Time-sharing for OS

by ALLAN L. SCHERR and DAVID C. LARKIN

International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION

The objective of the Time Sharing Option (TSO) was to provide a general purpose time-sharing capability within the existing framework of the System/360 Operating System. General purpose time-sharing includes:

• A natural command language in which remote terminal users define their work.
• Support for basic applications such as text editing, problem solving, and program development and testing in a variety of languages.
• Extendibility—a simple way for terminal users or installation management to add specialized interactive applications to the system.

The ground rules for providing this function within the operating system included:

• Preserving compatibility with current OS data set formats and access methods.
• Following OS conventions for user program interfaces.
• Allowing no degradation of existing OS function because of the presence of time sharing in the system.

TSO is designed for System/360 Models 50 and up, with 512K bytes of main storage to operate both batch and time sharing concurrently. With 384K bytes, either time sharing or batch processing can be active, but not at the same time. TSO will also operate on the System/370 Models 155 and 165. TSO will be available in the first quarter of 1971.

TSO FUNCTIONS

We shall look first at the functions and capabilities provided by TSO, and then at how this support was incorporated into the operating system.

The TSO command language is similar in style to some existing terminal-oriented or time-sharing systems, such as CALL/360-OS, Dartmouth BASIC, CTTS, and CP67/CMS. Commands are simple English words or abbreviations: for example, “RUN” to compile, load, and start a program, or “LINK” to invoke the Linkage Editor. Command operands provide flexibility for varied function, but almost all operands normally take a default value, and do not have to be entered. When required information is omitted, the system prompts the user for it. Explanations of command functions and formats are available through a HELP facility. Most messages have been kept brief, but when a novice needs further explanation of a message, he can receive it by typing a question mark.

The command language can be used to:

• Enter, store, modify, and retrieve data at the terminal.
• Solve problems.
• Develop and test programs.
• Execute programs, either interactively, in the time-sharing environment, or by submitting them for batch execution.
• Control most of the operation of the system from a remote terminal.

The installation management determines what capabilities will be made available to each terminal user. A profile of each user’s typical processing requirements and authorizations is kept within the system.

The profile is used to allocate resources for the user when he activates his terminal to start a time-sharing session or “logs on” to the system. A new dynamic allocation facility allows the user to create new data sets, or allocate old ones, during the terminal session. The user does not have to specify these data sets in advance, as he would in a batch environment.
Text and data handling

Commands are available to enter, store, edit, and retrieve data sets consisting of text, data, or source programs. For example, the text for reports can be created, stored, edited, and printed in a format of the user's choosing at the terminal. Data sets can be edited by referring to each data record by a line number, or by referring to the data by context. For instance, a particular string of characters, say a symbol name, can be changed to another string, in a single line or wherever it occurs throughout a program, with a single command.

Problem solving

Three language processors specially designed for mathematical problem solving by users who are not necessarily professional programmers are available under TSO. Two are components of the Interactive Terminal Facility (ITF):

- ITF: BASIC, a simple algebra-like language easily learned by anyone familiar with mathematical notation.
- ITF: PL/I, a subset of the full PL/I language, that provides a powerful conversational language. ITF: PL/I statements can be executed line-by-line, as they are entered, or collected into procedures and subroutines for later execution.

An error in either ITF language can be detected as soon as the incorrect statement is entered, allowing the user to correct it before going on. An interactive debugging feature allows the problem-solver to test his procedures in ITF: BASIC and PL/I using his own statement numbers, labels, and variable names.

The third problem-solving language is Code and Go FORTRAN, a version of the full FORTRAN IV language with some extensions included for the time-sharing environment. Fields in the FORTRAN source statements do not need to begin in particular "columns" of the statement, but can be entered free-form. A new list-directed I/O statement is provided to simplify access to the terminal from an executing FORTRAN program. The FORTRAN syntax checker and the DEBUG package are provided for statement scanning and program testing.

Program development

The programmer is able to invoke from his terminal any processor or most programs written to operate under the operating system. In particular, he can call FORTRAN, COBOL, PL/I, ALGOL, assembler language, and other OS-compatible language processors for his source programs. Any of these languages can be used to write interactive programs, since standard OS sequential access methods (BSAM and QSAM) can be used for I/O directed to the terminal.

