CP/M: A Family of 8- and 16-Bit Operating Systems

Dr Gary Kildall
Digital Research
POB 579
Pacific Grove CA 93950

This article is about microprocessors and CP/M: where they came from, what they are, and what they're going to be. Where they came from is history, what they are today is fact, and what they will become is, like any projection of technology, pure "science fiction" speculation. CP/M is an operating system developed for microcomputers. But as microprocessors changed, CP/M and its related programming tools evolved into a family of portable operating systems, languages, and applications packages.

The value of computer resources has changed dramatically with the introduction of microprocessors. Three major events have precipitated a revolution in computing: hand-threaded core memory has been replaced by mass-produced semiconductor memory; microprocessors have become plentiful; and IBM decided that the punched card is obsolete. Low-cost memory and processors have reduced the cost of computer systems to a few hundred dollars, but IBM's specification of the floppy disk standard has made the small computer system useful.

In the early days of the 8080 microprocessor, a small company called Shugart Associates was taking shape up the street from Intel. Shugart Associates, along with a number of other companies, viewed the floppy disk as more than a punched card replacement: at that time the primary low-cost storage medium was paper tape (used in applications ranging from program development to word processing). At a cost of $5, a floppy disk held as much data as two hundred feet of paper tape, and a disk drive retailed for only $500—an unbeatable combination. Memory, processor, and floppy-disk technology improved, and by the mid-1970s, a floppy-based computer could be purchased for about one quarter of a programmer's annual salary. Quite simply, it was no longer necessary to share computer resources.

Since that time, microprocessors have been applied to a variety of computing needs beyond replacement of low-end minicomputers. Due to applications such as machine-tool movement and sensing, data acquisition, and communications, current interest lies in real-time control. In a real-time operating system, process management can be separated from the I/O (input/output) system (which is not required in many applications). Real-time facilities allow the execution of interactive processes according to priority, and their addition or deletion in a simple fashion. This results in a custom operating system designed to solve a particular problem. In contrast to timesharing, real-time operating systems have minimal "interrupt windows" in which external interrupts are disabled. Real-time operating systems such as the Intel RMX and National Starplex packages provide this level of support.

The 16-bit version of CP/M is basically the same as the 8-bit version, with the addition of memory management and enhancements to the file system.

The emerging interest in *local networks* poses a new challenge to designers of operating systems. Recently, Intel, DEC (Digital Equipment Corporation), and Xerox formed an alliance to promote Ethernet, a *packet-switching* network intended to provide point-to-point data transfer in an office environment. (In a packet-switching network, data from several slow-speed sources, such as user terminals, is collected over local lines by a single network node, which then periodically transmits the data to its destination at a much higher speed, in groups called packets.) In terms of evolution and potential, Ethernet is today what floppy disks were a decade ago. This inexpensive office network performs such tasks as the transfer of a form letter from data storage at one location to a memory typewriter in another part of the

building. When modifications are completed, the letter is typed locally or sent to a laser (or other) printer that is a shared network resource.

Most timesharing systems handle a network through simple file transfers between the machines (*nodes*) in the net, but real refinements occur when the operating system itself is distributed among the nodes. File access is provided by one *server* node, while a computing function is performed by another. To the user, a *requester* node appears as a powerful computing facility, even though it may consist of only a local microprocessor, a console, and a limited amount of memory.

What refinements have been made to operating systems? Our models have been simplified; we understand primitive operations required for reliable process synchronization in real-time systems, and the human-oriented interface in interactive subsystems has been improved. We will, no doubt, continue to refine our models for timesharing and real-time operating systems, but the most exciting new operating system technology will develop around emerging network hardware.

