CP/40—THE ORIGIN OF VM/370

L.W. Comeau[169]

Introduction

Perhaps what is significant is not the ability to sit down and plan invention, but the ability to recognize innovation when it occurs spontaneously. Such was the case with a software system called CP/40. Originally planned as a measurement tool, it has grown to become an equal partner with IBM's two other operating systems, DOS/VSE and MVS.

It is further interesting to note that this stepchild of the time-sharing community now enjoys more popularity than its well-funded contemporaries, IBM's TSS and GE's (Honeywell) Multics. It would be extremely gratifying to attribute that success to brilliant design decisions early on in the program, but, upon reflection, the real element of success of this product was that it was not hampered by an abundance of resources, either manpower or computer power.

An early consultant's report rated the modest goals of the CP project and claimed it was not price competitive with the larger systems, the IBM 360/67 and the GE 645. In a follow-up look, the same consultant said that the obvious error in his previous work was to accept the claims of the larger systems versus what was achievable. The CP/40 system met its goal of 15 users; the larger systems were incapable of supporting hundreds of terminals, which was their design point

CP/CMS Team

To understand the rationale for CP/40, it is necessary to appreciate the backgrounds of its designers. There were five major contributors to the design of this system—Bob Adair, Dick Bayles, Bob Creasy, John Harmon, and myself. Of this group, three were programmers and the remaining, Harmon and Comeau, were systems engineers. None were neophytes in the time-sharing world. Bob Adair was involved in command and control systems being built by the MITRE Corporation, and the remaining four had been associated with the MIT Computation Center in the years preceding the CP design.

The MIT association, in particular the goals of and experience with Professor Corbato's CTSS system, strongly influenced the design of the terminal user interface for CMS. In fact, the emphasis on "user friendly" interfaces brought on by the success of Wang in the office systems area is hardly "new news" to this group. Those at MIT in the early 1960's put a strong emphasis on the needs of the non-computer-oriented professional.

Indeed, the artificial separation of word processing from data processing that occurred in the office system market comes as a complete surprise to the people who have been VM users over the years. Word processing has always been available to the VM group, first in the form of

---

[169] This paper is reproduced from the *Proceedings of SEAS AM82,* September, 1982, with the permission of SHARE Europe and L.W. Comeau.
UNIX is a trademark of AT&T Bell Laboratories.

editors and print formatters and, later, incorporating the advantages of electronic mail. This, too, was part of the MIT philosophy; that is, to be of any use a system should provide the basis for its own evolvement, including the requirement for documentation.

The CP/CMS team was formed as part of the Cambridge Scientific Center, which was established by IBM management in 1964 to provide a center for competency in time-sharing. Although IBM had worked with terminal systems, such as the Sabre, the American Airlines on-line reservations system, it was felt that the general purpose time-sharing environment was significantly unique so as to require a separate research.

Baseline

In the six months immediately preceding the design of CP-40, the Cambridge Scientific Center had been involved in preparing two proposals for time-sharing systems, one to Project MAC at MIT, which IBM lost to the GE 645 Multics system, and one to MIT's Lincoln Lab, which was a winning bid and was the genesis of the System/360 Model 67 and the TSS operating system.

The major technological change proposed for these systems was virtual memory. It was felt that this offered a solution to both the programmer productivity constraint and to the performance problems faced by earlier time-sharing efforts. It was widely accepted in the early 1960's that the cost of producing programs rose exponentially as one approached the memory limit of a particular machine. Virtual memory, by releasing the programmer from this constraint, would obviously lower the cost and time to produce a large application.

Since the early time-sharing experiments used base and limit registers for relocation, they had to roll in and roll out entire programs when switching users. There was some talk of the "onion-skin" technique, where small programs would displace only parts of large programs when users were swapped, but to my knowledge it was never implemented. Virtual memory, with its paging technique, was expected to reduce significantly the time spent waiting for an exchange of user programs.

What was most significant to the CP-40 team was that the commitment to virtual memory was backed with no successful experience. A system of that period that had implemented virtual memory was the Ferranti Atlas computer, and that was known not to be working well. What was frightening is that nobody who was setting this virtual memory direction at IBM knew why Atlas didn't work.

Similarly, the functions to be provided in the end-user terminal were the subject of debate in the mid-1960's. There was a requirement to provide application programs with each single character as it was typed. It was thought that with that capability errors could be turned around instantaneously, and the system could thereby save the user retyping time and effort. This requirement exists today in the "raw mode" available to programs written for the UNIX system*

The requirement for upper and lower case received strong emphasis then. Today, of course, one finds few terminals without that capability, but there was little appreciation for that function amongst terminal builders of the early 1960's. Terminals were viewed as strictly data-entry devices.

A third terminal requirement generated in this area was the ability to turn off printing in order to suppress printing when the user entered authorization codes. We take it for granted now, but most terminal manufacturers did not include this function in their early models.

It was against this background that the Cambridge Scientific Center undertook the CP/40 project. It was our intent to study programs and programmers in a time-sliced virtual memory environment.

