Design and Behavior of TSS/8: a PDP-8 Based Time-Sharing System

AD VAN DE GOOR, STUDENT MEMBER, IEEE, C. GORDON BELL, MEMBER, IEEE, AND DONALD A. WITCRAFT

Abstract—TSS/8 is an existence proof for a small-scale time-sharing system. Design, development, and performance analysis have occurred in quasiparallel. Performance analysis includes two models and two levels of simulation (using SIMULA). The final simulation, an accurate model of the real operating system, predicts the observed behavior.

Index Terms—Disk, PDP-8 computer, performance analysis, Simula, simulation, text editing, time-sharing.

INTRODUCTION

IN MAY, 1967 Carnegie-Mellon University began the design of a small-scale time-sharing system. The principal design goal was a general purpose time-sharing system, but with an order of magnitude decrease in the per console cost.

The design was predicated on giving a user a very short response to a typewriter or teletype input; but the user would have little arithmetic computing capability. Thus, through specialization the system would be used as follows.

1) A preprocessor to larger, more general purpose systems. In this mode all trivial tasks (syntax checking, editing, etc.) would be handled by TSS/8, in effect, at the lowest (cheapest) organizational level. Simple computational tasks (e.g., short BASIC programs) would also be run at the periphery.
2) Stand alone general purpose system. Users not requiring significant arithmetic capability and large primary memory would use the system "as is" (e.g., secondary schools and novice programmers).
3) The basis for developing specialized systems.

Manuscript received June 9, 1969; revised July 7, 1969. This paper was presented at the 1969 IEEE Computer Group Conference, Minneapolis, Minn., June 17–19, 1969. This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F 44620-67-C0058) and is monitored by the Air Force Office of Scientific Research.

A. van de Goor and C. G. Bell are with the Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa.

D. A. Witcraft was with the Digital Equipment Corp., Maynard, Mass. He is now with Logi-Call Systems, Inc., Palo Alto, Calif.

Specialized time sharing systems use the basic framework for terminal and file management (e.g., text editing, hotel reservations, etc.).

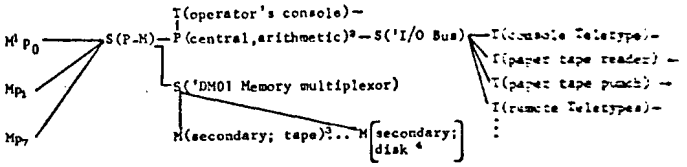
The main constraint of the system is based on the PDP-8. The PDP-8 was selected because of the following.

1) Though very fast, it represents about the smallest existing computer; any work done using it is easily extendable to larger systems.
2) The small primary memory of 4096 words (a memory field) has historically restricted the program size; therefore complete program swapping rates between primary and secondary memories are quite high.
3) PDP-8 computers have existed for a long time. The hardware as well as most of the software is well debugged.
4) A slightly larger computer tends to perform marginally better for compute-bound[1] jobs. (All systems without hardware floating point arithmetic appear to have this problem.) For non-compute-bound jobs a slightly larger computer tends to be more swap-bound[2] because:
   a) more bits have to be transferred because of the bigger words,
   b) software for these machines tends to be bigger (typically 8K words) for about the same function.
5) A PDP-8 computer was available.

The constraints on the design of TSS/8 were the following.

1) Achieve an order of magnitude improvement in the cost/performance ratio over completely general purpose time-sharing systems (e.g., 360/67, PDP-10.)
2) The system should be able to support about 20 on line users.
3) Response time for jobs with low computational requirements (like editing) should be good.
4) The system should be open-ended, i.e., addition of future software should be easy.
5) Existing user programs and system software (like the EDITOR, FORTRAN COMPILER, etc.) should be able to run on TSS/8 with no or only minor changes. This would prevent the necessity of reprogramming the PDP-8 software.
6) PDP-8 hardware changes should be kept low such that it would be relatively easy to change existing PDP-8 processors.
7) The system should be general purpose by providing protection among the users on a device, core memory, and file basis.

[1] A job is considered compute-bound when it does not terminate the allotted quanta of time running.
[2] A system is considered to be swap-bound when the processor is idle and waiting for jobs to be brought from secondary memory into primary memory.