Application Languages

Application languages form the top level of support for application programming. How does this level of language differ from other language levels? First and foremost, an application language contains the operations and data types suitable for expressing programs in a particular problem environment. FORTRAN (*FORmula TRANslation*), for example, was designed in the late 1950s for scientific applications; FORTRAN programs, therefore, consist primarily of algebraic expressions operating upon binary floating-point numbers expressed in scientific notation. However, FORTRAN contains only primitive file-access facilities and no decimal arithmetic, making it unsuitable for commercial data processing. COBOL (*COmmon Business Oriented Language*) has the commercial facilities, but it excludes scientific features such as a complete transcendental-function library.

In contrast to system languages that run on a given machine, these application languages would ideally contain no machine-dependent features. An application language is either poorly designed or ill-suited for a particular problem if the programmer is forced to use extra-lingual constructs to access lower-level functions of the operating system or machine. The language must be a standard, without the necessity for various locally defined language extensions. An extended standard language is of limited value since the extensions are unlikely to exist in other implementations.

The evolution of PL/I (*Programming Language/One*) provides a good example of refinement in application languages. PL/I is not a new invention: rather, it was defined by a committee of IBM users in 1960 as a combination of ALGOL (*ALGOrithmic Language*), FORTRAN, and COBOL, with a liberal sprinkling of new facilities. ALGOL's principal contribution was block structure and nested constructs, while FORTRAN contributed scientific processing and COBOL added commercial facilities. This combination produced a large, unwieldy language with twists and nuances that can trap the unwary programmer. Nevertheless, PL/I was quite comprehensive, and it served as the basis for uncounted numbers of application programs on large systems. One noted use of PL/I was in the implementation of the Multics operating system at MIT under Project MAC.

In 1976, an ANSI (American National Standards Institute) committee produced a standard language definition for PL/I. The standard is an implementation guide for compiler writers, and it precisely defines the form and function of each PL/I statement. Aware that PL/I was too large and complicated, the committee produced a smaller version for minicomputers, called Subset G. This new language excluded the redundancies and pitfalls of full PL/I but retained the useful application programming features. Recently approved by ANSI, Subset G has given new life to PL/I, with manufacturer support for the Data General Eclipse and MV/8000 computers, Prime computers, Wang machines, and DEC's popular VAX computer.

Strangely, the refinements found in application languages follow those of hardware and operating systems. Large, cumbersome languages have been rejected in favor of simple, Spartan programming systems that are consistent in their design. The resulting languages are easier to implement, simpler to comprehend, and allow straightforward program composition.

PL/M: The Base for CP/M

In 1972, MAA (Microcomputer Applications Associates), the predecessor of Digital Research, consulted with the small, aspiring microprocessor division of a semiconductor memory company called Intel Corporation. MAA defined and implemented a new systems-programming language, called PL/M (*Programming Language for Microcomputers*), to replace assembly-language programming for Intel's 8-bit microprocessor. PL/M is a refinement of the XPL compiler-writing language which is, in turn, a language with elements from Burroughs Corporation's ALGOL and the full set of PL/I.

The first substantial program written by MAA using PL/M was a paper-tape editor for the 8008 microprocessor, which later became the CP/M program editor, called ED. PL/M is a commercial success for Intel Corporation and, although licensing policies have limited its general accessibility, it has become the standard language of the Intel microprocessor world, with implementations for the 8080, 8085, and 8086 families.

MAA also proposed a companion operating system, called CP/M (*Control Program for Microcomputers*), which would form the basis for resident PL/M programming. The need

for CP/M was obvious: 8080-based computers with 16 K bytes of main memory could be combined with Shugart's new (at that time) floppy-disk drives to serve as development systems. For the first time, it was *feasible* to dedicate a reasonably powerful computer to the support of a single engineer. But the use of PL/M on larger timesharing computers was considered sufficient, and the CP/M idea was rejected.

The CP/M Family

CP/M was, however, completed by MAA in 1974. It included a single-user file system designed to eliminate data loss in all but the most unlikely situations, and used recoverable directory information to determine storage allocation rather than a traditional linked-list organization. The simplicity and reliability of the file system was an important key to the success of CP/M: file access to relatively slow floppy disks was immediate, and disks could be changed without losing files or mixing data records. And because CP/M is a Spartan system, today's increased storage-media transfer rates simply improve overall response. The refinements found in CP/M are based on its simplicity, reliability, and a proper match with limited-resource computers.

