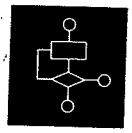

TIME-SHARING SYSTEMS:

VIRTUAL MACHINE CONCEPT VS. CONVENTIONAL APPROACH

Two fundamental classes of time-sharing systems are examined.

It has been said that the people involved in the development of time-sharing systems look upon time-sharing not as a technique but as a religion. This situation results from the devotion of these people to their work and the fact that their personal satisfaction is a contagious force and usually spreads throughout the group.

An unfortunate side-effect of this fervor is the belligerence of one time-sharing group toward another group. One of the contributing factors to this belligerence is the communications problem. Members of different time-sharing development groups frequently behave like people discussing different religions. When the discussion gets heated, each side falls back on the mystique of its own techniques and jargon, thus each side becomes convinced of its ultimate righteousness and the stubbornness of the other side.

Time-sharing is a new field which requires an entirely new vocabulary. The problem is that each group makes its own definitions and even if a term becomes commonly accepted, groups develop new words with the same meaning, as if the development of jargon were as essential as the development of systems.

Since this article discusses the aspects of whole classes of time-sharing systems, it can easily fall into the role of the innocent bystander in the midst of a war. Thus, the reader is asked to accept the definitions used here (at least while reading the article) and to consider the over-all discussion rather than the minor details.

CLASSES OF TIME-SHARING SYSTEMS

For expository purposes, all time-sharing systems will be divided into two classes:

Conventional Time-Sharing Systems (CTSS)—including MIT's CTSS and Multics, IBM's TSS/360, and others.

Virtual Machine Time-Sharing Systems (VMTSS)—including systems such as IBM's M44/44X project and CP/40 - CP/67 projects, and MIT's PDP-1 Time-Sharing System.

No special significance should be attached to the terms used. The VMTSS are relatively few, whereas CTSS are much more broadly accepted, thus more "conventional."

In this section, the basis for differentiating between the two classes is presented. The major factor is the user's independence of the time-sharing system, as measured by his ability to operate without it. (The terms "user" and "user program" are used interchangeably here.)

In CTSS, the user is aware of and interacts directly with the system. He must request that the **system** perform input/output for him, manage secondary storage for him, and perform many other functions. In general, the user's program could not run on any machine which did not include the CTSS. An alert reader could immediately comment that it may be possible to write a simple interface routine that could process the user's CTSS requests and perform the functions for him. But the fact that the user needs this extra level of buffering from the physical machine points out the difference: the user never interacts with the machine but is always one step away.

In VMTSS, the user is unaware of the system. It is totally invisible to him. The user must specifically write his own input/output routines, which contain actual machine I/O instructions. In most true VMTSS, the user can switch between the two standard physical hardware modes of operation, called "master" and "slave," or "supervisor" and "problem." Usually any special paging or mapping hardware that may be available is not made accessible to the user, but this is not a necessary restriction. In actuality, the user's privileged instructions, such as I/O or mode switching, are trapped by the VMTSS and appropriately simulated. The simulation may be simplified by the use of special hardware features, but these considerations (as in CTSS merely affect the efficiency of the system. The important points are the fundamental features of the systems, not the implementation of the features.

Since the user of a VMTSS actually writes his programs as if they were to run on a "real" machine, he can at any time separate himself from the system (and all of its unique special hardware, if any) and run on any "bare" machine. The instructions available on the M44/44X VMTSS unfortunately match neither the standard IBM 7044 nor the modified M7044 identically, thus, a program written on this system cannot be run on any existing real machine. This point is somewhat academic, since a machine could be built to have such an instruction set. On the other hand, the CP40 and CP67 VMTSS creates virtual standard IBM System/360 computers and the PDP-1 VMTSS creates virtual PDP-1 computers. Except for some minor restrictions, usually involving I/O timing dependencies and instruction sequence timing, the virtual 360's and PDP-1's are identical to their real counterparts.

The distinctions between CTSS and VMTSS are not completely well defined: for example, the MIT CTSS runs the FMS batch processing system as background in a "virtual" fashion, and the Project MAC Multics CTSS creates "virtual" core memory for the user. The differences between VMTSS and CTSS are largely in degree. A CTSS characteristically provides a **software** interface to the user, whereas a VMTSS presents the user with a simulated **hardware** interface to a virtual computer.