Some language processors specially modified for the terminal environment are available with the command language. These include an assembler (F), FORTRAN IV (G1), the PL/I Optimizing Compiler, and American National Standard (formerly USAS) COBOL Version III. The modifications to these processors include prompters, to ask the user for compilation options, and specially formatted diagnostic messages and listings designed for the terminal. Because of the compatibility between the batch and time-sharing environments, programs can be developed and tested at the terminal for eventual batch execution. Source language syntax checking is provided for PL/I (F) and all levels of FORTRAN, either line-by-line, as the program is keyed in, or by whole programs.

The TEST command allows programmers to start and stop programs, inspect and change the contents of main storage and registers during execution, trace program flow, and display the value of variables during execution. The TEST command can be entered whenever any program terminates abnormally, or whenever a user interrupts an executing program with the terminal attention key.

Command language extension

The TSO command language is open-ended; any user can add commands to the existing set. Each command entered from the terminal invokes a command processor to perform the function associated with the command name. Command processors are unprivileged programs, and any program can be defined as a command by adding the load module to a command library. Commands can be added to the system command library to be available to all system users, or to a private command library, to be available to only one user or a subset of the system users.

A set of service subroutines is available for use by all command processors written in PL/I or assembler language. These routines implement common functions and ease the task of creating new command processors. The functions provided include:

- Command syntax scanning and parsing.
NEW CONTROL PROGRAM FUNCTIONS

TSO has been implemented on the MVT configuration of the operating system, to take advantage of the existing program sharing, subtasking, and storage protection facilities, all desirable in a time-sharing environment. With a few modifications and extensions, the existing job management and task supervision functions were used, ensuring continued compatibility between the time-sharing and batch environments.

For each user logging on to the system from a remote terminal, an OS job is created. A large number of these time-sharing (or foreground) jobs, as well as batch jobs, share the available resources of the system. In particular, the jobs created for terminal users can share a foreground region of main storage. A single terminal job is brought into a foreground region and allowed to execute for a short time slice. The other terminal jobs are saved temporarily on an auxiliary storage device. At the end of the active job's time slice or when it starts waiting for I/O from the terminal, a main storage image of the job is copied out to auxiliary storage, and another job is brought into the region for its time slice. One or more regions of main storage can be defined as foreground regions, and remaining storage can be used for batch (or background) jobs, up to a total of 14 regions.

The process of copying job images back and forth between main and auxiliary storage shown in Figure 1 is called swapping. A dedicated access method supports swapping to 2311, 2314, and 3330 disk storage, and 2301, 2303, and 2305 drum storage. The channel programs are tailored to the individual devices in use. The process can be set up so that two swap devices are used in parallel, doubling the effective swap rate. In case a particular device becomes full, a spill mechanism is provided. For example, if a 2301 drum is normally used for swapping, but becomes full under a peak load, the overflow can be directed to another device.

Before a job can be swapped out of main storage, it must be "quiesced," or forced into an inactive state. I/O requests are intercepted, or allowed to complete if already in progress, and control blocks are "unhooked" from system queues. Most control blocks, including Task Control Blocks (TCBs) and I/O buffers, are kept in a reserved area within the foreground region, and are swapped out along with the job. Only that portion of the foreground region that the job is actively using is swapped out.

I/O buffers for the terminal are an exception to the quiescing procedure. All I/O for the terminal is handled through the Telecommunications Access Method (TCAM). Because of the relatively long periods of time involved in terminal I/O, quiescing is not practical. Terminal buffers are not swapped out along with the job, but are kept in supervisor main storage. Thus a user at the terminal is not aware of the swapping process: his terminal keyboard remains unlocked and he can enter input whether his job is in main storage or not.

CONTROL FLOW

Figure 2 is a generalized diagram of the flow of control among the routines implementing the time-sharing subsystem. These routines are brought into main storage only when the operator starts the time sharing operation; the system can be started beforehand for batch processing. If the operator stops time-sharing, the associated storage is again made available for batch jobs.

At the highest level of control under the MVT supervision routines are the Time Sharing Control Task and the TCAM Message Control Program. The Control Task handles system-wide functions such as initialization and control of foreground job swapping. The Message Control Program handles all I/O for terminals, both for TSO foreground jobs and any other teleprocessing applications that may be present in the system.

Below the Time Sharing Control Task is a Region Control Task for each foreground region. It supervises those terminal jobs assigned to its region, and handles the quiescing function. A LOGON/LOGOFF Scheduler is invoked for each user logging on to the system. This routine builds the internal job control language necessary to define the user's terminal job to the system, using information stored in the user profile and in a cataloged procedure specified by the user.
LOGON invokes, through MVT job scheduling routines, a problem program, called the Terminal Monitor Program, that handles both TSO and user-defined commands. When the user enters a command, the Terminal Monitor Program attaches the appropriate command processor or user program as a sub-task.

