R. J. Creasy

The Origin of the VM/370 Time-Sharing System

VM/370 is an operating system which provides its multiple users with seemingly separate and independent IBM System/370 computing systems. These virtual machines are simulated using IBM System/370 hardware and have its same architecture. In addition, VM/370 provides a single-user interactive system for personal computing and a computer network system for information interchange among interconnected machines. VM/370 evolved from an experimental operating system designed and built over fifteen years ago. This paper reviews the historical environment, design influences, and goals which shaped the original system.

Introduction

The Virtual Machine Facility/370, VM/370 for short, is a convenient name for three different operating systems: the Control Program (CP), the Conversational Monitor System (CMS), and the Remote Spooling and Communications Subsystem (RSCS). Together they form a general purpose tool for the delivery of the computing resources of the IBM System/370 machines to a wide variety of people and computers. The CP and CMS components evolved directly from an experimental operating system designed and built by the author and his group at the IBM Systems Research and Development Center in the mid-1960s. This center, now called the IBM Scientific Center, is located in Cambridge, Massachusetts.

CP is an operating system that uses a computing machine to simulate multiple copies of the machine itself. These copies, or virtual machines, are each controlled like the real machine by their own operating systems. CMS is an operating system that supports the interactive use of a computing machine by one person. It is the typical operating system used within each virtual machine. RSCS is the operating system used to provide information transfer among machines linked with communications facilities. These three systems, used together, produce a general purpose multiple access facility. Other operating systems can be used on each virtual machine as well and might be selected for batched job processing, for compatible interchange with other systems, or for other purposes.

In the spirit of reviewing the twenty-five years spanned by the *IBM Journal of Research and Development*, we take a moment to discuss the historical environment, design influences, and goals which formed the first ancestor of VM/370. The paper is historical and is not intended to be a critical look at operating system design. (That subject scarcely lacks coverage; see, for example, [1].) A few aspects of the CP/CMS design merit special attention, but mostly it is the particular selection, combination, and implementation of features which have proved useful. This retrospective look may provide some insight into the personality, capability, and potential of the VM/370 design.

Information can be found elsewhere to describe the current incarnation of VM/370 and related subjects. The manuals [2] provided for this IBM product cover concepts, user commands, and system information. More generally, virtual memory concepts [3], virtual machine concepts [4], and the virtual machine environment of VM/370 [5] have been discussed at length. The bibliographies of these references are very good.

An historical perspective

The roots of VM/370 are most deeply entwined with the style of use of the computing machines of the 1950s by scientists and engineers. In those days, the machines were used as personal tools, much like their predecessors which had been designed and dedicated to specific appli-

cations. Unlike the earlier machines, however, they did not sit idle when a problem was solved; their architecture was more general, with easily changed stored programs directing their actions on a variety of problems. These programs, hand crafted for the slow and costly hardware, required the entire machine. The users were normally present to make sure things were going well and, if not, reacted in real time to correct things or to collect information useful for diagnosis. The running time of a program was quite long compared to these user actions and to the time required to set up the machine for the next problem.

As machines became faster and program execution times shortened, the overhead of human operation became more significant. Simple job monitor systems were introduced to reduce the cost of the idle time between execution of different programs. These systems terminated a program in a preselected way, prepared the machine, and began execution of the next job. Machines continued to grow in capability and speed and to decline in cost per computing unit. With larger memories and independent I/O operation came the possibility of more efficient machine utilization. A portion of the machine could be dedicated to the programs which assisted in machine operation, the operating system. As a result, overall productivity of the system increased. Peripheral computers were used to prepare job streams for the main computer; they printed and punched batched output as well. The turnaround time of a job from submission to completion was measured in hours, sometimes days. The prescheduling of jobs resulted in more efficient machine utilization, but the users were more isolated from the machine. Developing and testing programs was a frustrating experience.

Thus the stage was set for the direct interactive use of machines by some users. The machines, occupied gainfully by the steady stream of batched work, could be interrupted now and then by people demanding a small amount of computer time. It was thought that the costs of such time-shared use could be reduced to the point where people, using typewriter-like equipment, could again command the machine to do their bidding. The pioneering work in such a general purpose time-sharing system was done at M.I.T. in the early 1960s [6]. With this system, called the Compatible Time-Sharing System (CTSS), a normal batched job stream was run as background to keep the computer busy. At the same time, several users could enter commands to prepare, execute, and terminate their programs. The machine directly responded to each of them in real-time. Programs, data, and textual material were created, modified, executed, formatted, and the like in a style similar to that produced today by VM/370, a style which is now common to many personal computers as well.