STUART E. MADNICK, Research Assistant ■ MIT, Cambridge, Mass.

FEATURES OF CTSS

Whereas VMTSS designers view with pride their ability to simulate real machines with ever-increasing accuracy, the CTSS supporters have a philosophy of giving the user as much freedom as possible and, furthermore, see no reason to limit him to the specifics of any real machine. It is possible to provide features and services in a CTSS environment that are not normally available on a computer.

Since the VMTSS merely create an "empty" virtual machine, the user must write or obtain routines to perform input/output for him. Even the loaders, assemblers, and compilers must be acquired by the user. The CTSS operates as a package deal; in addition to the time-sharing function, it provides many efficient utility routines and translators that are available to all users. For example, as F.J. Corbato and V.A. Vyssotsky state in "Introduction and Overview of the Multics System":

> **However, at Project MAC it has turned out that simultaneous access to the machine, while obviously necessary to the objective, has not been the major ensuing benefit. Rather, it is the availability at one's fingertips of facilities for editing, compiling, debugging, and running in one continuous interactive session that has had the greatest effect on programming."**

Most CTSS are designed with the idea of the computer utility in mind. Efficiency and reliability are of prime importance, and the particulars of operation are of minor concern as long as they are acceptable to the user. In the QUIKTRAN and CPS systems, for example, the user communicates with the system via typewriter terminals using FORTRAN or PL/1 language, respectively. Specifics, such as whether the programs are compiled or interpreted, are of little significance as long as the function is performed. As an aside, it is possible to consider such CTSS as creating "virtual" FORTRAN or PL/1 computers.

The CTSS protect each user from conflict with other users but, at the same time, users are allowed to share data and program procedures. The system operates as a massive pool of resources. It can allocate processor time and storage capacity in an efficient manner since it is supplied with information from the system routines being used. Each user enjoys the benefit of the efficiency without having to average the demands of his particular program.

Even without the additional scheduling information available from the CTSS routines, the CTSS technique should be more efficient than the virtual machine approach. A significant amount of input/output is required by most programs. On a CTSS, this is performed by a special call to the system which then schedules and performs the function in the most efficient manner. The VMTSS user must generate I/O instructions which are then trapped, diagnosed, translated, mapped, executed, and the results simulated. In fact, the user might even supply, for his input/output, elaborate error-checking and recovery routines which duplicate the functions that are already performed by the system.

FEATURES OF VMTSS

The most important feature of the virtual machine approach is that the system is not dedicated to any language, application, or mode of operation. Although most VMTSS supply a basic monitor with assemblers and compilers in the form of a read-only tape or disk shared by all users, the user can choose an alternate monitor or write his own. Multiple concurrent users may load the same or different software systems; in which case, each enjoys a dedicated virtual machine, while time-sharing the real computer. For example, under CP67 it is quite typical to have one user running the standard IBM batch processing system OS/360, another running a standard stand-alone utility program (such as tape conversion), a third running his own developmental operating system, while the remainder of the users operate under the supplied console-oriented operating system, CMS.

STUART E. MADNICK is a Research Assistant at MIT while simultaneously completing degrees of Masters of Science in E.E. and Management. He has been on the teaching staff of MIT's Digital Computer Programming Systems course for several years and has worked as a consultant in the area of language processors, multi-access computers, and file systems for the Lockheed Palo Alto Research Laboratory, INTERCOMP, Computer Software Systems, IBM Cambridge Scientific Center, and various MIT Departments.

Software written for a virtual machine may be executed on a standard real machine. Systems programmers designing input/output supervisors, monitors, and other privileged routines are probably the major users of on-line debugging techniques, and it is precisely these problems that the conventional time-sharing system is usually unable to handle. On the VMTSS, all systems programmers could be doing on-line debugging on virtual machines, simultaneously. The developed software can thus be delivered for use without ever wasting any real machine time for debugging.

