Some ways in which the virtual storage systems TSS/360 *and* CP-67/CMS *have been used in a research environment are described emphasizing the features of each of these operating systems found to be most useful. Descriptions of research projects employing the systems are given with the discussion centering on the reasons for choosing a particular system in each project.*

Uses of virtual storage systems in a scientific environment

by P. H. Callaway, J. P. Considine, and C. H. Thompson

In this paper, we describe some of the ways in which the virtual storage systems TSS/360 and CP-67/CMS have been used by scientists and programmers at the IBM Thomas J. Watson Research Center. We discuss some of the salient features of the two operating systems and summarize some of the projects that have been and are being carried out using these systems. For a more thorough discussion of the virtual storage concept itself, the reader is referred to other papers in the literature.[1,2]

The uses to which these systems are put vary from those that are relatively small applications, in terms of the demands on the system and the programming effort involved, to large applications and systems modifications. Some of the small applications such as document preparation and batch job preparation occupy a relatively small portion of the resources of the system but represent the interests of a large number of users. For instance, on the TSS/360 system, over half of the commands entered from terminals are addressed to an editing facility, principally the REDIT context editor. Although all projects utilize the editing facilities as an integral part of their programming activities, we concentrate in this paper on those that represent substantial nonediting demands on the systems.

As background, we mention that at the Research Center TSS/360 operates on a System/360 Model 67 with one million bytes of main storage and CP-67 operates on a Model 67 with 768 K bytes of main storage. In addition, The Research Center has a System/

360 Model 91 with two million bytes of main storage offering OS/360 with MVT/LASP batch processing and interactive APL service. There are also several IBM 1800 and 1130 computers in the building.

Model 67 software systems

TSS/360 and CP-67 were both designed to utilize the relocation hardware of the Model 67 of the IBM System/360 to provide multi-user conversational systems; their approaches, however, are different. TSS/360 uses the relocation hardware to provide each user with a maximum virtual storage of either 16 million or 4 billion bytes, depending on the hardware addressing option selected on the Model 67. The single overall design of TSS/360 attempts to supply a comprehensive system, with supervisor code, user support code (access methods, etc.) and compilers.[3] CP-67 attempts rather to provide each of its multiple users with a simulated System/360 machine—hardware only—in which the user can then use whatever operating system he wishes; e.g., DOS/360, OS/360, CMS. The Cambridge Monitor System (CMS) is an operating system designed for a single interactive user. It may be run either on a stand-alone System/360 or on one of the virtual machines provided by CP-67.[4]

TSS/360

features

Some of the features of TSS/360 that are used at Research are: dynamic allocation of resources, the Virtual Access Method (VAM), the Program Control System (PCS), and sharing of programs and files.

In addition to these standard features, two important functions have been added locally. These are the virtual storage context editor (REDIT) and the remote job entry to the OS/360 batch machine (NETOS).

Dynamic allocation of resources and VAM are features that are used more implicitly than explicitly. The resources allocated include virtual storage and external or file storage space on direct-access devices.

The dynamic allocation of virtual storage is carried out primarily through the functioning of a program called the dynamic loader. In general, the dynamic loader is invoked when a program is executed. It ensures that virtual storage is allocated for all routines referred to either directly or indirectly by the program. There are three main features which this program affords the user:

1. Flexibility in the use of subroutines. The dynamic loader loads a new version of a subroutine without the necessity of

recompiling the other routines in the program. Although TSS/360 offers a fully functional linkage editor, most users have no need of this facility and use only the dynamic loader.

2. Reduction in overhead. Although virtual storage is allocated for all programs referred to by a program, pages of the program that are not referred to during execution are not read from the program library. In addition, resolution of address constants is deferred until the page is brought into main storage. Thus, much overhead is not incurred for parts of the program that are not used.
3. Conservation of virtual storage. It is possible to write a program in such a way that virtual storage is not even allocated for a routine until it is actually needed. A call to the dynamic loader with the name of a routine as a parameter causes the loading of that routine in the middle of the execution of a program. That parameter can be variable so that only routines needed are loaded and have virtual storage assigned.

The dynamic allocation of file storage is best discussed in conjunction with VAM. Relocation hardware on the Model 67 dictates a natural unit for the movement of programs and data in and out of main storage. This unit is the *page*, which contains 4096 bytes. Thus very efficient supervisor routines have been developed for processing units of this size. VAM builds on the existence of these routines by adopting the page as the physical unit for external or file storage. Direct-access volumes are arranged in page-size blocks, and data sets are composed of numbers of these pages.

