limits */
void a20(int flag); /* enable a20 address line */
unsigned int big_read(LPTR address); /* retrieve a byte */
void big_write(LPTR address,unsigned byte); /* store a byte */
void big_fer(LPTR src,LPTR dst,unsigned long count); /* copy an array */
```

Chapter 17 deals with advanced debugging features of the 80386, and I may return to it someday. I browsed it rather quickly, because I have often found debuggers harder to understand than the programs that I want to debug. My impression, though, is that a properly designed 80386 debugger can act like a hardware debugger and therefore be less confusing than some of the software debuggers. In chapter 17, a fundamental 80386 debug program is presented with several options and optional accessories. The routines are written in assembly language, except for a couple of small C header files needed for debugging C programs.

Chaper 18 implements an ingenious method (Roden, 1989; Williams, 1990) for using all available memory on an 80386, in real mode under MS-DOS, and by programs written either in C or in assembly language. This is accomplished by calls to the routines whose prototypes are shown in the bottom section of Table 1. The routines of chapter 16 (upper section of Table 1) are used in the examples in chapter 18 to allocate and free blocks of extended memory to be used by big_read(), big_write(), and big_xfer(). Some potential users may anticipate storing and retrieving only aligned 16- or 32-bit words rather than 8-bit bytes. In that case, the big_read(), big_write(), and big_xfer() routines could be revised in the assembly code listed in chapter 18 (pp. 702-705).

The main trick behind the method of chapter 18 is accomplished by the extend_seg() routine, which temporarily enters protected mode, sets some 80386 segment registers to their maximal limit of 4 GB, and returns to real mode. Amazingly, none of the ordinary operating system's processes are disturbed by this operation or by the associated calls to big_read(), big_write(), and big_xfer(). However, the method can come into conflict with other programs that access upper memory, such as Windows and data base programs. Some memory management is taken care of by the small routines presented in chapter 16, but those facilities cannot deal with complex multiprocessing. The safest precaution in using these methods is to make sure that no other programs are coresident by removing all "device" directives from the CONFIG.SYS file and rebooting. That may be a bit of overkill, however, and experimentation can reveal which devices can safely be left in. For example, I found that the HIMEM.SYS driver of DOS 5.0 can be left in, but if the DOS=HIGH directive is included, MS-DOS refuses to grant any extended memory space to ext_alloc(). Another precaution is not to spawn programs written by this method from other resident programs such as Codeview, Windows, or the shell commands of integrated environments.

Chapter 19 is about DOS extenders, and it presents the complete assembly code, in two versions (PROT and X386), for such a program. A DOS extender is a more ambitious programming project than are any of the others presented in this book, and it is done almost entirely in assembly language. DOS extenders provide better memory management to avoid clashes between multitasking programs in their accesses to memory. The goals are lofty, as is apparent from this description:

> [PROT] allows you to write assembly language programs that use 32-bit addressing and access all the 386's features. In addition, PROT allows you to do I/O using ROM BIOS or DOS. This extender also has provisions for direct access to the PC hardware (for instance, to write directly to the screen). (p. 713)

The other version, X386, is described as having the additional capability of making it possible to write protected mode code in any language that can generate an interrupt. I have not tested these chapter 19 programs, but I see no reason to doubt the claims made about them. They go beyond my present needs, which is likely to be true for other potential users of stand-alone 80386 systems for laboratory applications.

My only complaint about this useful book is that the publisher omitted chapter numbers from the running heads. A book that is used as a reference is made much more useful by the inclusion of that feature. Many of the utilities in this book can be used by anyone who knows how to copy MS-DOS files and type command lines to a C compiler or assembler. They can be adapted for special applications by users who have a bit of experience in programming.

For assembling, I used MASM 6.0 rather than MASM 5.0, which was used in the book, and found that some small changes were needed in some of the assembly language programs to achieve compatibility. I tried Turbo C 1.5 for the compiling, although Turbo C 2.0 or higher was recommended, and found that it worked for most programs and

produced much smaller .EXE files (about half the size) than did the Borland Turbo C++ version of C, but it required the use of the assembly language version of the SEG4GB file in chapter 18.

This book gives a balanced treatment of the topics of basic and advanced MS-DOS programming and of protected mode techniques. My interests were not as balanced, but I profited from those sections that dealt with my interests. I think that readers with other interests might find the same.

REFERENCES

DLHOPOLSKY, J. G. (1988). C language functions for millisecond timing on the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **20**, 560-565.

EMERSON, P. L. (1988). TIMEX: A simple IBM AT C language timer with extended resolution. *Behavior Research Methods, Instruments, & Computers*, **20**, 566-572.

EMERSON, P. L. (1989). NEWSART: Negative binomial weighted spectral analysis in real time. *Behavior Research Methods, Instruments, & Computers*, **21**, 353-368.

GOLDSTEIN, L. J. (1989). *IBM PC and compatibles* (4th ed.). New York: Simon & Schuster.

MICROSOFT CORPORATION. (1991). *MS-DOS programmer's reference: Version 5.0*. Redmond, WA: Microsoft Corporation.

NORTON, P. (1985). *The Peter Norton programmer's guide to the IBM PC*. Redmond, WA: Microsoft Press.

NORTON, P. (1985). *Inside the IBM PC and PS/2* (3rd ed.). New York: Simon & Schuster.

RODEN, T. (1989, November/December). Four gigabytes in real mode. *Programmer's Journal*, pp. 89-94.

SARGENT, M., III, & SHOEMAKER, R. L. (1986). *The IBM Personal Computer from the inside out* (rev. edition). Reading, MA: Addison-Wesley.

WILLIAMS, A. (1990, July). DOS+386=4 gigabytes. *Dr. Dobb's Journal*, 62-71.

(Manuscript received January 6, 1992;
revision accepted for publication April 11, 1992.)