[1] M(primary; core memory; 4096 words; 12 bits/word; 1.5 μs/word)
[2] P(arithmetic, central; 1address/instruction; 1 ~ 2 words/instruction)
[3] M(secondary; magnetic tape; ~ 3 × 10^6 bits; .33 μs/word)
[4] M(secondary; fixed head disk; ~ 5 μs/w; taccess: 0 ~ 34 ms; ~ 1.1 × 10^6 words)

where
P/Processor; M/Memory; T/Terminal; S/Switch

Fig. 1. PDP-8 hardware configuration.

HARDWARE

A block diagram of the hardware configuration is given in Fig. 1. The PDP-8 is a small, general purpose computer. In its standard configuration it has a core memory of 4096 (12-bit) words with a 1.5μs cycle time. The memory is expandable in increments of 4096 words (called fields) to a maximum size of eight fields.

The S(P–M), a processor-memory switch, allows the processor or the DM01 switch to have access to the primary memory fields. The S(DM01) switch allows the high-speed secondary memory, two tape units and a disk, to access primary memory.

The low-speed devices are connected via the S(I/O) bus. Data transfers to and from these devices is on a character-by-character basis under programmed control.

MEMORY PROTECTION

Because user programs written in machine language are allowed, an effective memory protection scheme had to be implemented. The instruction set for the PDP-8 makes addressing inside a 4096-word memory field easy. Crossing field boundaries either for data or instructions is relatively difficult, however, because it has to be done by special change data and instruction field instructions (CDF and CIF) which are in the input-output transfer (IOT) class. A simple memory protection scheme is obtained by only allowing a user to access data within a single field. Any accesses outside the field cause a program trap.

MODES OF OPERATION

When running programs in a time-shared environment, not all instructions from a user program can be executed directly. Some have to be analyzed by the monitor for possible memory protection violation, device assignment, etc. Hardware was added to allow for this, giving the processor two modes of operation.

1) MONITOR *mode*: In this mode all instructions are legal and will be executed by the hardware directly.

2) USER *mode*: User programs are run in this mode. Certain instructions and classes of instructions are illegal here because of their interference with the time-shared operation. The illegal instructions are:

HLT halt
OSR inclusive OR switch register with AC
IOT all input-output transfer instructions.

When any of the illegal instructions occur, a flip-flop is set which when the interrupt is enabled will cause a trap. This will automatically tranfer control to location 1 of memory field 0 and puts the system in monitor mode (field 0 is the field where the monitor resides) thus transferring control from the user's program to the monitor.

Instructions were added and some were changed to manipulate the mode and trap flip-flops and to save or restore the mode of operation upon entering or exiting an interrupt state.

Memory Configuration

Basically the memory is divided into two classes:

1) primary (core) memory (M_p) for programs being executed by the processor;
2) secondary memory (M_s) for programs which are not currently active or that are waiting to be executed and data files.

Primary Memory

The primary memory is a standard core-memory, and holds the programs the central processor (P_c) is interpreting. In determining the amount of primary memory, the tradeoff between memory size and system performance has to be considered. Initially it was assumed that field 0 would be sufficient to contain all of the resident part of the monitor. The lack of enough buffer space for I/O for more users necessitated two fields for the resident monitor. Because of the imposed limitation on the size of a user program, one field is needed to run either user programs or parts of the nonresident monitor. In order to allow for simultaneous swapping and processing, at least one more field is required.[3] For these reasons the minimum system has four fields of core memory of a maximum possible eight.

Secondary Memory

The Secondary Disk Memory: The secondary disk memory serves the following functions: swapping device, file storage device for system programs (nonresident monitor, etc.), and temporary storage for user programs. Because of its use as a swapping device, it was very important to have a low access time (i.e., high number of revolutions and fixed heads) and a high transfer rate. For this reason and for its low cost per bit, the Burroughs model 9370-2 disk was selected. Its characteristics, however, were such that it necessitated a rather elaborate interface. This interface maps the semi-continuous address space of the computer into a segmented BCD-addressed address space of the disk; has a buffering system which can be generalized to arbitrary long waiting delays; and has hardware to detect out of range disk address requests and to detect and prevent the execution of erroneous commands. The average access time is 17.3 ms, the transfer rate is 1 word per 5.0 μs, and the capacity is 1.1×10^6 words.