When a file is being created, the system assigns a certain number of pages. If more are needed, they can be obtained dynamically. The user does not have to preallocate all the space he needs, nor does he have to consider the type of device his data set is to reside on. In general, he may not even know. Also, when he erases data sets, the pages are made available again to the common pool from which subsequent requests are filled. Because allocation is in units of a page, there are no requirements for contiguity and no problems with small amounts of unusable space being scattered around on a volume. In addition, all data sets are cataloged, so the user need not keep track of volume information. The user is thus relieved of many of the bookkeeping duties associated with creating, maintaining, and erasing files. A relatively high degree of space utilization is also achieved.

The Program Control System (PCS) provides the user with a means of referencing instruction and data locations within his program symbolically; i.e., using the symbol rather than some storage address to specify the location desired. This can be done while the program continues to execute, or by interrupting the execution. In addition to displaying and altering variables or

instructions, the user can also interrupt, and alter or resume the flow of program execution. When the full capabilities of PCS are exploited, it becomes a powerful means of specifying a problem and directing its solution, used in conjunction with the more conventional language processors such as FORTRAN and the TSS/360 Assembler.

Central to the symbolic capability of PCS is the existence for each program of an internal symbol dictionary (ISD). This is a table of each symbol appearing in the program together with its location within the program and attributes such as integer, real, character, halfword, fullword, etc. This table is created by the compilers and stored with the program in the program library. It is dynamically loaded into virtual storage when a PCS reference is made to a symbol within the program; it varies in size from a few hundred bytes for simple programs to thousands for more complex ones. Its presence in storage is required for efficient PCS processing, but it does increase, sometimes substantially, the storage requirements of the program. Since, however, there is a very large virtual storage available in TSS/360, this factor does not present any additional difficulties. To provide a similar function in a fixed-size storage system would be more difficult and costly to the user in terms of either real main storage space or, if the ISD were to be accessed from direct-access storage, execution time.

The TSS/360 system at the Research Center has a powerful and efficient locally implemented context editor called REDIT.[5] (Most of the basic commands were adapted from the CMS editor.) All editing operations are carried out on a copy of the file being edited. Good performance is achieved primarily because this entire working copy is maintained in virtual storage throughout the editing process. (Many editors that run in nonvirtual storage systems perform a record I/O operation for each request at the cost of performance.) Modifications are made permanent only when requested by the user; at that time, the working copy is written out from virtual storage onto permanent storage. The ability to acquire large amounts of virtual storage to contain large files being edited as well as to incorporate additions that might be made is important to this design. Thus, virtual storage provides the support for using fast "in-core" editing techniques on very large files. The speed, power, and simplicity of the editor make it one of the most valuable tools of the Research Center TSS/360 users.

It is clear that we could implement on a nonpaging machine an editor that did its own software paging. Without the advantage of hardware interruptions when accessing material not in main storage, the implementer would have to set up and maintain tables and check them on each reference.

NETOS[6] is a facility for sending source programs and data to the Research Center's System/360 Model 91 for compilation and execution. Provision is also made for sending data sets produced on the Model 91 back to TSS/360. Since editing and debugging are more efficiently done in TSS/360 and the execution of large jobs is more efficiently done on the Model 91, the NETOS facility allows the user to develop and run programs more efficiently and easily than on either machine alone.

The last of the features of TSS/360 to be discussed in this paper is the sharing of programs and files among users. The TSS/360 catalog structure facilitates sharing of files by allowing a user to assign his own name to a file owned by another user, if the file owner has granted him access. The owner can grant either read-only, read-write, or unlimited access. The sharing user can then utilize this private name as if it were the name of one of his own files (within the constraints of the type of access granted him). The virtual storage of the Model 67 is divided by the hardware into segments. This allows TSS/360 to separate shared programs and data from private work. Different segment addresses are assigned to programs that may be shared by more than one user so that only one copy of a shared program must be in main storage at any time. Each user of a shared program is coupled by the relocation translation mechanism to the same copy of the shared program in main storage.

TSS/360 projects

The projects described in this section have been carried out on TSS/360 over the past few years and represent some of the major demands made on the system resources over that time. Some projects represent significant increases in function of the basic TSS/360 system. These include the TSS/360 network development and the TSS/360 data migration projects.

Some of the projects are major applications heavily dependent on the unique aspects of large virtual storage and other TSS/360 features. These include the simulation and study of environmental phenomena, analysis of the behavior of users of the TSS/360 system, automated image processing, the medical data bank and retrieval system, and the experimental learning projects.

TSS/360 is also used for interactive debugging and editing in conjunction with production runs on the Model 91 batch machine. Differential equation solving is an example of this mode of operation.