Batched job systems remained the main mode of computer operation and continued in their evolution. They could overlap the termination, operation, and setup of sequential jobs. Later, more complex systems would multiplex the execution of several programs in an effort to reduce the idle time of individual machine components. For all but the most specialized of uses, like airline scheduling or real-time control, these sophisticated batched job operating systems managed the machine well.

During this same period, time-sharing systems were designed to solve similar problems; these solutions recognized the on-line user as an important new factor. In addition, user interfaces, tools, and application programs were developed uniquely for interactive use. Techniques to support dynamic allocation of the machine and demand scheduling became more efficient and predictable. Most importantly, the machine and people costs steadily moved to favor interactive systems. Today's batch operating systems include modifications and extensions to support interactive use, although the style and capability presented to the user tend to reflect the heritage of the underlying system.

The future promises inexpensive and powerful machines that can individually serve each of us as assistants as a machine now stores, modifies, and formats this text under my command. A pattern of use, similar to that decades ago, is emerging with the use of personal computers as machines dedicated to a single user. The cost of these modern machines is so incredibly low compared with their progenitors that the idle time caused by human interaction or lack of work is not significant. But these machines will not be isolated in laboratories. They will be interconnected in many ways as the emphasis moves from computing convenience to information access.

Computing use has come full circle to repeat history in a variety of ways. The ingredients of dedicated machine use and interactive support, occurring separately in the past for most people, will combine with machine to machine communication to characterize these new systems. It is precisely because these features also characterize VM/370 to its users that this system design should be viewed in the modern context. VM/370, with its collection of interconnected, independent machines each serving one user, can provide architectural compatibility with the future's networks of personal computers.

Design influences and goals

The CP/CMS system was conceived in 1964 as a second-generation time-sharing system for the newly announced IBM System/360. It began as an experimental software

project [7, 8] designed to support all the activities of the Cambridge center, including such diverse activities as operating system research, application development, and report preparation by programmers, scientists, secretaries, and managers. Soon after its inception, it was convenient for the system to be recognized and financially supported from outside of the center as a tool to evaluate and test the performance of operating systems. It later gained acceptance as a time-sharing system after its installation at M.I.T.'s Lincoln Laboratory. Originally called a pseudo-machine time-sharing system, CP/CMS was named a virtual machine system from the description of similar but independent ideas [9]. It was to integrate traditional computer operations within a more general and interactive framework.

In 1966, CP-40 and CMS both became operational using an IBM System/360 Model 40 modified by the addition of an associative memory device for dynamic address translation [10, 11]. CP-40 was run only on this unique hardware at Cambridge. At about the same time, CP-67 was built to use the address translation feature of the newly announced System/360 Model 67. CP-67 and CMS were installed on many of these production machines. VM/370 became available in 1972 for the IBM System/370 computer family whose members all included virtual memory hardware. CP and CMS have seen continuous production use since 1967 with over 2500 systems now in operation. We now review some of the work which influenced the conception and design of the CP/CMS system.

- *Systems influence*

The early 1960s witnessed many concurrent projects in time-sharing system design. In addition to the first operational general purpose systems at M.I.T. and at Bolt, Beranek, and Newman, several of the early efforts mentioned in [12] took place at the Carnegie Institute of Technology, IBM Corporation, M.I.T., Rand Corporation, System Development Corporation, and the University of California at Berkeley. A wide variety of ideas and opinions covered every aspect of the sought-after interactive facilities, from the overall system architecture to the details of program interfaces, especially editors. Many days were spent discussing issues, such as character by character input processing, which today command just as much attention as then.

Recall that very little hardware was available for continuous and convenient human interaction with machines. Terminals were a problem to find and attach to a computer. Special purpose hardware was usually required, and primitive connections, such as driving a teletypewriter from a memory bit, were not unknown. In addition, the computers in those days did not provide many of the protection features commonplace today. Sharing a machine safely among programs and users provided many thorny problems. As one solution, most early time-sharing systems provided new or modified languages operating through interpreters or restricted execution contexts.