Important decision-making functions of the control program have been isolated in a single component called the Time Sharing Driver, shown in Figure 3. Each of the other routines continually passes information on system events to the Driver—time slice end, swap out complete, a job waiting for terminal I/O, etc. From this stream of information, the Driver maintains a picture of the current system workload, and based on this information, makes decisions about what should be done next. The Driver orders actions to be carried out by other control routines through a parameter list interface. The Driver itself is entirely insulated from the rest of the system by the Time-Sharing Interface Program. This design allows installations with interests in special-purpose or experimental time-sharing systems the freedom to replace the Driver with a scheduler of their own design.

SYSTEM TUNING

Time-sharing system performance is often measured by response time at the terminal. Response time is in turn largely determined by three factors:

- The number of users logged on to the system.
- The average user “think” time per interaction (one give-and-take between the user and the system).
- The average service time, per interaction, including processing and swapping overhead.

The second two factors depend on the type of processing the users of the system require. A system set up to handle the relatively short think times and processor times typical of text-editing applications would not perform optimally for a set of users running lengthy numerical analysis programs. To allow for varying installation and user requirements, the TSO Driver accepts controlling parameters from the installation. These variables describe system configuration, such as the number of foreground regions to maintain; they may request the Driver to use one of several algorithms it has for scheduling use of system resources; they may specify constants used in the algorithms. This flexibility allows the installation to tailor the scheduling and time slicing functions to best serve its mix of jobs.

For instance, consider an installation whose users typically do two types of processing—problem solving with one of the conversational languages such as ITF: PL/I, and mathematical calculations that are long-running, compute-bound jobs. This installation will
probably wish to optimize response to the conversational users, rather than use a simple round-robin type of scheduling.

TSO allows the installation to set up multiple service queues for the jobs assigned to each time-sharing region. In this case, two queues could be defined, with the higher priority queue for the conversational jobs. Jobs on the queue could be given a four or five hundred millisecond time slice—enough to complete almost any single transaction in a problem-solving language. A job that doesn’t finish a transaction within a time slice would be shunted to the second queue, where it would be allowed a five or ten second time slice, depending on the number of jobs on the queue. Anytime a job on the higher priority queue is ready to execute—typically when it receives a line of terminal input—it will be swapped in for its time slice immediately, interrupting any second-queue job that happens to be executing. This preemptive scheduling will reduce response time to conversational requests to the minimum, at the cost of lengthening somewhat the time for completion of the compute-bound requests.

Among the scheduling variables that can be set by the installation are the number of service queues for each region, the average service time for each queue, the minimum time slice guaranteed to each job on a queue, the thresholds—either in execution time or program size—that determine to what queue a job is assigned, and the method to be used for dividing execution time among multiple foreground regions and the background regions.

CONCLUSION

With the addition of logic to handle the swapping of terminal jobs in and out of foreground main storage regions, the existing MVT control program capability for multiprocessing was found to be a suitable base for a general purpose time-sharing system and, through the natural command language, the full services of the operating system are available to remote terminal users.

REFERENCES

1. IBM system/360 operating time sharing option: planning for TSO
   IBM Data Processing Division Form GC28-6698 White Plains 1969

2. CALL/360-OS description manual
   IBM Data Processing Division Form H20-0673 White Plains 1969

3. J G KEMENY T E KURTZ
   BASIC
   Dartmouth College Hanover 1966

4. F A CRISMAN ed
   The compatible time sharing system: a programmer’s guide
   MIT Press Cambridge 1965

5. Control program-67/Cambridge monitor system
   IBM Cambridge Scientific Center Cambridge 1968

6. Certain of the functions described are provided as IBM
   Program Products, available for a separate license fee

7. G ESTRIN L KLEINROCK
   Measures, models, and measurements for time-shared
   computer utilities
   Proceedings 22nd ACM National Conference Thompson Book Company Washington DC

8. J M McKINNEY
   A survey of analytical time-sharing models
   Computing Surveys June 1969

9. A L SCHERR
   An analysis of time-shared computer systems
   MIT Press Cambridge 1967

10. A L SCHERR
    Time-sharing measurements
    Datamation Vol 12 No 4 April, 1966