One project that has systems orientation but has been implemented as an application is the TSS/360 source library maintenance and retrieval package.

environmental phenomena simulation and study

The project on simulation and study of environmental phenomena has developed programs on TSS/360 to simulate the growth of forests and the analysis of meteorologic and hydrologic data.[7,8] Nearly all of the programs are written in FORTRAN. The debugging and editing of these programs is done on TSS/360 using the PCS and REDIT functions. Most of the simulation programs run interactively, with PCS being used as the master control language. By running in this fashion, any subset of the program variables or parameters can be changed symbolically and the flow of control altered dynamically from the terminal. This allows for a great deal of freedom with very little preplanning or recompilation and gives the simulation package user a much more powerful and easier-to-use tool than could be obtained by any of the conventional alternatives. A typical interactive session with the simulation programs is one to two hours long.

During the two months development of a forest simulation program, many algorithms were tried and discarded. The hierarchy of assumptions was determined empirically in a strongly interactive mode. The resulting model has been tested against the data from 10 by 10 meter plots of the Hubbard Brook Ecosystem Study and found to be quite accurate.

Some of the other programs are heavy CPU users and use large arrays. Typically these programs are debugged interactively on TSS/360 using PCS and small arrays; they are then sent to the Model 91 OS/360 machine via the NETOS command for execution on the large arrays. When the size of the arrays is very large (1400K bytes of data or so), the programs are run on TSS/360 where the large virtual storage is available. Many of the programs also produce output for pen and ink plotter or microfilm plotter. (Library subroutines have been added to TSS/360 at the Research Center for this purpose.)

automated image-processing

In another project, images to be processed are acquired by an IBM 1800 satellite computer and sent to the TSS/360 machine over the 1800-TSS/360 network to be stored in TSS/360 data sets. One image currently occupies 128K bytes of storage, and generally three images are in storage at a time. The programs currently have an arbitrary limit of 25 images at one time in storage, but there are plans to raise this limit. It would also be desirable to process color images and higher resolution images of 512K and 2048K bytes. Virtual storage allows for the processing of more and bigger images with a minimum of additional programming effort.

Three man-years have been spent developing the current programs in this project which are written in PL/I, FORTRAN, and Assembler language. The data set sharing facilities of TSS/360 are used for both programs and data. Various system commands are

issued from within the programs including data set definition and editing. Dynamic calls on programs from within other programs are also made via the TSS/360 dynamic loader. In a nonvirtual storage system, this sort of dynamic operation would have to be preplanned, and large sets of programs would have to be recompiled and/or link-edited. On TSS/360, these programs can use either tape or direct-access data sets as input without modifications or preplanning. The user also has available a hardware feature of the IBM 2741 terminal—the attention key—to initiate specific kinds of processing.

As might be expected, this project makes heavy use of the graphic output facilities provided by the library subroutines mentioned earlier. Also, some of the basic image-processing routines are executed on the OS/360 machine, using the NETOS facility.

medical data bank and retrieval system

"Present methodology for arriving at data-inferred categories covers a broad area of loosely related techniques, objectives and concepts. The iterative and interactive nature of the analytic process and the role of the computer in facilitating this analysis have been examined in a case study of a real problem—the assessment of the physiologic status of critically ill patients.[9] Three stages in the evolution of a severe infectious process producing a shock state in these patients were identified based on quantitative physiologic measurements. These stages were interpreted in the larger frame of reference provided by the clinical record and medical experience in a manner which has important clinical implications."[10]

This project has been conducted by Research Center investigators in collaboration with physicians from the Albert Einstein College of Medicine. Case histories of critically ill patients have been stored in TSS/360 data sets. A system has been developed for use by a nonprogrammer physician and/or statistician for analyzing and interrogating this data base and adding to it. Using PCS and TSS/360 command procedures, researchers have developed a language that allows users to interrogate the system in their own terms and direct that various analytical functions be performed or results displayed. This language allows modification by merely editing a command procedure rather than by having to recompile a program.

Large blocks of data are accessed randomly by this system, with the data being handled as arrays in storage by the programs that refer to them. This feature depends heavily on the availability of large amounts of storage space, without which a major redesign would be required. The data are stored as modules in a library; they can then be loaded by the TSS/360 dynamic loader as required, rather than all data having to be

processed by I/O routines before execution can begin. This feature means that the four million-byte data base is available for the programs in a matter of less than a minute rather than the 30 or more minutes which would be required to process that much data by the regular I/O routines.

The libraries containing the case histories and the programs are shared, enabling use by several people more or less concurrently. About eight million bytes of on-line storage are currently in use. The programs are written principally in FORTRAN with some Assembler language coding. A link to the 1800 processor is used for video display on screens driven by the 1800.

analysis of user behavior

One project is studying the behavior of the interactive users of TSS/360. All transactions of all users of TSS/360 are recorded and time-stamped by a program called SIPE.[11] Some data reduction programs are run against these raw data to sort them by user identification (USERID). This sorted data base is then processed further to produce profiles for each user and profiles of total system use. The following sort of information is compiled: system response time, user response time, average number of characters typed by the user of the system, frequency of use of the various commands, frequency of user attentions (i.e., the user striking the ATTN key on his 2741 terminal), and the frequency of successful and unsuccessful compilations. The analysis of these data has produced important insights of value to designers of future interactive systems.