By the mid-1970s, CP/M added a new philosophy to operating system design. CP/M had been implemented on several computer systems, each having a different hardware interface. To accommodate these varying hardware environments, CP/M was decomposed into two parts: the invariant disk operating system written in PL/M, and a small variant portion written in assembly language. This separation allowed computer suppliers and end users to adapt their own physical I/O drivers to the standard CP/M product.

Hard-disk technology added yet another factor. CP/M customers required support for disk drives ranging from single 5-inch floppy disks to high-capacity Winchester disk drives. In response, CP/M was totally redesigned in 1979 to become *table-driven*. All disk-dependent parameters were moved from the invariant disk operating system to tables in the variant portion, to be filled in by the system implementer.

CP/M is now a multifunction program whose exact operation is defined externally through tables and I/O subroutines. The widespread use of CP/M is directly attributed to this generality: CP/M becomes a special-purpose operating system when it is field-programmed to match an operating environment. Through the efforts of system implementers who provide this field-programming, CP/M is used worldwide in close to 200,000 installations with over 3000 different hardware configurations.

CP/M, PL/I, and PL/M have all played a role in the development of CP/M-86.

MP/M

As single-user CP/M became widely accepted, Digital Research began to develop a new operating system for real-time processing. The design called for a real-time nucleus to support cooperating sequential processes, including a CP/M-compatible file manager with terminal-handling capabilities. This operating system, called MP/M (*Multiprogramming Monitor for Microcomputers*), is a further refinement of the process model found in Intel's RMX and National's Starplex. As a side effect, the combination of MP/M's real-time nucleus with the terminal handler and the CP/M file system produces a traditional timesharing system with multiprogramming and multiterminal features.

Timesharing allows programs to execute in increments of processor time in a "lock-step" fashion. In a timesharing context, a printer program, often called a *spooler*, might have the task of printing a series of disk files which result from program output. The spooler starts with a disk-file name and, by using increments of processor time allocated by the real-time nucleus, writes each line from the file to the printer. Upon completion, the spooler obtains another disk-file name and repeats the process. You can, for example, send the name of a disk file to the spooler and, while the file is being printed, edit another file in preparation for compilation. The spooler and editor share processor time to complete their respective tasks. In general, many such processes share processor time and system resources.

MP/M process communication is performed through *queues* (or waiting lines) managed by the nucleus. The spooler, for example, reads file names from an input queue posted by another process (which reads spooler command lines from the console). When the spooler is busy printing a file, additional file names may enter the input queue in a first-in first-out order.

Process synchronization through queuing mechanisms is commonplace, but MP/M treats queues in a unique manner, simplifying their use and decreasing queue management overhead. Queues are treated as files: they are named symbolically so that a queue can be added dynamically. Like files, queues have queue control blocks that are created, opened, deleted, written, and read. In fact, the set of queue operations closely matches the file functions of CP/M so that MP/M provides a familiar programming environment.

The implementation of queues is transparent to an operator or system programmer, but it is important to MP/M's effective operation on limited-resource computers. Queues are implemented through three different data structures, depending upon the message length. So-called "counting semaphores" count the occurrence of an event with message length zero, and are implemented as 16-bit tallies. Single-byte messages are processed using a circular buffer. Similarly, queues containing addresses are processed using circular buffers. In all

other cases, MP/M uses a general linked list, which requires additional space and processing time. It is this sensitivity to the capabilities of limited-resource computers that makes MP/M effective: while real-time operating systems often incur 25 to 40% overhead, MP/M has been streamlined to increase available compute time by 7% over single-user CP/M.

