Sasha Solodilov

CSCI 475

May 3, 2022

Microsoft Disk Operating System (MS-DOS)

The advent of the disk operating system was a groundbreaking discovery in the field of computer science, and it greatly propelled the development of today's modern operating systems. The disk operating system, or DOS, came about during the time when personal computers started to gain popularity in the 1970s and 1980s. The Microsoft Disk Operating System (MS-DOS), based largely off of an 86-DOS developed by programmer Tim Patterson at Seattle Computer Products, is an operating system that runs from a disk drive. Users are able to interact with the MS-DOS through a command-line environment, where programmers must either type out all commands on a keyboard or enter them in through a file. The DOS responds to a multitude of commands all stored in the DOS directory on a disk, and, although seen as antiquated and unsuccessful by some, it was an effective way of OS communication for its time, and is the basis for the modern-day shell.

MS-DOS does not contain thread support, and only allows one process to be run at a time, as was the standard for the MS-DOS and Control Program/Monitor era. In the 1980s, when the MS-DOS was most widely used, users would "execute an application, exit from it back to the prompt, and then invoke another [application]" (Varma). This means that MS-DOS was designed for a single-user environment, and it only had a single thread of execution, as it strived to "minimize resource utilization, both in memory and on disk (Varma)."

Since MS-DOS does not support multithreading and synchronization, it is significantly different from the treatment of processes and threads that were touched upon in class. Most importantly, the inability of MS-DOS to run multiple threads of execution means that it does not have responsiveness, and a single program is not able to multitask. With single-threaded execution, actions would have to be done sequentially, which means that the performance of processes run on MS-DOS as compared to an operating system with threads would likely be much lower. A few sources suggest that MS-DOS “could [potentially] have a user-level thread library,” which would allow “flexible thread scheduling that is defined in user-level code” (Shenoy). However, the scarcity of such sources leads one to believe that user-level threads were not implemented in MS-DOS, especially in the earlier versions of the disk operating system. As such, MS-DOS differs from the thread management discussed in class due to the fact that it neither has kernel-level threads nor user-level threads which most modern operating systems support.

One way to get around MS-DOS’s design for a single-user machine environment was through the use of Terminate-And-Stay-Resident programs, known as TSRs. Normally when a program runs on MS-DOS, the memory of that program is freed after execution so that other programs may be executed. This means that the program has to be “reloaded from disk back into memory to be used again” (Indiana University). A TSR can get around this situation by using a system call after it is finished executing. This system call returns control back to the DOS, but the TSR remains in the computer’s memory for later use. Other programs can be run while the TSR sits inside memory, or the programs can have the TSR run. This was a sneaky way to partially overcome DOS’s single-program execution and multitask, that is, be able to perform several tasks at once (Indiana University) .

Though MS-DOS did not support multithreading, Microsoft attempted to launch “MS-DOS 4.00” which was marketed as a “multitasking version of MS-DOS” (WinWorld). This new version of MS-DOS was said to include “preemptive multitasking, shared memory, semaphores, [and] advanced memory management,” (Multitasking MS-DOS 4.00) however there is little information on the specific implementation of the above additions, and if there would be successful synchronization given these new features. The release of the multitasking MS-DOS 4.00 was met with disappointment from users, and it was never used in North America. In addition to the limited release of this version, the new update was limited by a real-mode 8086 environment, which was a memory-addressing scheme used by x86 processors. It was also said to have multiple bugs and errors, and was met with overall dissatisfaction. IBM ultimately decided to not pursue MS-DOS 4.00, and they later released a different DOS 4.0 in 1988.

Since MS-DOS did not feature multitasking, and did not have to synchronize threads, it did not have a specific scheduling policy, and it runs programs sequentially as they are loaded into memory from disk. In MS-DOS, memory was organized in several layers. The MS-DOS was responsible for managing up to 1 MB of memory, with the lower 640KB reserved for specifically running programs. The 640KB of RAM is known as “conventional memory.” Above the conventional memory is what is known as the “upper memory,” ranging from 640 KB to 1024 KB, and above that sits extended memory. The upper memory, sometimes referred to as reserved memory, contains various “extra” components, some of which being ROM hardware drivers, video refresh buffers, and add-in cards (Duncan).