Magnetic Tape Memory: At least two magnetic tape units are connected to the system. When TSS/8 is used as a "stand alone" time-sharing system, they are used as "backup" for the disk and as the "mass storage" device for users to enter their files onto the disk.

User Machine

Looking at Fig. 2, several levels of machines can be identified. The absolute machine is the collection of the TSS/8 hardware components. By adding the monitor to the system, the virtual machine is obtained, which allows time-sharing and adds a set of powerful software-interpreted instructions. Adding the library programs (e.g., the EDITOR, the FORTRAN COMPILER, BASIC) creates the user machine. This is the machine the user sees from his console. The user in turn, by writing programs, can create new machines.

Instructions to the virtual machine allow a user's program to transfer files. In this way a user can have programs of almost arbitrary size which he can control as a paging environment. Some of the TSS/8 software, like BASIC, uses the paging feature.

TSS/8 Monitor

The monitor consits of the following two parts.

1) The resident monitor is a collection of routines which have a high frequency of use or are directly necessary to keep the system running (e.g., disk service, teletype service, scheduling, buffers). It occupies two fields.
2) The nonresident monitor is divided into two parts: the file handler which controls the hierarchial file system, and the system interpreter which interprets user's commands. Each part is about one field in size.

Fig. 3 shows the cooperation of the various programs and parts of the system. On top are the different jobs (only one is shown) and the different hardware devices (only one is shown). All communications of jobs with devices, and vice versa, is done through traps and interrupts. The monitor never looks for work to do. Work is signified by some outside event (e.g., time up, a character input).

Job-device communication is through the common pool of buffers which are dynamically linked together. The buffers are individually eight words long, each capable of storing ten characters. To illustrate how data communication is handled, let us look at a job requesting teletype input. The running job first gives a special

[3] It was expected that the improvement in response time by adding one field would be much more than from any field added thereafter. Simulation results later on show this.

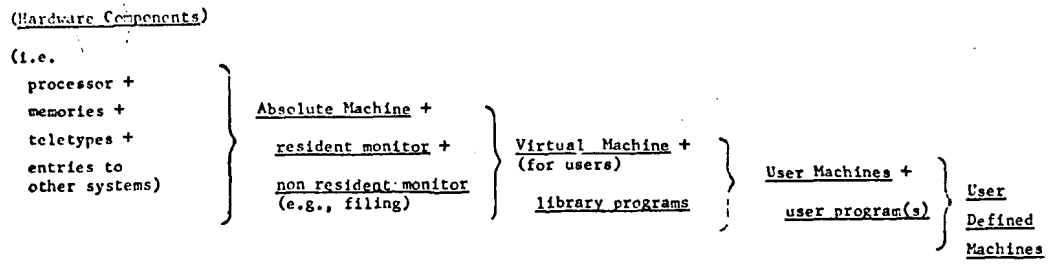


Fig. 2. Levels of machines from the hardware to user defined machines.

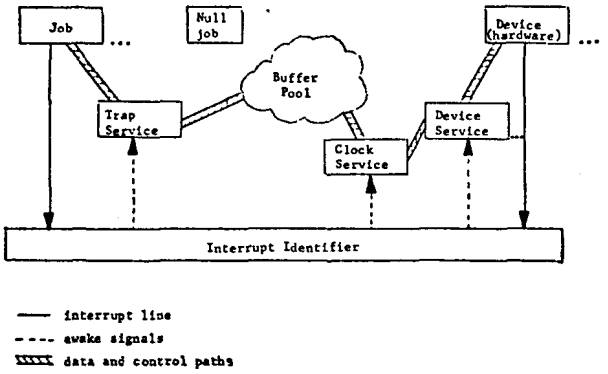


Fig. 3. Cooperation of tasks.

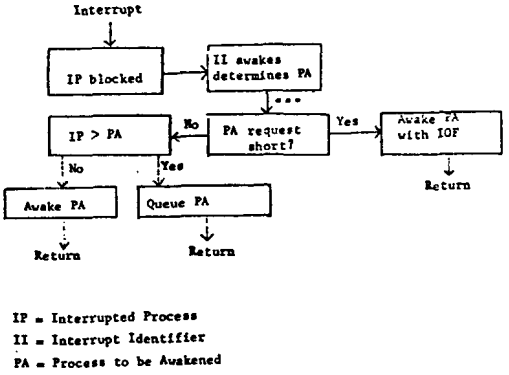


Fig. 4. Interrupt handling.

instruction (from the IOT class) which is trapped to the monitor. The interrupt identifier (II) then activates the correct trap service routine. This routine checks for characters in the input buffer (associated with the job) and takes a character from the buffer and transfers it to the appropriate place in the user's program. In this way IOT instructions are simulated.