Like CP/M, MP/M is separated into variant and invariant portions. The file-system interface is identical to that of CP/M, with the addition of user-defined functions to handle non-CP/M operations (such as control of the real-time clock). Field-reconfiguration of MP/M allows a variety of device protocols including CP/M-style busy-wait loops, polled devices, and interrupt-driven peripherals. In fact, the variety of interface possibilities makes the MP/M implementer a true system-software designer, since a fine-tuned MP/M system may operate considerably faster than its initial implementation.

What are the refinements found in MP/M? First, it is a state-of-the-art operating system based on current process-synchronization technology and microprocessor real-time system design philosophies. Process communication is conceptually simple and requires minimal overhead. Finally, it is the only operating system of its type that can be field-tailored to match almost any computer configuration.

CP/NET

CP/NET, introduced in late 1980, leads a series of network-oriented operating systems that distribute operating system functions throughout a network of nonhomogeneous processors. CP/NET connects CP/M requesters to MP/M servers through the use of an arbitrary network protocol. Similar to CP/M and MP/M, CP/NET consists of the invariant portion, along with a set of field-reconfigurable subroutines that define the interface to a particular network. For purposes of CP/NET, this interface need only provide point-to-point data-packet transmission. Since the actual data transmission media are unimportant to CP/NET, any one of the number of standard protocols can be used, from low-speed RS-232C through high-speed Ethernet. Physical connections are also arbitrary, allowing active hub-star, ring, and common-bus architectures.

The invariant portions of CP/NET operate under a standard CP/M system to direct various system calls over the network to an MP/M server. The MP/M server, in turn, responds to network requests by simulating the actions of CP/M. This simulation is transparent to an application program: any program operating under standard CP/M operates properly in the network environment.

Suppose, for example, you wish to store common business letters in a central data base under MP/M and access these letters from a CP/M-based word processor. You begin by assigning one local disk drive to the MP/M master, using the CP/NET interface. You then direct your word processing system to read the particular letter on the assigned drive, causing the data to be obtained from the server rather than from the local disk. After local update using your word processor, you can print the result on your local printer or optionally assign your listing device to the network for printing at the MP/M server.

CP/NET is accompanied by three related network operating systems: CP/NOS, MP/NET, and MP/NOS. CP/NOS is, in effect, a diskless CP/M, which can be stored in read-only memory, and that operates with a console, memory, and network interface. MP/NET, on the other hand, is a complete MP/M system with an embedded network interface that, like CP/NET, allows local devices to be reassigned to the network. MP/NET configurations allow MP/M systems as both requesters and servers with CP/M requesters. Finally, MP/NOS contains the real-time portion of MP/M without local disk facilities. Like CP/NOS, MP/NOS performs all disk functions through the network.

The interface protocol is publicly defined so that non-MP/M or non-CP/M systems can participate in network interactions. A server interface for the VAX 11/780, for example, is under preparation so that it can perform I/O functions for a large number of MP/M and CP/M requesters.

The principal advantage of CP/NET is that all CP/M-compatible software becomes immediately available for operation in the network environment, solving the problem that builders of network hardware face: the total absence of application software. Although the promise is there, networking is in its infancy, and CP/NET is truly a software package awaiting the evolution of suitable hardware.

PL/I: The Application Language

In 1978, Digital Research investigated the final level of software support: application languages. One such language was to be supported throughout the operating system product line, and the choice would have to be a multipurpose language. Further, the language would have to be an international standard to promote the generation of software by independent vendors. Standard Pascal seemed a logical choice but was rejected for several reasons. First, Pascal is an ALGOL derivative with scientific orientation. Commercial facilities in the standard language are absent: decimal arithmetic, file processing, string operations, and error-exception handling were essential. Further, separate compilation and initialization of tables were not in the language. There was a temptation to extend Pascal in order to include these features, but these extensions would have defeated the benefits of standardization.

PL/I Subset G was the obvious choice. It satisfied scientific and commercial needs and, because of subset restrictions, was consistent and easy to use. The project was a bit daring, however, because Subset G was unknown in the computer community. PL/I was viewed as a large IBM-oriented language with huge, inefficient compilers that required tremendous runtime support.

