A FACILITY FOR EXPERIMENTATION IN MAN-MACHINE INTERACTION*

W. W. Lichtenberger
Computer Center and Department of Electrical Engineering
University of California, Berkeley

and

M. W. Pirtle
Computer Center
University of California, Berkeley

INTRODUCTION

The broad objective of the project of which the work reported below is a part is to explore and develop techniques in man-machine interaction. The situation involving a person interacting with a machine in the performance of a task generally requires that the person be on-line with the machine. The amount of machine time wasted while the person is carrying out his part of a task may require, even in an experimental situation, a time-sharing system.

The Role of Time-Sharing in the Project

The time-sharing facility described below was constructed

(a) to develop and test some ideas in time-sharing a digital computer and

(b) to develop a useful facility for a variety of experimenters in man-machine interactive areas.

It should be emphasized that the time-sharing system, although general in nature, is an experimental system intended to give great flexibility and fast response to a limited number of users. (In particular, it is not designed to serve a large number of users over a broad spectrum of problems as a utility approach1,2 to time-sharing.)

Design Philosophy of the Time-Sharing System

Because of the variety of tasks to be performed by the users of the system it was felt that each user should be given, in effect, a machine of his own with all the flexibility, but onerousness, inherent in a "bare" machine. It was also felt that additional features should be provided to enable the user to reduce the onerousness, perhaps at the cost of flexibility, to the extent desired. Thus each user is given a "copy" of a slightly modified SDS 930 with 16K of fast memory. This "copy" differs from the normal Scientific Data Systems (SDS) 930 only in (1) the obvious impairment of certain real-time capabilities which result from the necessity of running programs for short, nonregular intervals, (2) the substitution of a set of instructions which initiate system-controlled input/output for the standard I/O instructions, and (3) the addition of many new (software-interpreted) instructions along with various system routines and a number of large-scale subsystems.

In order to cut down response time it was felt desirable to have more than one program present in memory and to swap with auxiliary memory only.

---

*The work reported in this paper was supported by the Advanced Research Projects Agency, Department of Defense, under Contract SD-185.
when necessary. In practice, swapping occurs relatively frequently.

For further economy of both active and auxiliary storage, it was felt desirable to provide for common programs — single copies of programs shared by more than one user. Common programs are pure procedures with a unique copy of temporary storage assigned to each user. Many system routines are written as common programs, as are some large-scale systems.

The major part of the system executive, for example, is a common program. Since this part of the executive was written as if it were dealing with only one user it was simpler to write and to debug. Furthermore, the same part of the executive would require no changes if more central processors were added to the system. Simplicity, small size, and flexibility were among the goals of the system executive, and all of these goals have been to some degree achieved.

The project objectives made it desirable to base all input/output around the remote consoles as much as possible and to minimize the role of more standard I/O equipment (cards, line printers, magnetic tapes, etc.). The user is given mass storage in the form of either word-addressable or sequential files and a generalized file-handling capability. Files are independent of any peripheral I/O device or storage medium and are addressed homogeneously regardless of their current position in the storage hierarchy. The file-handling facilities augmented by comprehensive editing programs provide the users at remote consoles the ability to manipulate information conveniently within the system.

SYSTEM DESCRIPTION

Local Units

As shown in Fig. 1, the system is built around a modified SDS 930 central processor and a main memory consisting of two 16K modules of core storage. The main memory is augmented by a large capacity drum which is in turn augmented (it is anticipated) by a mass storage unit. Filling out the list of local components are the teletype, multiplexor, the I/O processor, a 45KC magnetic tape unit, and a 200-cpm card reader.

A cursory description of the SDS 930 central processor and its modifications for time-sharing is the subject of several of the following sections of this paper; therefore, the various memory devices will be given first attention.

The main memory consists of two modules of 16,-384 words. The words are 25 bits in length (including a parity bit), and the memory cycle time is approximately 1.9 microseconds. Each of the modules is connected to three memory buses. These buses have fixed priorities, with the drum I/O processor, the general I/O processor, and the CPU connected to buses having progressively lower priority. To accommodate the high data transfer rate of the drum (525 x 10^3 words per second), the timing for the main memory units and for the CPU are derived from a timing track on the drum, and the memory addresses are interleaved between the two modules. By having the drum I/O processor reference the modules alternately, the CPU can operate at approximately 65 percent capacity during drum transfer operations. This assures that interrupt processing capability is preserved and that a significant amount of processing accompany the data transfer operations.

The next level of storage is in the form of a magnetic drum having a capacity of 1,376,256 words and a data transfer rate of approximately 525 x 10^3 words per second. The drum is word-addressable to facilitate handling files and has a storage format commensurate with its function of swapping programs, or parts thereof, between the drum and the main store. This format provides 84 bands of 16K words, with each band divided into 8 segments of 2K. Each of the segments is separated by a gap of sufficient length to allow the drum I/O processor to accept an instruction between segments. This feature facilitates the scatter reading and writing necessitated by the memory page technique employed (cf. Memory Relabeling and Protection, below).

The next level of storage will be in the form of a mass storage unit having a capacity in excess of 10^8 words and an access time not greater than 0.5 second. This unit will have some type of interchangeable cartridge, thus providing yet another level of storage having a still greater capacity and access time.

The teletype multiplexor consists of 16 input and 16 output buffers along with control logic to notify the computer of buffer conditions requiring service. The general I/O processor is another device having

---

*The drum is being produced to local specifications by Vermont Research, North Springfield, Vermont.
Figure 1. Configuration of equipment.