The Compatible Time-Sharing System [6, 12], first operational in 1961 and in production use at M.I.T. from 1964–1974, most strongly influenced the CP/CMS system design. CTSS, a first-generation time-sharing system, provided a subset of the machine for use by normal batch programs. The compatibility and protection techniques it used were simple and effective. The background programs were run without modification as with a normal system. The time-sharing supervisor would steal and restore the machine without their knowledge. This technique was extended to its fullest in CP/CMS. Many other CTSS design elements and system facilities, like the user interface, terminal control, disk file system, and attachment of other computers, served as operational prototypes.

The implementation of CTSS illustrated the necessity of modular design for system evolution. Although successful as a production system, the interconnections and dependencies of its supervisor design made extension and change difficult. A key concept of the CP/CMS design was the bifurcation of computer resource management and user support. In effect, the integrated design was split into CP and CMS. CP solved the problem of multiple use by providing separate computing environments at the machine instruction level for each user. CMS then provided single user service unencumbered by the problems of sharing, allocation, and protection. As an aside, the MULTICS system [13] of M.I.T.'s Project MAC and CP/CMS were both second-generation systems drawing heavily on the CTSS experience with very different architectural results.

Recursive systems, such as biological reproduction and to some extent the early LISP design [14], exhibit powerful, elegant concepts which influenced the CP/CMS system design. In reproduction, the cell is duplicated including the duplication mechanism. With LISP, several primitive functions act on a simple data structure to define an architecture within which all functions and data, both system and user, are represented. The duality of functional mechanisms and their representation as the data upon which they operate are illustrated by these systems and by CP/CMS.

- *Hardware influence*

The family concept of the IBM System/360, announced in spring 1964, was a most amazing turning point in comput-

er development, one which was not universally greeted with enthusiasm. We believed that the architecture of System/360, combining scientific and commercial instruction sets, would be around for a significant period of time. The trauma associated with widespread recoding of programs also pointed to a long life. In addition, we speculated that many operating systems and a large number of application programs would be produced over the lifetime of that machine design.

How could we design a single operating system to accommodate all of these programs in a smooth and compatible way? What interface and capabilities could be designed to support such a wide range of applications? Going back to basics turned up the answer: The instruction set, the essence of this new machine line as documented by the System/360 principles of operation manual, would be the interface to the time-sharing control program. Each user would have the complete capability of the System/360. Machine sharing and allocation would be accomplished below this level by the Control Program. This apparent replication of the machine for each user not only guaranteed compatibility but provided an avenue for future machine development whereby resource control could be incorporated into the machine architecture.

The design of System/360, in order to facilitate the multiplexed execution of several jobs in a scheduled job environment, provided two instruction execution states: privileged and problem. The instructions available in problem state are those commonly used by application programs. They are innocuous to other programs within the same machine and can be safely executed. However, privileged instructions affect the entire machine as well as report its status. As they are encountered in problem state, the machine blocks their execution and transfers control to a designated program. When using CP, each virtual machine program is actually executed in problem state. The effects of privileged instructions are reproduced by CP within the virtual machines.

The System/360 machine design made possible the reasonable implementation of multiple machine environments with only one major exception. There was no practical way to move programs within the memory after they had been prepared for execution. Yet experience with demand scheduled systems suggested the need for dynamic program relocation. The IBM 7094, modified for CTSS, contained a relocation register to offset memory addresses, but it was of little use because a program could only be moved as a unit. Some technique had to be found to break programs into pieces which could be moved into, out of, and within the memory independently of each other.

- *Virtual memory influence*

In the late 1950s, work at the University of Manchester led to the Ferranti Atlas machine design, which provided automatic extension of main memory using drum storage [15]. Motivated by the high cost of fast memory and by the inconvenience of machine-specific two-level store manipulation, it was to spawn virtual memory. The memory was split into fixed length page frames which were used, as required, to execute a program stored in the larger address space of the drum. This idea was extended [16] and incorporated into the 1964 machine design proposal of M.I.T.'s Project MAC for the MULTICS system. Multiple memory segments, each like a single memory broken into pages, formed the program address space. The same idea was developed [17] and implemented by IBM in an experimental system called the M44/44X, first operational in 1965. The memory architecture of the System/360 Model 67 and the System/370 family follow the same design trend.