Teletype device input is handled as follows: When a character is entered from a keyboard, an interrupt occurs. The interrupt identifier activates the appropriate device service routine which takes the incoming character from the keyboard and stores it in a one word internal buffer associated with the device. Every 90 ms (timed by the clock interrupt) all internal buffers are scanned and if a character is found, it is transferred to a buffer associated with the device. Teletype output is handled similarly but in reverse order.

Interrupt Handling

Fig. 4 shows how interrupts are handled in TSS/8.[4] An interrupt causes the current process to be interrupted (IP blocked). The interrupt identifier (II) establishes the identity of the interrupt. When the process to be activated (PA) is very short, it is executed immediately and interrupts are disabled. Depending on the time the PA request requires, either the PA will be awakened or the PA will be queued, to be awakened after higher priority interrupts have been serviced. The priority is mainly determined by the characteristics of the process requiring service.

[4] Reentrant programs cannot easily be written. The subroutine calling instruction JMS places the return address in the subroutine. Also, the PDP-8 does not have index registers, thus it is difficult to access different data areas associated with a common process.

Scheduling

The present scheduling algorithm is essentially round robin. Each user is served in a fixed order, independent of the time his request is received. A user requiring service is run for a fixed quantum of time, 100 ms, and the next user requiring service is run. If a user requires input-output, his time is terminated. The order in which the users are examined is fixed. Users with file-transfer requests are served first. A user is swapped out only when he is done running and another user has to be swapped in or when another user with a higher position in the round robin scheduler wants to be swapped in.

Performance Analysis

The following section will analyze the system for the case of all users doing text editing. The edit times for different commands and line lengths of a standard PDP-8 editor were measured. It was found that all commands took about the same time (typically 2.0 ms). The time to input or output lines was found to be linear with the number of characters and was about 2/7 ms per character.

A feasibility calculation was made before the system was built. If we assume continuous program swapping between core and disk, the following times are available per user.

Average swap time		71.0 ms
Swap overhead (due to cycle stealing)	$44 \times 0.3 = 13.2$ ms	
Average edit time per user	12.0	
Total		25.2 ms
Left for system overhead		45.8 ms

The example shows that under the assumptions made, 64.5 percent overhead is allowable.

A simulation was made (in ALGOL) and the effects of queuing due to swap and process wait were studied. The results of this simulation corresponded with what was expected and showed under simplified assumptions that acceptable response times[5] could be obtained.

Fig. 5 shows a simplified model of the second simulation program (written in SIMULA). The "think time" is the time between the completion of a service and request for a new one. Think times for this were extracted from JOSS statistics [1] and from Scherr [2]. These times are taken from a negative exponential distribution with an average of 25 seconds. The queuing delay is the time spent waiting for core space and processor time. This depends heavily on the number of fields and the characteristics of the disk and processor. The processing delay is the time it takes to process the job. It is initially allowed to be 100 ms, but it may be shorter for interactive jobs and longer for compute-bound jobs. The time the processor can spend on processing user jobs is less than the elapsed time because of overhead. This overhead is the clock service overhead (7 percent), plus the swap overhead (30 percent while swapping), and the character handling overhead to and from the I/O devices, which varies with the number of users.

The requested processing time for a job is the time the job would take on a standard PDP-8 given a certain command and line length (measured) plus the time spent in the monitor due to servicing trap interrupts. The system transfers data between the user and the monitor on a character-by-character basis; a line-by-line transfer basis would reduce the monitor overhead for each new job.

Fig. 6 shows the simulated response times with two user fields and the Burroughs disk. The 90 percent curve indicates the elapsed time after which 90 percent of the users are served. The increase in response time with more users occurs because:

1) the overhead increases with the number of users (this increases the elapsed time of a running program), and
2) there is an increase in probability that the user requesting service has to wait in a queue.