In theory, it would be possible to develop a time-sharing system on a virtual machine, or even develop a VMTSS on a virtual machine provided by a VMTSS on a virtual machine, provided by a VMTSS, etc. *ad infinitum.* In practice, this is seldom done due to the difficulty of providing access to the special hardware needed by the VMTSS, since it is not usually made available to the user on a virtual machine. This limitation could definitely be overcome if it proved sufficiently important.

It is possible for a virtual machine to have a richer configuration than the machine on which it is being run since, for example, the amount of virtual memory and number of input/output devices are not limited by the physical configuration. Programs requiring very large amounts of core memory or input/output capability can be run and tested on a virtual machine even though the available real machine does not have sufficient memory or I/O devices.

In general, VMTSS are more modular, easier to write, and better protected against software failure. There are usually three rings of protection on a VMTSS (as opposed to two on most CTSS):

- *Control program, which creates the virtual machine;*
- *Monitor system, which actually runs in a simulated supervisor mode;*
- *User programs.*

Whereas most of the effort in a CTSS is spent in the development of the secondary file system and the translators, the VMTSS only requires the control program to create virtual machines. Any available monitor that exists for the real machine equivalent can be run on the virtual, or a specially-tailored monitor can be developed and tested and debugged on the virtual machine.

Experiments have shown that the use of virtual machines may even increase the throughput of batch jobs, many of which operate on a "compute then print" basis, and do not balance the hardware usage of a real machine. By running two or more of these jobs at once, some jobs will be printing while others are computing. The over-all effect is that the computer is used more efficiently and, despite the VMTSS overhead, the throughput is actually increased. The potential of this technique is still uncertain, but several clever algorithms have been developed to increase or decrease the number of active batch-running virtual machines to optimize machine usage. In principle, this technique is the basis for the complex multi-processing batch systems such as IBM's OS/360 and GE's GECOS.

Finally, a VMTSS can be used to evaluate and help improve programs. Very expensive hardware monitors and meters have been developed to measure the performance of programs. A VMTSS can collect the same, or more elaborate, statistics as the hardware devices. A user might run a program or monitor systems on a virtual machine just to obtain measurements of amounts of memory used, input/output usage, idle time, and many other relevant parameters.

CONCLUSION

This article is not intended to show that Conventional Time-Sharing Systems are better than Virtual Machine Time-Sharing Systems, or vice versa. In a very broad sense, the following summarizes the characteristics of both approaches:

Conventional Time-Sharing Systems:

- *Computer utility—provide basic functions to many users*
- *Pool of resources*
- *Efficiency.*

Virtual Machine Time-Sharing Systems:

- *Modular development*
- *Medium-scale time-sharing systems*
- *Development of systems programs*
- *Program evaluation and measurement.*

The reader should be aware of the existence of both approaches, and will then be in a position to render a decision as to which system is more appropriate for a specific situation.

BIBLIOGRAPHY

Adair, R.J., et al., "A Virtual Machine System for the 360/40," *IBM Cambridge Scientific Center Report,* May 1966.

"Adding Computers Virtually," *IBM Computing Report,* Vol. III, No. 2, March, 1967.

Corbato, F.J., M.M. Daggett, and R.E. Daley, "An Experimental Time-Sharing System," *1962 Spring Joint Computer Conference.*

Corbato, F.J. and V.A. Vyssotsky, "Introduction and Overview of the Multics System," *1965 Fall Joint Computer Conference.*

Corbato, F.J., et al., *The Compatible Time-Sharing System,* MIT Press, 1963.

Dennis, J.B., "Segmentation and the Design of Multiprogrammed Computer Systems," *IEEE International Convention Record,* 1965.

Fano, R.M., "The MAC System: The Computer Utility Approach," *IEEE Spectrum,* Vol. 2, pp. 55-64, Jan. 1965.

Linquist, A.B., R.R. Seeber, and L.W. Comeau, "A Time-Sharing System Using an Associative Memory," *Proceedings of the IEEE,* Dec. 1966.

O'Neil, R.W., "Experience Using a Time-Shared Multi-Programming System with Dynamic Address Relocation Hardware," *1967 Spring Joint Computer Conference,* Vol. 30.

Sayre, D., *On Virtual Systems,* IBM Watson Research Center, Yorktown Heights, New York.