The conventional memory controlled by MS-DOS is divided into two sections: operating system memory, and transient program memory (TPA). The OS memory sits in the lowest portion of the conventional memory, and contains the interrupt vector table. This is the part of the memory that the CPU needs for system operation (Crow). Above the OS memory sits the transient program memory, which is memory pertaining to running programs–it has the blocks of memory that are allocated and released when programs are executed. The TPA, or memory arena, is “dynamically allocated in blocks called arena entries,” and each entry has a “control structure called an arena header, and all of the arena headers are chained together” (Duncan). The TPA has three functions that it uses in order to manage memory for a specific program: allocate, release, and resize. When a program needed to be loaded into memory from disk, the MS-DOS program loaded first called a function to allocate memory for that program. This function, known as “48H,” provided memory for the program itself, and for the loaded program’s environment (Duncan). After memory has been allocated, the program is read from disk into its assignment memory slot. The release function, known as “49H,” releases all blocks of memory held by the program after the program terminates and finishes execution. The third function, 4AH, is for resizing memory that was already allocated.

The ways in which memory is allocated differ based on which file type is being loaded in from disk. The two most common file types loaded in are .COM and .EXE, where .COM files are simple, with no headers, and are used for executing commands and instructions, while .EXE files have headers and certain metadata. For .COM files, the “ largest available memory block in the TPA” is automatically allocated, while for .EXE files, the two fields MIN_ALLOC and MAX_ALLOC are used to determine memory allocation. The MAX_ALLOC field defines the maximum amount of additional memory the program needs, if it were available (Duncan), and this is first attempted to be given to the program. If this amount of memory space is not available, then the total memory space is given to the program, provided that it does not dip below the requirements set by the MIN_ALLOC field. The resize function comes in at this point, and is used to resize any memory that is not needed immediately.

After memory blocks have been allocated to a program, there are two common operations that may occur, the first of which is to reduce the memory initially allocated. This is done so that there can be “enough room to load and execute another program under [the] control” of MS-DOS (Duncan). The second common operation would be to provide additional memory to the program, if, for example, the program needs additional space for “an array of intermediate results. (Duncan). If this is the case, the MS-DOS would then search through the chain of arena headers to see if there is enough memory anywhere that can be doled out. The first fit, best fit, and last fit strategies are used to select memory space. The default strategy used is first fit, where the arena entry selected is the one at the lowest address that has enough space to satisfy the request.

In the late 1980s, companies including Microsoft came up with a way to get around the 1MB limit of conventional memory that the MS-DOS could use, which became known as expanded memory. This was a way to provide programmers “large, additional amounts of fast RAM” (Crow) for applications, the most widely used one was developed by COMDEX, Lotus, and Intel, called Expanded Memory Specification. Programs that wished to use this additional memory would call an Expanded Memory Manager in order to “reserve pieces [...] [of] memory,” and from there could “request that a reserved page be mapped, release a reserved page,” (Crow) and so on. This was implemented using a mapping technique, where ”hardware was used to map the expanded memory to a piece of address space” (Indiana University). The use of Expanded Memory Specification, although popular for DOS, declined after users switched to a more user-friendly operating system.

Memory management in MS-DOS was widely different from the memory management techniques touched upon during lecture. Virtual memory was not supported by MS-DOS, and modern memory management strategies such as paging were not supported in early versions of the disk operating system.

MS-DOS was an influential operating system that reached peak popularity with the arrival and wide-spread use of personal computers. Though it may seem limited in its capabilities in comparison to modern operating systems used today, it was very useful during its time. Although MS-DOS lacked thread support and robust memory management techniques, it paved the way for modern operating systems widely used today, and was a foundational building block for tools such as shell. Learning about MS-DOS, and delving further to look at other historical achievements in computer science, makes one truly appreciate the past technologies that paved the way for current advancements in operating systems.

Bibliography

Crow, Jerry. “The MS-DOS Memory Environment.” ACM SIGICE Bulletin, vol. 21, no. 3, 1996, pp. 2–16., https://doi.org/10.1145/226036.226037.

Duncan, Ray. Advanced MS-DOS Programming: The Microsoft(R) Guide for Assembly Language and C Programmers. Microsoft Press, 1986.

“Indiana University.” On a PC, What Is the Difference between Extended and Expanded Memory?, https://kb.iu.edu/d/aalu#:~:text=The%20expanded%20memory%20requires%20hardware,it%20to%20the%20lower%201MB.

“Indiana University.” What Is Terminate and Stay Resident?, https://kb.iu.edu/d/afdi.

“Multitasking MS-DOS 4.00 4.00.” *WinWorld*, https://winworldpc.com/product/multitasking-ms-dos-/400.

Multitasking MS-DOS 4.00, https://www.pcjs.org/software/pcx86/sys/dos/microsoft/4.0M/.

Shenoy, Prashant. “CMPSCI 377 Operating Systems, Lecture 6.” 20th Sept. 2013, University of Massachusetts Amherst. Lecture.

Varma, Jayant. Learn Lua for IOS Game Development. Apress, 2013.