Fig. 7 shows the simulated response times of the system as a function of the number of users and the number of user fields. In the case of one user field ($U=1$), the response times are much worse than those for two or more user fields. This is due mainly to the fact that swapping and processing of user programs cannot be overlapped. After 16 users the effect of queuing for swapping becomes visible. With two user fields, the preformance is improved considerably. Adding more user fields only helps for a small number of users because the probability of not having to swap (because the user is already in core) increases. With six user fields and four users, no swaps are necessary. Fig. 7 shows that the performance increase by adding the second user field is greatest; more user fields contribute only marginally to the performance increase.

Fig. 8 shows the response times of the system with one, two, and six user fields and the DEC DF08 disk compared with the CMU system (two user fields and the Burroughs disk). The DF08 disk has an average access time of 16.6 ms (Burroughs disk is 17.3 ms) and a transfer rate of one word per 16.6 μs (Burroughs disk is one word per 5.0 μs). Fig. 8 shows that with one user field the system becomes swap-bound very rapidly. With two user fields the system's response time is much worse than the CMU system (broken line).

Fig. 9 shows the performance of PDP-8 based,

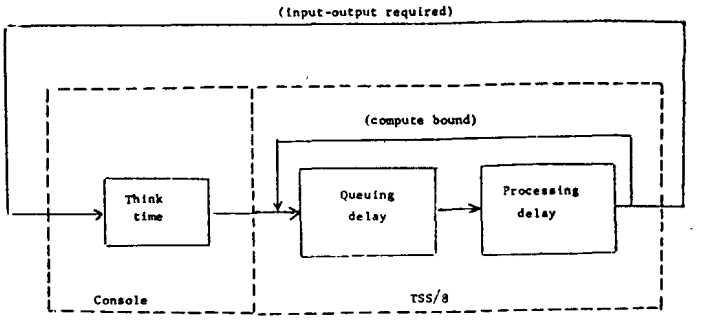


Fig. 5. Simplified simulation model.

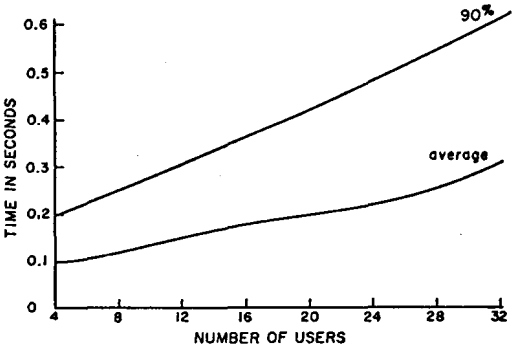


Fig. 6. User average and 90 percent response times of TSS/8.

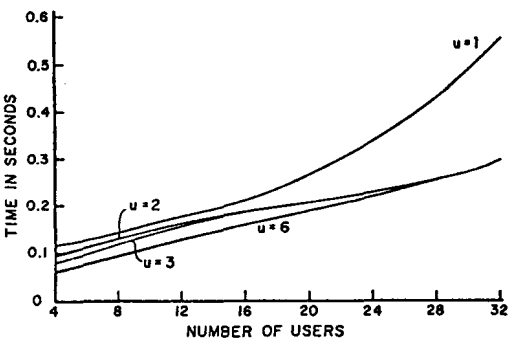


Fig. 7. User average response times as a function of the number of user fields.

[5] The response time is defined as the elapsed time between a service request and the request completion.

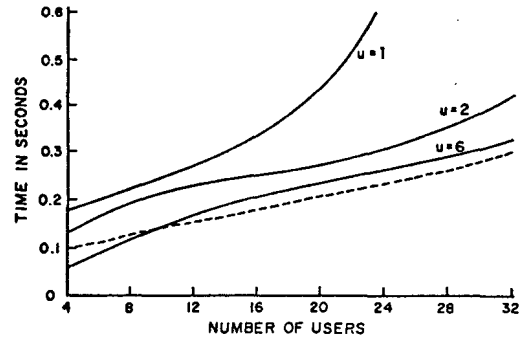


Fig. 8. User average response times with DEC DF08 disk as a function of the number of user fields.