The analysis programs are written in PL/I and build large list structures and tables that require a large amount of storage. Extensive use is made of both the stream and record I/O features of PL/I in both sequential and indexed access. The programs are executed interactively so that the human analyst can intervene in intractable or unforeseen cases. The large virtual storage system has greatly simplified the development of these programs. The input data for these programs are used by other project members for other associated investigations. The data-sharing facilities are very valuable in this regard. TSS/360 also provides temporary storage which is erased at the end of a terminal session. This project has found this storage very valuable for scratch space during program development and testing, since it does not affect the allocation of permanent storage for each user.

Documentation and reports are prepared using the editing and documenting facilities of TSS/360.

experimental learning

In another project, an on-line question-answering system had been developed to study whether or not there can be an effective learning environment in which the only material presented is responses to student-initiated questions. A data base, which is

potentially very large, is completely loaded into virtual storage. The programs assume that storage is almost infinite in size.

A large amount of direct-access space is required to store the data bases and associated programs. TSS/360 sharing allows many students to use the same data base and also removes the need for multiple copies of programs and data. REDIT and text-formatting facilities are used extensively for the preparation of the data base. PCS has been used for much of the debugging together with a disk-patching utility that enables corrections to be made to assembled code by storing the correct instructions in the copy in the program library. The speed of development of this system has been definitely increased because of the presence of the large virtual storage and PCS, but as yet no answer to the original question has been obtained.

source library

The primary function of the source library maintenance and retrieval project is to maintain strict control of a source library, specifically the source code for the TSS/360 system, while giving the user the necessary tools and information to carry out his modifications with little inconvenience. A library system is established, with the data set sharing functions of TSS/360 being the method to give users access to the library. They are supplied with commands and routines that enable them to request copies of the source and submit their changes for inclusion in the library. The system uses VAM and TSS/360 command procedures and code to handle the movement of data from library to user and back. It has greatly simplified the modifications and additions to TSS/360 system code. It can also be applied to private source libraries simply by changing the identity of the librarian, a user with special privileges.

solving differential equations

The applications just described make frequent use of the special features of TSS/360, which are essential to the progress of these projects. We will now mention briefly some more conventional computing applications that are carried out on TSS/360.

In one instance, differential equations being solved represent the excitation and propagation of electrical impulses in nerve and heart tissue. The solutions are arrived at by an iterative process based on assumed starting values, and the interactive system is very useful for monitoring the course of the calculations using PCS. In this fashion, if the solution diverges, the run can be terminated and better starting values substituted for the next try. Another set of equations being solved describes the behavior of electrons in solids.

In both cases, programs are written in FORTRAN and run on either TSS/360 or OS/360. Debugging and editing are carried out on

TSS/360, and production runs are sent to the faster Model 91 for OS/360 processing via the NETOS link. The use of arrays exceeding the capacity of the Model 91 is anticipated. At that point, execution will then take place under TSS/360 without reprogramming because of the large virtual storage available.

TSS/360 network development

The TSS/360 system has been augmented by the addition of a large amount of code developed at the Research Center for the support of inter-processor communication,[12] including the basic teleprocessing routines called the Computer Access Method (CAM). Beginning with the support of satellite computers such as IBM 1800's and IBM 1130's the project expanded to include the ability for TSS/360 systems to transmit data and job requests from one to another over telephone lines. An additional function supplied by this effort was the implementation of the NETOS remote job entry facility for transmission of jobs to the OS/360 batch machine and reception of resulting data sets back to the TSS/360 machine. One of the mainstays of this effort was the fact that TSS/360 has a special access method designed for the support of nonstandard devices. Using this access method, we can issue channel programs to any device and handle the returns as the device requires with the system simply acting as agent. This project was also aided by the ability of a virtual storage problem program to communicate with the supervisor and make specific requests such as task initiation, special message handling, etc., which are essential to the transfer of data from TSS/360 system to TSS/360 system.