Vehicle

The 360/40 was chosen as the vehicle on which to implement the Scientific Center's time-sharing experiment. This was not the result of extensive load analysis but because of a lack of availability of a 360/50, which was thought to have the CPU power required to support our interactive population. It turned out to be fortuitous, because the modifications required to segment the memory for virtual addressing were easily accomplished on that hardware (System/360 Model 40).

Virtual memory on the 360/40 was achieved by placing a 64-word associative array between the CPU address generation circuits and the memory addressing logic. The array was activated *via* mode-switch logic in the PSW and was turned off whenever a hardware interrupt occurred.

The 64 words were designed to give us a relocate mechanism for each 4K bytes of our 256K-byte memory. Relocation was achieved by loading a user number into the search argument register of the associative array, turning on relocate mode, and presenting a CPU address. The match with user number and address would result in a word selected in the associative array. The position of the word (0-63) would yield the high-order 6 bits of a memory address. Because of a rather loose cycle time, this was accomplished on the 360/40 with no degradation of the overall memory cycle. In addition to the translate function, the associative array was used to record the hardware use and change status and the software-noted transient and locked conditions relative to a particular block of 4K bytes in the memory.

Since the array functioned as a content-addressable store when in supervisor state, searches to satisfy the LRU algorithm were quite fast. There is considerably more information on the associative memory in the referenced IEEE article by Lindquist, Seeber, and Comeau <ref. 1>.

The major difference between the CAT (Cambridge Address Translator) associative memory and our current line of relocate mechanisms was that the CAT was a memory-mapping device, whereas today's hardware employs a program-mapping scheme.

In a memory-mapping translation mechanism, there is one relocating entry for each page of real memory. In a program-mapping scheme, the hardware contains relocation information relative to the particular program that is executing, and information relative to real memory blocks is maintained elsewhere. The System/370, for instance, maintains use and change data in its key storage, which is a totally independent mechanism from the relocation mechanism.

Since it appears logically that memory mapping, *a la* S/360 Model 40, is superior to program mapping, then why isn't it prevalent today? The answer is cost; the original array cost thirty-five times what a conventional memory cell did at that time, and since then it seems that associative logic is still roughly eight to ten times what conventional logic costs. There has been little work done within IBM on associative technology, and, therefore, there is little likelihood that it will ever become price competitive in our hardware. What we should now look at is absolute cost and what associative logic can give us in additional function.

There is an additional problem in a memory-mapping scheme which doesn't exist with program-mapping techniques; that is page sharing. Memory-mapping schemes have only one entry per page of the real storage. To allow access among a group of users requires either changing that entry or having a second, or all-user, userid. The latter, of course, doesn't accommodate a selective sharing of memory.

The value of sharing programs in memory is suspect, based on the measurements taken during the life of the CP/40 project. There are some applications where it is desirable to share data in memory, and here the memory-mapping scheme is deficient. Interestingly enough, although sharing data has always been unpleasant in VM, yet no one seems to have as yet put forth an elegant solution.

CP/CMS Design

The two different goals of our project, one to measure S/360 software in a virtual memory, time-shared environment, and, second, to provide the Scientific Center staff members with an interactive facility, led us to a design which cleanly separated those two requirements.

The measurement requirement dictated that the functions to be measured and the algorithms to be modified and tested be very localized. A second motivation for maintaining a distinct separation became apparent when the strong wills and opinions of the group became apparent. I think that most designers recognize the need for good separation of function in programming system design, but compromise becomes the rule very early in the effort. With the particular group assembled to build CP/CMS, the personalities reinforced that design principle, rather than compromising it.

It seems now that the decision to provide a Control Program interface that duplicated the S/360 architecture interface <ref. 2> was an obvious choice. Although it was, given our measurement objective, it wasn't, given our in-house interactive system objective.

We were more secure with the decisions for the CMS external interface. It was clear, based upon the experience gained with CTSS, that a user-friendly command language was key. Another thing we had learned was that the system had to be very forgiving, and although options were desirable, default-mode, non-required parameters were to be a paramount design consideration in CMS.

The choice of an architected interface, the S/360, between CP/40 and its operating system turned out to have been most fortunate. It permitted simultaneous development of CP and CMS; it allowed us to measure non-virtual systems, OS and DOS, in a virtual memory environment, and it also provided a high level of integrity and security.

Conversely, this same design made sharing programs and data somewhat difficult. Even today's VM system seems lacking in this regard.

Program Model

Even though we all realized that there were loops in programs, essentially our model was that a program progressed linearly in its execution and data references. There certainly was no notion of "working set" prior to our original experiments.

We hoped to determine the proper page size, the rate at which page turning would occur in a multi-programmed environment, and the value of shared program pages. Program and user characteristics included the number of instructions executed between I/O requests, which is still key for system designers. What was the time slice required to deliver acceptable performance? How much time did the user think between commands?