The hardware modification to the IBM System/360 Model 40 used for the CP/CMS project, most similar to the Atlas design, provided for relocation and protection. The main memory was split into a fixed number of pages. During program execution, each memory address and an active user number were compared, using an associative array, to the list of virtual page addresses present in real memory. The position of the matching entry, if it existed, was encoded into a physical page address. No degradation of the memory cycle was required for this action. If no match was found, a page fault was indicated, resulting in a machine interrupt. The memory structure provided by segments and pages, key to the MULTICS design for example, was not required by the CP/CMS design. A difficulty of virtual memory, solved by programming and some restriction, was the control of System/360 I/O channels—processing units operating independently of the dynamic address translation hardware.

- *Design goals*

The early time-sharing systems supported input, edit, and output of programs and data; most supported programs in machine code, like the output of the FORTRAN compiler; some only interpreted programs in source language form; but no system could safely execute machine code which like itself manipulated all features of the system. The ability of these systems to support the full interactive cycle of program refinement, test operation, production use, and program enhancement for a wide range of applications was not extended to the operating system itself. System developers could only reap the benefit of computer assisted programming up to the point of actually trying the code, at which time they required a dedicated hardware system, usually at night or on weekends. We

wanted all hardware function delivered to the user, if desired. In addition, the system was designed for continuous operation. CP/CMS was the first system to extend availability to all users.

The necessity of compatibility for evolutionary growth of software was demonstrated by CTSS; for hardware, by the IBM System/360 family. The ability to accomplish steady growth, accommodate change, and provide smooth access to past knowledge is a requirement for any successful system architecture in the future. This includes the ability to utilize production batch, real-time, and other specialized systems as the hardware speed and cost allow. It also includes the capability to incorporate more sophisticated hardware construction techniques. For example, a piece of CP might be implemented within hardware to extend machine capability. Users of virtual machines and their operating systems would see no change in this function except possibly in cost or speed. Conversely, CP might simulate a new hardware feature to be tested. Performance might be poor, but programs using this feature could be run. This was done within IBM to prepare System/370 programs using a System/360.

A virtual machine cannot be compromised by the operation of any other virtual machine. It provides a private, secure, and reliable computing environment for its users, distinct and isolated like today's personal computer. Experiments by one user present no problems to a payroll job on another virtual machine. New facilities, such as a data base system or specialized device support, can be added without modification or disruption of current capabilities. Modular functionality is strongly fixed in the CP/CMS design. Each virtual machine system can provide an important service: by CMS, for execution of user programs; by RSCS, for a store and forward computer networking system. Many important areas remain to be considered. Those capabilities necessary within CP to support virtual machine operation can be carefully chosen and implemented; unessential function can be moved into a virtual machine.

One of our important design goals was the production of a virtual machine identical to its real counterpart. We expected movement of systems among real and virtual machines. For example, system device addresses, fixed on real machines, were easily changed on virtual machines. A program tailored for a real machine could be accommodated simply on a virtual machine. This level of indirection also made the virtual machines insensitive to many changes in the actual hardware. Some devices can even be simulated efficiently. It is tempting to produce features unique to the virtual machine. But any schism between virtual and real machine capabilities can only limit future options. Features available within the virtual domain can be evaluated as extensions to the machine architecture, then incorporated or eliminated. CMS originally operated on real hardware but that is no longer possible without recoding.

The design of CP/CMS by a small and varied software research and development group for its own use and support was, in retrospect, a very important consideration. It was to provide a system for the new IBM System/360 hardware. It was for experimenting with time-sharing system design. It was not part of a formal product development. Schedules and budgets, plans and performance goals did not have to be met. It drew heavily on past experience. New features were not suggested before old ones were completed or understood. It was not supposed to be all things to all people. We did what we thought was best within reasonable bounds. We also expected to redo the system at least once after we got it going. For most of the group, it was meant to be a learning experience. Efficiency was specifically excluded as a software design goal, although it was always considered. We did not know if the system would be of practical use to us, let alone anyone else. In January 1965, after starting work on the system, it became apparent from presentations to outside groups that the system would be controversial. This is still true today.

The control program

A program coded for a computing machine can produce results in either of two ways. It can be loaded into the machine and executed. Or a person can interpret the program, step by step, using the architecture manual as a guide. In both cases the results should be the same. (Actually, results may differ because timing relations among machine components or error conditions may not be precisely defined.) In the first case the program is being executed on a real machine; in the second case, we can say the program is being executed on a virtual machine. A computer can be programmed to replace the person in this interpretation process. The Control Program is an operating system which uses this technique to determine the operation of virtual machines.