The implementation of the system was aided by the data and program-sharing facilities of TSS/360 that enabled the multi-man project to proceed in awareness of each other's work. One specific aspect of this sharing is that the NETOS processor that receives output from the OS/360 batch machine can utilize the sharing logic of TSS/360 to ensure that the data sets become the property of their rightful owners.

data migration

Another modification to the TSS/360 system was in the area of the on-line data base and its control. On-line storage is a valuable asset, and to ensure maximum utilization with minimum inconvenience to the user, a system called data migration was instituted.[13] Data sets that have not been referenced for a certain period of time are moved from the permanently mounted on-line volumes to demountable volumes. This process is called "migration". The data sets remain in the TSS/360 catalog, and when the user refers to one, he receives a message advising him of its migrated status. He then issues a simple command to restore it to its on-line status and uses it normally. The successful operation of this system depends in great measure on the central catalog of TSS/360 and on VAM. The cataloging of data sets relieves the user of the need to know where his data set is and allows the

system to keep track of its location for him. Thus even though a data set has been migrated, the user cannot inadvertently create a data set with the same name on on-line storage, since the catalog allows only one data set of a given name for each user.

VAM makes it possible to move data simply between volumes without elaborate concern for device type or preallocation of space, thus making the process of migration and restoring of data sets a very efficient one. The user is only slightly inconvenienced by occasionally having to restore a data set that has been migrated.

What the system gains for the user is in actuality a much larger effective on-line storage ration than he could possibly have otherwise. Since the number of volumes mounted at one time on the system depends on the number of drives available, an increase in total on-line space available can only come with an increase in available drives, an expensive proposition. Instead, the increase in effective storage comes at the expense of an increase of the number of volumes available for migrated storage, a much less expensive item. The user is relieved of the necessity of deciding to discard data which he might later need just because there is no place to store it.

These system modifications made use of some of the strong points of TSS/360—catalog structure and sharing, VAM, and support of nonstandard devices.

summary

The synopsis just presented of the projects in operation under TSS/360 gives some indication of the scope of uses to which the system is put. From the system point of view, the central catalog and sharing mechanisms and the ease with which nonstandard devices can be supported have been of greatest utility. Effective communication between user task and supervisor has also been very valuable. From the application user's point of view, the large virtual storage, data and program sharing, and PCS have found greatest utility. The existence of the powerful and convenient context editor for programs and text, REDIT, has been integral to the progress of all users.

CP-67/CMS

features

The CP-67/CMS users at the Research Center have been attracted by several major features of the system. These include: (1) virtual machine capability and associated functions, (2) simple, easily learned command and file system in CMS, including a context editor, (3) OS/360 compatible FORTRAN, PL/I, etc. available at the terminal under CMS, (4) good response to interactive transactions, and (5) compactness of CP-67/CMS, with the ability to easily generate new versions containing local modifications.

The virtual machine capability provides the designer, implementer, or modifier of a System/360 operating system with a convenient means of carrying out system development and modification. The CP-67 system simulates a standard System/360 hardware environment for each user of the system. The user can load his system into this virtual machine and test, debug, and modify the system while sharing the real hardware facility with other users. Since each user is provided with a separate address space (virtual storage) starting with address zero, few modifications to systems being tested are required to take account of this sharing. By eliminating the usual requirement for stand-alone machine time for system program testing, more diverse use of the hardware can be made, and numerous development projects can proceed in parallel.

Several other aspects are significant. Systems can be developed for machines that are not available for testing or perhaps do not even exist. Interprocessor communication systems, for instance, can be tested by activating two different virtual machines and communicating between them by means of standard communication hardware. Because CP-67 simulates the System/360 hardware environment for each virtual machine, there is a well-defined interface that provides a unique opportunity for monitoring the activity of a virtual machine. Static displaying of registers, program status words, and dumping of main storage is possible, as well as dynamic tracing and data collection concerning events of interest in the virtual machine. Thus program development and debugging are easier in the virtual machine environment because pertinent data are more readily available to the user when his system is operating either normally or abnormally.

In addition, working systems can be operated on virtual machines in the absence of the real hardware, making possible extensions in capability and availability.

The speed of development efforts is also increased by the availability on the same system (CP-67) of the Cambridge Monitor System (CMS). This conversational system, affording simple file structure, easily learned command structure, and good editing facilities, facilitates program preparation and testing. Convenient communication between different CMS virtual machines and users, and between a CMS virtual machine and a virtual machine running a large operating system, such as OS/360, greatly speeds up the edit-compile-test cycle required by the development process. The editing facilities also support the preparation of documentation for systems and applications.

System developers and applications programmers are attracted by the availability under CMS of OS/360 compatible language processors, e.g., FORTRAN and PL/I. This means that the results of

interactive program development can be used directly for production processing on an OS/360 batch machine.

CP-67 and the Model 67 hardware make it possible for all CMS users to share the same copy of the CMS nucleus in real main storage. This enhances CMS performance by reducing the amount of paging required for essential CMS operations.