Experiments

The experiments run on the CP/40 system yielded significant results in the area of virtual memory. First, we discovered the phenomenon currently known as "thrashing". I first reported it to an internal IBM conference on storage hierarchy in December, 1966 <ref. 3>. In a follow-up paper on virtual memory <ref. 4>, we showed the dramatic reduction in the required number of page swaps that could be achieved through some very simple user optimization procedures.

A third report finally published in 1971 <ref. 5 and 6> contains the largest amount of data ever compiled on a controlled virtual memory experiment. It originally took 63 hours to run, so it is doubtful anyone would find it worthwhile to repeat. The experimental factors chosen for that experiment were:

- replacement algorithm
- subroutine ordering
- problem programs
- memory size

For each factor, three variables were chosen, such that the total number of experimental values was 81. Although some of the factors could be argued as to their validity, the experiment which measured page swaps and active and inactive counts gave researchers a better feel for the interactions caused by these factors.

The most significant result of the CP/40 work was the recognition that a multiprogramming system naturally divides its function into three levels of privilege and protection:

- *level 0*—the control program level—assumes allocation responsibility for the serially reusable resources of the system. It is at this level that multiprogramming or inter-job management takes place.

- *level 1*—the job management level—contains the functions normally required by a single job to start, stop, do command interpretation, maintain data files, etc.

- *level 2*—the application level—contains the function a particular user wishes to accomplish.

In CP/40 and subsequent VM offerings, we implemented this three-level structure in what is essentially a two-level architecture (S/370, S/370). By defining a true three-level multiprogramming architecture, the overall job of providing those functions necessary to accomplish the level 0 and level 1 tasks would be greatly simplified, and the resultant path lengths would be considerably reduced.

Although most system programmers I've discussed this structure with agree to its merit, I'm not familiar with any hardware implemented this way.

A Look Back

The virtual machine design, as represented by the CP/CMS system, certainly has proven its viability, lasting now over fifteen years. During this period, we have added many new devices and software functions, so the system also meets the test of extendibility.

The success of the CP/CMS system certainly is in no small way attributable to its friendly and forgiving user interface.

A second contributor was the clean separation of function in the CP product, which made it easy for the sophisticated user population to remove, add, and substitute functions supplied by IBM, thereby greatly expanding the talent working on enriching the system.

Perhaps the most significant factor which contributed to CP's success in the middle to late 1960's was the failure of its competitors, Multics and TSS, to meet the commitments made to the marketplace. The market had been "hyped" to expect major function and performance (in the hundreds of terminals), and when they failed to deliver, expectations were lowered, but the need for time-sharing still existed among the customers. The CP/CMS system was operational and, therefore, represented an acceptable alternative.

The lesson to be learned when comparing CP/CMS with its contemporaries, TSS and Multics, is that it is easier to provide functions to an extendible high-performance system than it is to improve the performance of an integrated rich function system.

The problem for the future seems to be maintaining an architected three-level system structure within VM. There are major differences in the way an architect approaches the definition of an interface versus the way a programmer approaches it. An architect tries to insure the durability and completeness of what he specifies. He recognizes the scarcity of resources, be they parameters, bit interrupts, etc., and tests his specifications to make sure there is no needless resource consumption. The programmer views his job as to satisfy the requirements for a particular function. There is little concern for durability and conservation in this community.

As an example, the virtual machine interconnection facilities today are: Channel-to-Channel Adapter, VMCF, and IUCV. Essentially, they are all trying to pass data between virtual machines, but instead of extending and modifying the architecture of the original technique, the programming community invented net new things.

DIAGNOSE is another example of this phenomenon. Functions are provided in CP, invoked through the DIAGNOSE interface, without an appreciation for the logical (three-level) structure behind the VM design. Because of this, movement of CMS without CP level 1 and level 2 code to a standalone environment has proved extremely difficult. Remember that in the original implementation, CMS ran on a S/360 Model 40 without the requirement for the CP multiprogramming software.

In closing, I believe there is much to be gained if we build hardware which supports a three-level software structure, and I hope to stimulate interest in this by this presentation.

References

1. "A Time-Sharing System Using an Associative Memory", A.V. Lindquist, R.R. Seeber, and L.W. Comeau, *Proceedings of the IEEE,* vol. 54, no. 12, December, 1966.

2. "A Virtual Machine System for the 360/40", R.J. Adair, R.U. Bayles, L.W. Comeau, and R.J. Creasy, *IBM CSC Report,* May, 1966.

3. "Operating System/360 Paging Studies", L.W. Comeau, *IBM Storage Hierarchy System Symposium,* December, 1966.

4. "A Study of the Effect of User Program Optimization in a Paging System", L.W. Comeau, *ACM Symposium on Operating Systems,* October, 1967.

5. "A Multifactor Paging Experiment: Part I, the experiment and conclusions", R.T. Tsao, L.W. Comeau, and B.M. Margolin, *Statistical Computer Performance Evaluation,* Academic Press, Freiburger *etal.,* ed., 1972.

Same as above, *IBM Research Report RC3443,* July 9,1971.

Les Comeau