The Digital Research implementation of Subset G was started in mid-1978 and completed two years later. The compiler is a three-pass system written in PL/M. The first two passes are machine independent and produce symbol tables and intermediate language suitable for any target machine. The third pass is largely machine dependent and is dedicated to code optimization and final machine-code production. The compiler is accompanied by a linkage editor (compatible with the Microsoft format), a program librarian, a set of runtime subroutines, and a relocating macro assembler.

Thus, PL/I completes the final level of the inverted pyramid of support tools. The message should be clear to the application programmer: it is not the system language or the operating system which is important in the production of a final application. Rather, it is the availability of a standard, widely accepted application language that can provide program longevity. Once expressed in PL/I Subset G, the program can be transported through the CP/M family of operating systems to a variety of minicomputer systems. Digital Research has a long-term commitment to PL/I support for popular operating systems and processors.

New Processor Architectures

We've spent little time discussing processor refinements. What is happening to our software tools as we augment our 8-bit machines with the more powerful 16-bit processors? Will 16-bit processors replace 8-bit machines, or are they simply a temporary phenomenon in the transition to 32-bit machines?

There are several considerations when answering these questions. First, 8-bit machines are economical to produce, their software systems are mature, and they satisfy the needs of a substantial computer base. Therefore, we can safely assume that 8-bit machines are here to stay. Newer 16-bit machines are marginally faster, but they have substantially more address space. To use this addi-

tional address space, the computer must contain more memory, which increases the computer system cost.

As system costs increase, the margin between low-end minicomputers and high-end microcomputers diminishes, placing microcomputer hardware and software manufacturers such as ourselves in direct competition with major minicomputer manufacturers. The 16-bit machines, by their nature, introduce memory segmentation problems that are not present in 32-bit processors.

Finally, we should note that 16-bit minicomputers are already outmoded, and all serious manufacturers are pushing 32-bit machines. This leads to the following conclusion: if we are tracking the minicomputer world, we can assume that the future will be with the 32-bit processors.

Currently, however, 32-bit machines are not available in quantity. Even when they are available, there will be delays while manufacturers tool up for production. At the moment, the 16-bit processors offer an intermediate solution. Digital Research has provided initial support for Intel's 16-bit machines—iAPX-186 and iAPX-286—which are versions of its 8086 product line. Intel provided PL/M-86, rehosted from the 8080 line, which was used by Digital Research to generate CP/M-86 and MP/M-86. In both cases, the fundamental design remains basically the same as that of the 8-bit version, with the addition of memory management and enhancements to the file system that match new computing resources. A familiar program environment is retained so that program conversion is simplified.

CP/NET and related network software will be available sometime this year. Intel's 8087 (an arithmetic coprocessor for the 8086) is of particular interest since it directly supports binary and decimal operations, which substantially increase PL/M-86 execution speed.

In addition to the 8086, the CP/M family will be adapted to the 16-bit machines that prove popular, with special interest in the 32-bit architectures as they become available. During this development and rehosting, however, the 8-bit processors will continue to be supported with new tools and facilities, since this constitutes, without doubt, our best customer base for some time to come.

Software Vendors

We've concerned ourselves with three levels of software tools that support the most important level: the application programs. A major reason for CP/M's popularity is the general availability of good application software. At last count, there were about 500 commercially available CP/M-compatible software products.

Through the combined efforts of CP/M distributors, independent vendors, and CP/M users, we are participating in a software commodity market with quality and variety that is unequaled by any minicomputer or mainframe manufacturer. The large CP/M customer base allows a vendor to produce and support a software package at low end-user cost. This increases the customer base, drawing more vendors with lower-cost good-quality products. This cyclic effect is, today, solving the "software crunch."

The tools are available, and it is the responsibility of independent software vendors to continue developing their own specialized markets. In this way, computer software technology will reach virtually all application areas where low-cost, reliable computing is required. Refinements? My friend, they're up to you.■