CP-67/CMS projects

In discussing the projects carried out on CP-67, we distinguish between those that rely on the existence of the private virtual machine capability and those that are developed in the CMS framework. Included under the first classification are the integrated computer network and laboratory automation projects. The second category includes the symbol manipulation project and the computer-aided circuit design efforts. Two other important activities, System/A development and the maintenance and enhancement of CP-67 and CMS themselves, make much use of both aspects of the CP-67/CMS system.

network projects

The purpose of the integrated computer network projects is to produce modifications and enhancements of OS/360 code to enable computer-computer communication over teleprocessing lines, including transmission of data, jobs, and output.[14,15] The initial decision to use CP-67 was made partly on the basis of the availability of the virtual machine capability. In particular, testing of the processor-processor communication links was neatly handled by activating two or three OS/360 virtual machines under CP-67 and carrying out the communication over standard hardware.

Members of these projects have claimed that debugging in a virtual machine environment is "an order of magnitude easier" than in a batch environment. For example, the network system was converted to a new version of OS/360 at another location under a batch system. It is estimated that this process took almost five times longer than it would have in the CP-67 environment, even though the programmers involved were equally conversant with the procedures of both systems.

New code was prepared, assembled, and partially debugged under CMS before the object code, together with the job control language, was shipped to the OS/360 virtual machine for link-editing and testing. Editing and assembly were able to proceed in parallel by using the CMS batch facility, which enables the user to ship jobs to a batch CMS virtual machine for asynchronous execution. The CMS editor was used to produce project documentation.

Programmer productivity in the CP-67/CMS environment was felt by project management to have improved. However, the observation was made that some conversational users lapse into sloppy habits, knowing that their errors are easily corrected. Batch users tend to be more careful because of the larger time penalty for a wasted run.

The main price paid for the virtual machine capability is the increase (by two to four times) in the amount of CPU time required for a job. This is due to the overhead in the simulation of the real System/360 interface including the interception, handling, and reflection of privileged instructions.

laboratory automation

Several laboratory automation experiments at the Research Center were developed around a System/360 Model 44 and its operating systems PS44 and MPS44. The Model 44 was augmented by separate experimental hardware designed to facilitate highspeed data acquisition and analysis. The programming takes full advantage of the dedicated machine environment of the Model 44 and the special hardware to provide a multiprogramming system capable of supporting several real-time experiments concurrently. These include a Scanning Electron Diffractometer and a Scanning Electron Microscope.

For a variety of reasons it became necessary to supply some of this function in a virtual machine environment. The first problem encountered was that the Model 44 multiprogramming system used dynamic channel program modification to process concurrent experiments efficiently. This is a function which CP-67 does not support in its virtual machine environment. A compromise had to be made; this took the form of providing separate virtual Model 44 systems for each experiment, allowing CP-67 to handle the multiprogramming. By this and other manipulations it was possible to provide an acceptable level of performance to the experiments with virtual machines running modified Model 44 systems.

Thus, in spite of the fact that real-time data acquisition and processing are not an application to which the CP-67 virtual machine environment appeared particularly suited, careful study and analysis of the problems resulted in a quite viable system supporting these experiments.

The projects just described typify the kinds of situations in which the existence of the virtual machine capability is advantageous. Let us turn now to some applications that are primarily related to the CMS environment.

symbol manipulation

The primary goal of the symbol manipulation project[16] has been the design and implementation of a symbolic mathematics facili-

ty, which provides users with a powerful algebraic capability. Since its very inception, the project has aimed toward making this capability available in an interactive environment. The resulting system is called SCRATCHPAD.

SCRATCHPAD consists of some 3000 functions written in the LISP programming language and runs on an experimental System/360 LISP system. The features of SCRATCHPAD include a concise input language with notations resembling those of conventional mathematics, a large library of symbolic facilities, and a flexible evaluation mechanism that provides the interactive user with considerable control over a symbolic calculation.

System/360 LISP was initially implemented in a batch environment because of the early availability of OS/360. Subsequently, a completely compatible version was developed for CP-67/CMS to provide the desired interactive environment for SCRATCHPAD. The most recent version of System/360 LISP used for the SCRATCHPAD system requires a large (768K) virtual System/360 machine. Such large storage requirements would considerably impact the availability of SCRATCHPAD were it restricted to the batch OS/360 environment.

During the development of a large system such as SCRATCHPAD, functions tend to be defined, or modified, with considerable frequency. This fact, coupled with the usual style of writing LISP functions as short definitions, made the implementation of SCRATCHPAD considerably more efficient in an interactive environment. The debugging process was enhanced by the ability of the interactive user to utilize the file-handling and editing capabilities of CMS directly from LISP. System/360 LISP also includes dynamic debugging facilities such as function tracing and the ability to display the active function stack after errors and interruptions.