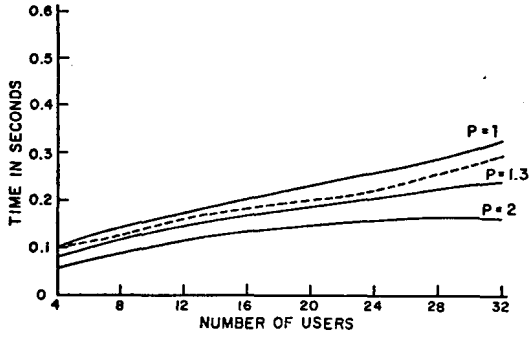


Fig. 9. User average response times as a function of processing power of 16-bit processors with four thousand word programs.

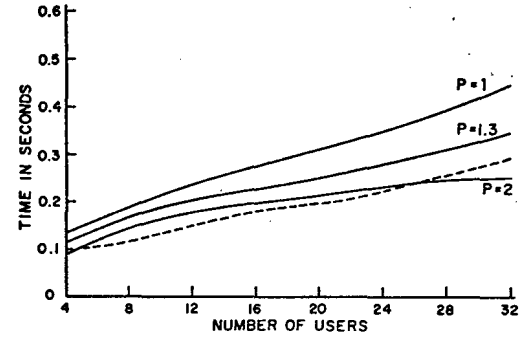
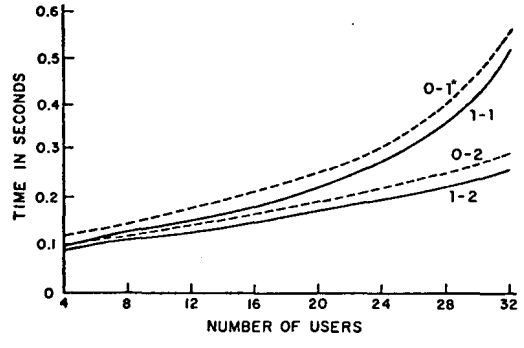


Fig. 10. User average response times as a function of processing power of 16-bit processor with eight thousand words programs.



* X−Y means: X = number of user fields to be emptied; Y = number of user fields.

Fig. 11. Average response times with scheduler modifications.

Burroughs disk system compared with systems of different processor power (P). The plot assumes a 16-bit computer with instruction set improvements corresponding to 1, 1.3, and 2.0 times a PDP-8. We also assume the same disk. The swap time will increase by a factor of 33 percent because more bits have to be transferred. The assumption was made that the 16-bit machine has software written to operate in a 4K words environment. Fig. 9 shows that when $P=1$ the system performs worse because of the increased swap time (as could be expected). The crossover point lies somewhere between $P=1$ and $P=1.3$. With $P=2$ the system performs consistently better than the CMU system.

Fig. 10 is similar to Fig. 9, except for the assumption that the system has software written to operate in an 8K words environment (as is very often the case with larger machines). Because of the large increase in swap time, the system only performs better when $P=2$ and there are a smaller number of users. With a large number of users the power $P=2$ is always offset by the increased swapping time.

Fig. 11 shows the effect of a small change in the scheduler. The new scheduler will try to keep one user field free, i.e., when a user is served and the number of free fields is 0, a user will be swapped out. In this way a user requesting service can be immediately swapped in directly. With a small number of users, however, this might be offset by a decrease in the probability of a user being in core.

Conclusions

The basic system design objectives were realized. The simulations indicate that the hardware structure of the system, for doing text editing, is near optimal, although certain minor modifications can still increase the performance. As a direct consequence of the simulations, some improvements have been made. Also, possible improvements in the operating system can be studied.

Although the simulation results have not been compared precisely with empirical results, an eight user system has been checked and roughly agrees. Subsequent work will make such a comparison and also account for discrepancies.

Acknowledgment

Although the system was initially conceived and designed at CMU, the final design and implementation was done at Digital Equipment Corporation. Without a real system to show feasibility, simulation would have been an exercise of questionable value. CMU appreciates the loan of a PDP-8 for both a user and a performance measurement system.

References

[1] G. E. Bryan, "JOSS: 20,000 hours at the console—A statistical summary," the RAND Corporation, August 1967.
[2] A. L. Scherr, *An Analysis of Time-Shared Computer Systems*. Cambridge, Mass.: M.I.T. Press, 1967.