The Control Program is a general multiprogramming system that uses virtual machines to organize independent job streams. A portion of the machine controlled by CP is needed to support the allocation and management of the rest of the machine. The remainder is available for the virtual machines. The techniques used by CP to share machine components range from the simple to the sophisticated. For example, a magnetic tape drive is attached, if not in use, to a virtual machine with a simple CP command. It is dedicated to that machine until detached.

Each virtual machine is identified by an entry in the CP directory. This entry contains, among other things, the definition of permanent virtual disks mapped to specified disk volumes. Temporary disks, available to an active machine for a session, are dynamically defined by a CP command from space set aside for that purpose. The simulation of a virtual card reader and punch or a line printer is more difficult. All information to and from these devices is stored by CP in its spool system. It is labeled with the virtual machine identification and associated with its correct virtual component. CP operates the real equipment consistent with the user expectation of a dedicated resource. Control of the memory, CPU, and I/O channels requires more sophisticated methods. Management of the machine resources to achieve various response and efficiency profiles for each virtual machine is the most difficult problem to solve. This is because of the wide variety of programs accommodated by CP, ranging from conversational interactions to day-long computations. The constantly varying workload blurs the traditional distinction between batch and time-shared operation.

All work done by the machine under control of CP is the result of virtual machine action. There is no other facility, like a background job stream, to be used in place of a virtual machine. Any virtual machine defined in the CP directory can be activated by a person using a keyboard and display terminal. The terminal is then used as the control console of a virtual machine. Commands to CP take the place of switches and buttons used to start and stop the CPU, display and store memory contents, trace execution, etc. It is also possible to command CP to create or delete hardware components, to change the machine configuration, or to perform other services, like the selection of printer paper. Commands to CP can be thought of as requests to a machine operator of extraordinary capability. In addition to console function mode, the user's terminal can be used by the program being executed in the virtual machine. This is the normal method of communication between the user and application programs. But another type of terminal connection is possible. In this case, a virtual machine is not activated by the user. Instead, with a CP command, the terminal is attached to a selected machine that is already active. The terminal is then under exclusive control of the program in that machine. A multi-user information management system might be such a program.

The System/370 virtual machine can be in basic or extended control mode. Basic control does not include the virtual memory hardware. This type of machine is used by CMS and other operating systems that do not themselves utilize the address translation hardware. Extended control mode is selected when an operating system that controls virtual memory, such as CP or OS/VS, is executed in a virtual machine. This might be the case when a new version of CP is to be tested. At many installations, a batch operating system is run in a virtual machine to be compatible with traditional procedures, application programs, or other locations. In fact, at a few installations, CP is used only to concurrently operate production and test versions of the same batch operating system. Virtual machines are not provided for personal computing, at least not outside of the operations group!

A key feature of the CP/CMS design is the independence of each virtual machine. All connections between virtual machines are explicit because they are specified and controlled by CP, whether via shared memory or disks, I/O channels, unit record media, or telecommunications lines. There are no hidden dependencies or relationships between systems in separate virtual machines. If these machine interconnections are within the domain of the computer architecture, evolutionary growth from virtual to real systems is possible. As larger, faster, and less expensive machines become available, the software systems supporting interconnected virtual machines can move smoothly to collections of real machines.

Any computer, real or virtual, without software is of use only to computer students. This is the case with a newly activated virtual machine. Of course, instructions can be stored in the memory, one by one, using the terminal as the machine console, and then executed. No operating system is needed. This is impractical but certainly familiar to those with today's microcomputers or yesteryear's memories. As with a personal computer, a choice of many operating systems may be available to the user. A most important characteristic of the Control Program is to give each user the choice of any software, whether backlevel, standard, modified, or home-brewed. Normally the machine console is first used to specify an initial program load operation. The program so loaded is usually an operating system, probably CMS, that will operate the computer in a manner convenient to the user.

Conversational monitor system

In the beginning, the initials CMS had various meanings: Console Monitor System, Cambridge Monitor System, and so on. By any name, CMS is a disk-file-oriented operating system to support the personal use of a dedicated computer. It is a single user system providing function in the style of CTSS. CMS began operation on a real IBM System/360 serving the user at the system console. When CP became operational, CMS moved from the real to the virtual hardware.