The characteristics of a symbolic computation using SCRATCHPAD are a small number of input/output requests, relatively widely scattered storage references, and a high CPU utilization. The typical ratio of virtual CPU activity to overhead activity is often 9:1 or higher. Primitive functions and common data objects have been collected into a small area of storage to improve paging characteristics. Still, the high requirement for storage and the large demands on the CPU make it impractical to operate more than a few SCRATCHPAD systems in the CP-67/CMS environment when a large number of other users are on the system, or when all SCRATCHPAD users are simultaneously carrying out large-scale symbolic calculations. Other System/360 LISP activities place an appreciably lighter load on CP-67/CMS.

Since March 1971, approximately 30 user problems have been

attempted on the SCRATCHPAD symbolic mathematics system. In many cases, the interactive use of SCRATCHPAD was of considerable benefit in achieving results, either because of the immediate response that it gave, or because of the necessity of trying alternate paths before completing the computation.

computer-aided circuit design

Algorithms have been developed using FORTRAN programming to assist in the design of semiconductor integrated circuits and devices.[17] The major computing requirements involve compiling a design engineer-oriented input language into a large sparse system of linear algebraic equations and solving them. The development of the programs has been done by a group of people who have found that the management, coordination, and productivity of the group has been enhanced by working in a time-sharing environment rather than in a batch environment. The availability of OS/360-compatible compilers in the interactive CMS environment has given them the best of both worlds by making the results of the interactive development efforts directly available for processing on the fast Model 91 OS/360 batch machine.

The inherently large scale of the applications and the necessity for generality in the design language make the total program package reasonably large (about 200 routines, comprising some 60,000 cards for the source decks). Main storage and disk requirements are correspondingly large, so that the assignable virtual machine size and disk-linking features of CMS become important, as well as, of course, its responsiveness.

System/A

One project begun on CP-67 was an experimental multiprocessing operating system for System/370 hardware called System/A. This project relied heavily on the existing and somewhat modified resources of CP-67/CMS for nearly all aspects of its computing requirements. The major portion of the programs was being written, compiled, and debugged under CMS, conversationally or in batch. System components were to be integrated and tested using a new test environment dependent on the CP-67 virtual machine principle, with CMS providing system maintenance capabilities and used to monitor and debug the multiprocessor software that would run in a virtual machine suitably tailored to simulate the hardware.

There are three major reasons why CP-67/CMS was chosen to provide the support for this project:

1. PL/I was chosen as the language in which the system would be written for reasons of programmer productivity, language features, and capabilities. The PL/I optimizing compiler was needed for maximum efficiency of the resulting code. Examination of alternatives led to the decision that this compiler could be installed under CMS without undue effort.

2. The CP-67 virtual machine capability provided the most straightforward possibility of modeling the System/370 environment and gaining months of development time prior to the availability of the actual hardware.

3. CP-67/CMS is small and manageable enough that a small number of people could undertake to modify and enhance it sufficiently to create an environment suitable for the development of the system. This supposition has been supported by the speed and ease with which a host of features and functions have been added to CMS.

system programming

One of the functions carried out on the CP-67/CMS system is work on the system itself. This includes maintenance, enhancement, measurement, and analysis of CP-67 and CMS. The virtual machine capability of CP-67 includes the ability to model a Model 67 with its relocation hardware as well as the standard Systems/360 line. The running and testing of new versions of CP-67 in a virtual Model 67 created by CP-67 constitutes a relatively light load on the system. Consequently, such activities may take place at any time of the day. Several development cycles (updating, generating a new system, running the new version in a virtual Model 67, finding bugs and correcting them) may be achieved in a morning's work. Thus development of measurement and analysis tools and incorporation of additional CP-67 functions have been able to proceed at a far faster pace than was possible prior to the existence of the virtual Model 67 function. New program features can easily be received, applied, tested, and installed on the floor system within the space of a few hours if necessary, with little effect on the day-to-day running of the system except for reloading the floor system with the new items included.

CMS systems programming is similarly straightforward. A stable CMS virtual machine is used to provide all the support procedures and utilities to create updated versions of CMS, which can then be tested from the same virtual machine. A new CMS component may be created, link-edited into the nucleus and tested in a matter of minutes. Nonresident or transient modules may be handled even faster. All together, with the aid of the powerful CMS procedures for program preparation, CMS and CP-67 systems programming proceeds rapidly without the need for off-hours testing and stand-alone machine time.

summary

This synopsis of the projects undertaken on CP-67/CMS indicates the uses to which the unique features of that system have been put at the Research Center. For systems programmers and applications programmers who wish an interactive system with OS/360-compatible language processors, the CP-67/CMS combination has proved quite satisfactory. While the availability of the

virtual machine function leads to a large number of systems-oriented users on the system, there are still a number of primarily application-oriented users who find their requirements satisfied by the system.

Summary comment

This paper described some of the uses to which the virtual storage systems are being put at the IBM Thomas J. Watson Research Center. In addition to these relatively large-scale uses of the systems, a number of people are using the systems primarily, or even exclusively, for editing and document preparation. The interactive nature of these systems commends them to this application, and the presence of powerful editors simplifies it further.

To summarize the Research Center experience with interactive virtual storage systems, we can say that when functions exist, users will find ways to exploit them for the solution of their problems. Problems undertaken on a system tend to reflect the functional capabilities of that system, and the ease with which that system can be used. The users at the Research Center have found TSS/360 and CP-67/CMS useful tools in the furtherance of their research goals.

ACKNOWLEDGMENT

The authors would like to thank the users of virtual storage systems at the Research Center for the time they have spent with us, discussing the uses to which they are putting these computing systems. We hope we have faithfully if not adequately presented their work.

CITED REFERENCES

1. R. P. Parmelee, T. I. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual storage and virtual machine concepts", *IBM Systems Journal* **11,** No. 2, 99–130 (1972).
2. B. W. Arden, B. A. Galler, T. C. O'Brien, and F. H. Westervelt, "Program and addressing structure in a time-sharing environment," *Journal of the ACM* **13,** No. 1, 1–16 (1969).
3. A. S. Lett and W. L. Konigsford, "TSS/360: A time-shared operating system", *AFIPS Conference Proceedings, Fall Joint Computer Conference* **33,** 15–28 (1968). Also, *TSS/360 Concepts and Facilities*, C28-2003, IBM Corporation, Data Processing Division, White Plains, New York (1968).
4. R. A. Meyer and L. H. Seawright, "A virtual machine time-sharing system," *IBM Systems Journal* **9,** No. 3, 199–218 (1970).
5. C. H. Thompson, *User's guide to the Research Context Editor*, Research Report RA 28, IBM Thomas J. Watson Research Center, Yorktown Heights, New York (November 1971).
6. W. S. Hobgood, "Evaluation of an interactive-batch system network", *IBM Systems Journal* **11,** No. 1, 2–15 (1972).
7. D. B. Botkin, J. F. Janak, and J. R. Wallis, "Rationale, limitations, and assumptions of a northeastern forest growth simulator," *IBM Journal of Research and Development* **16,** No. 2, 101–116 (March 1972).

8. N. C. Matalas and J. R. Wallis, "In hydrology h is a household word," *Proceedings of the Warsaw Symposium of the International Association of Scientific Hydrology* **1,** 375–393 (1971).
9. R. M. Goldwyn, L. Loh, and J. H. Siegel, "The analysis of physiologic abnormalities in the critically ill using a time-shared system for the conversational manipulation of a large data bank", *Proceedings of Princeton Conference on Information Sciences and Systems*, 317 (1969).
10. R. M. Goldwyn, H. P. Friedman, and H. H. Siegel, "Iteration and interaction in computer data bank analysis: A case study in the physiologic classification and assessment of the critically ill", *Computers and Biomedical Research* **4,** 607–622 (1971).
11. W. R. Deniston, "SIPE: A TSS/360 Software Measurement Technique", *Proceedings of the 24th National Conference of the ACM*, 229–245 (1969).
12. R. M. Rutledge, A. L. Vareha, L. C. Varian, A. H. Weis, S. F. Seroussi, J. W. Meyer, J. F. Jaffe, and M. A. K. Angell, "An interactive network of time-sharing computers", *Proceedings of the 24th National Conference of the ACM*, 431–441 (1969).
13. J. P. Considine and A. H. Weis, "Establishment and maintenance of a storage hierarchy for an on-line data base under TSS/360", *AFIPS Conference Proceedings, Fall Joint Computer Conference* **35,** 433–440 (1969).
14. D. Fredericksen and R. W. Ryniker, "A computer network interface for OS/MVT", *Proceedings of SHARE XXXVI* (1971) (in press).
15. D. B. McKay, D. P. Karp, J. W. Meyer, and R. S. Nachbar, "Exploratory research in netting", Chapter 12 of *Computer Communication Networks*, edited by N. Abramson and F. Kuo, Prentice-Hall, Englewood Cliffs, New Jersey (in press).
    Research Center, Yorktown Heights, New York.
16. J. H. Griesmer and R. D. Jenks, "SCRATCHPAD/1–An interactive facility for symbolic mathematics", *Proceedings of the 2nd Symposium on Symbolic and Algebraic Manipulation*, Editor, S. R. Petrick, Association for Computing Machinery, New York, New York, 42–58 (1971).
17. G. Hachtel, F. Gustavson, R. Brayton, and T. Grapes, "A sparse matrix approach to network analysis", *Computerized Electronics, Proceedings of the 2nd Cornell Electrical Engineering Conference*, School of Electrical Engineering, Cornell University, Ithaca, New York, 68–82 (1969).