The heart of CMS is the file system, which manages permanent information stored on disk. The user of the file system, via CMS services, sees a collection of named files, grouped by disk. Each file contains records of fixed or variable length. Any record can be accessed, but additional records must be appended to the file. Each disk managed by the file system contains a set of files, all listed in a single-level directory. These files and their directory are completely resident on one disk and are independent of the files on other disks. The file system automatically manages the physical organization of the data. It stores all files in fixed length data blocks, maintains pointers, and allocates space invisible to the file system user, who only uses disk name, file name, and record number to access data. Each disk is managed by its user, who must create, archive, and erase files as necessary.

Sharing of data among users is accomplished outside of CMS by connections established by CP between virtual machines. Disks shared among machines provide the usual means of accessing common information. Typically, several disks, in addition to each user's private disk, contain programs, data, and the CMS operating system. This structure of multiple disks, each with a single directory, was chosen to be simple but useful. Multi-level linked directories, with files stored in common areas, had been the design trend when we began. We simplified the design of this and other components of CMS to reduce implementation complexity. We had to produce a system we could use to produce more sophisticated systems. We did not have the people and time to solve the problems associated with more ambitious features. In addition to disk sharing for multiple user file access, copies of data files are frequently transmitted between users via simulated unit record equipment. This method generalizes nicely for users of connected collections of machines and is discussed in the RSCS section.

The CMS nucleus resides in low memory. Although it is key-protected to prevent accidental destruction of important data like file directories, user programs have the run of the machine and can modify the system or use privileged instructions. Of course the integrity of the other machines is not compromised. Programs are coded in some source language, processed to produce machine instructions, and loaded into memory. At that point, they can be executed or stored on disk as a memory image, called a module, ready for fast loading in the future.

The CMS command processor obtains input from three sources: that typed by the user at the machine console, that stored in an input stack, or that contained in a disk file used by the EXEC processor. This input, via the file system, selects a file to produce further input or to load and execute as a program module. When there is nothing left to do, CMS waits for more input from the user. Because all commands available to the user are stored as disk files and not built into the system, it is convenient, in an open-ended fashion, for a user to replace, change, or add commands.

CMS was developed to support a particular type and style of work. It was designed to support its own development and maintenance. It is used to maintain the other components of VM/370 as well. Familiarity with previous work, as a designer and user, provided criteria to select or reject features based on ease of implementation, generality, and utility. In most cases, a subset of features was selected with the expectation of future work. We expected many operating systems to flourish in the virtual machine environment. What better place for experimentation with new system ideas? This has not been the case. Instead, many features were added to CMS to extend its usage into areas better served by new systems. A notable exception is RSCS, a system designed to control network communication from a dedicated machine.

Remote spooling and communications system

RSCS is an interrupt-driven, multi-tasking system that uses a dedicated computer with attached communications equipment for the support of data file transfer among computers and remote work stations. This system, developed after CP and CMS, has been described by its authors [18] in a way which illuminates sound concepts in virtual machine operating system design. RSCS is a paradigm which provides user service in the spirit of the CP/CMS concept. Operating in a virtual machine, it uses unit record equipment and communication links to send and receive files, in a store and forward fashion, to and from virtual machines, real machines, and remote work stations.

Within each VM/370 installation are many virtual machines identified by unique names defined within the CP directory. From a network viewpoint, CP appears as the central node of a star network with a virtual machine at each point. Information flows from a machine at the point, through CP at the center, and then back out to a point machine. So a file can be passed to someone by simply specifying the receiving machine's name and transmitting the file. This communication technique is only useful within one real system because the files are transmitted using each virtual machine's card reader and punch; it is impractical between real machines.

RSCS operates in a virtual machine which has attached telecommunications lines, channel to channel connections, and other computer communication devices. It

accepts files sent to it by other users of the VM/370 system or by another computer. Using a unique identifier for each computer system, RSCS then selects a communication link and sends the file to its counterpart on the next system, which repeats this process until the file reaches the destination. The two-tier address structure, system and user, used by RSCS currently addresses within IBM an estimated 50 000 users of more than 400 systems spanning five continents.

Summary

VM/370 has evolved into an operating system which, judging by its use and popularity, has some merit. However, it is only a small step toward the goal of making the use of computers easy and convenient. Although, as with computing in general, it is impossible to step back the clock even a few years without realizing the progress which has been made, many of us do not appreciate how far we have come because we can see how far there is yet to go. Incorporation of high-level function within the hardware, easy to use hardware and software for less specialized user groups, and evolutionary growth complementary to the future's distributed computer and information utility all seem possible within the original CP/CMS design, but remain to be done.

The Control Program provides an environment for the operation of established systems and for the evolution of new systems and hardware within the multiple machine architecture. The ability to run several operating systems at the same time, both for function and convenience, appears to be a key element in the acceptance of VM/370. Each operating system, like CMS, RSCS, or OS/VS, provides important capabilities for an installation. Test and production systems can coexist to smooth the transition to a new release. Specialized and experimental systems can be developed conveniently without the disruption of normal work.

Acknowledgments

The design and implementation of the first CP/CMS system was a significant step in operating system design. I am indebted to the people who contributed to and supported our initial system design and implementation, in particular the work of L. W. Comeau, R. U. Bayles, and R. J. Adair on the Control Program, the work of J. B. Harmon on CMS, and the work of L. W. Comeau on the dynamic address translation hardware. The complete support of N. L. Rasmussen, manager of the Cambridge center, was crucial to the success of the entire project. Many dedicated and capable people, within and outside of IBM, quickly joined the ranks and continued the effort.

References

1. *Operating Systems Techniques*, C. A. R. Hoare and R. H. Perrott, Eds., Academic Press, Inc., New York, 1972.
2. *IBM Virtual Machine Facility/370 Introduction*, IBM Systems Library order number GC20-1800, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, NY 12602.
3. P. J. Denning, "Virtual Memory," *Computing Surv.* **2,** 153–189 (1970).
4. R. P. Parmelee, T. I. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual Storage and Virtual Machine Concepts," *IBM Syst. J.* **11,** 99–130 (1972).
5. *IBM Syst. J.* **18,** No. 1 (1979).
6. F. J. Corbató, M. Merwin-Daggett, and R. C. Daley, "An Experimental Time-Sharing System," *Proc. Spring Joint Computer Conference* (AFIPS) **21,** 335–344 (1962).
7. R. J. Creasy, "Research Time-Sharing Computer," IBM Systems Research and Development Center, Cambridge, MA, January 1965 (available from the author).
8. R. J. Adair, R. U. Bayles, L. W. Comeau, and R. J. Creasy, "A Virtual Machine System for the 360/40," *IBM Scientific Center Report 320-2007*, Cambridge, MA, May 1966.
9. D. Sayre, "On Virtual Systems," IBM Thomas J. Watson Research Center, Yorktown Heights, NY, April 1966 (available from the author).
10. L. W. Comeau, "Cambridge Address Translator," IBM Systems Research and Development Center, Cambridge, MA, April 1965 (available from the author at IBM Corporation, Department 57Z, Neighborhood Road, Kingston, NY 12401).
11. A. B. Lindquist, R. R. Seeber, and L. W. Comeau, "A Time Sharing System Using an Associative Memory," *Proc. IEEE* **54,** 1774–1779 (1966).
12. F. J. Corbató *et al.*, *The Compatible Time-Sharing System, A Programmer's Guide*, M.I.T. Press, Cambridge, MA, 1963.
13. F. J. Corbató and V. A. Vyssotsky, "Introduction and Overview of the MULTICS System," *Proc. Fall Joint Computer Conference* (AFIPS) **27,** 185–196 (1965).
14. John McCarthy *et al.*, *LISP 1.5 Programmers Manual*, M.I.T. Computation Center and Research Laboratory of Electronics, Cambridge, MA, August 1962.
15. T. Kilburn, D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner, "One-Level Storage System," *IRE Trans. Electron. Computers* **EC-11,** 223–235 (1962).
16. J. B. Dennis, "Segmentation and the Design of Multi-Programmed Computer Systems," *J. ACM* **12,** 589–602 (1965).
17. R. A. Nelson, "Mapping Devices and the M44 Data Processing System," *Research Report RC 1303*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1964.
18. E. C. Hendricks and T. C. Hartmann, "Evolution of a Virtual Machine Subsystem," *IBM Syst. J.* **18,** 111–142 (1979).

Received April 14, 1980; revised February 18, 1981

The author is located at the IBM Scientific Center, 1530 Page Mill Road, Palo Alto, California 94304.