**NOTE:** These units are not yet completely specified. It is anticipated that they will be operational by spring of 1966.
very little complexity. It is made up of a central control channel and several independent subchannels. The subchannels operate concurrently and may retain their requisite information (word count and current address) internally or in main memory.

The final two local units, the 45KC magnetic tape unit and the 200-cpm card reader are used by the occasional user who desires to transmit data between this and some other system via one of the media processed by these units. In addition, magnetic tape is being used temporarily as a secondary storage medium. This function, of course, will be assumed by the mass store upon its acquisition.

**Remote Units**

The remote units include 10 model 33 and model 35 teletypes, 2 different types of CRT display-keyboard units, and a PDP-5 with a CRT display and Rand Tablet.

The teletypes are operated in the full-duplex mode with each character being individually processed by the CPU. That is, the teletype keyboard and printer are treated as independent units by the system I/O programs, and, for further flexibility, the input characters to the CPU are processed on a character-by-character basis rather than on a message basis. This procedure consumes processor time (approximately 300 microseconds to input and echo a character, and 200 microseconds to output a character), but experience indicates that the capabilities thus obtained justify the processing expenditure.\(^*\)

In fact, the full-duplex, character-by-character I/O philosophy will probably be carried over to the alphanumeric CRT display-keyboard stations. These stations will consist of CRT display units driven by a central buffer and control unit located in the vicinity of the stations and keyboards which will communicate with the CPU via the central control unit.

The second type of CRT display-keyboard stations will be similar to the Culler-Fried console.\(^5\) These consoles employ a storage tube display with a script generator, i.e., a generator which produces short vectors of length and angle specified by the input character.

For research efforts requiring a more capable CRT display system, a remote unit is provided which consists of (1) a PDP-5 processor with 4K words of memory, (2) a CRT-display unit\(^6\) with character, vector, and script generators,\(^7\) and (3) a RAND tablet.\(^7\) These three major components are integrated into a unit which provides the researcher with a large amount of flexibility with regard to both the use of the present system and the addition of supplemental equipment to provide still greater capabilities.

In the present system, the PDP-5 functions as a buffer and controller for the CRT display and the RAND Tablet and performs some elementary operations such as smoothing the data input from the Tablet. All computations are performed in the central computer.

**FEATURES OF THE MODIFIED 930 USED FOR MULTIPROGRAMMING**

**Modes**

The role of the system monitor is unique among programs residing in the machine. Reflecting this fact, the 930 has been modified to operate either in monitor or in user mode. Monitor mode permits the monitor the use of privileged instructions and unrestricted memory. The function of user mode is the subject of the following sections.

**Protection of the System from User Action**

It is necessary to protect the system and all other users from certain actions of any user. Such actions include the execution of instructions which: (a) affect peripheral equipment, (b) halt computation, (c) interfere with rapid response to interrupt requests, or (d) access unassigned memory locations. User actions of type (a) and (b) are handled merely by trapping the offending instruction (the term *trap* means an interrupt of highest priority). Instructions which fall into this category are called *privileged instructions*. When the 930 is in user mode an attempt by a user to execute a privileged instruction will result in the execution of a no-op followed by a transition to monitor mode and a transfer of control to a memory location unique to the illegal instruction trap.

\(^*\)It is calculated that 16 teletypes executing independent input and output simultaneously at full speed consume 8 percent of processor time.

\(^5\)The display is being produced to local specifications by the Burroughs Corporation, Ann Arbor Laboratory, Ann Arbor, Mich.
Type (c) actions are treated by permitting interrupt requests to preempt execute and indirect address operations. If either of these operations are in process when an interrupt request occurs, the operation is aborted and the interrupt request acknowledged. Upon return to the interrupted program the aborted instruction is begun anew. In this way, infinite indirect address and execute loops in a user's program cannot halt the system.

The solution to user actions of type (d) is related to memory relabeling and is discussed in the following section.

Memory Relabeling and Protection

The memory relabeling or paging technique adopted provides both for dynamic program relocation and memory protection with no increase in memory access time. The technique was adopted initially because it eliminates the need to move information around within the fast memory in order to provide space for incoming programs and because it easily provides memory protection. Other important uses (discussed later) became apparent as the system progressed. The implementation consists of 8 relabeling registers of 6 bits each laid out in 2 registers as shown in Fig. 2.

![Figure 2. Physical arrangement of relabeling registers.](image)

For purposes of relabeling, the memory is divided into 16 pages or blocks. Calls are addressed by block number and location within a block as specified by subfields of the address.

The block or page size is fixed at 2K by the 11 bits of the least significant part of the address (which may be thought of as an address within a page or a page address). Because the address field of the 930 contains 14 bits, only 16K or 8 pages are permitted each user. The upper 3 bits of the address constitute the page number \( R_i \) which may be different from time to time as the program is moved in memory (cf. Fig. 3). Because of the spatial relationship of the page number and page address, the user is not conscious of page structure. Note that relabeling permits user's storage to be located in noncontiguous blocks while appearing to the user and to the machine to be connected.

Of the 6 bits in a relabeling register, the lower 5 are used for the actual page numbers. Addresses after relabeling are therefore of length 16 bits, permitting as much as 64K of fast memory in the system. The sixth bit in a relabeling register designates a read-only block. The facility to have read-only storage enables users to share subsystems directly without interference and without the necessity of calling the monitor constantly to change relabeling.

Absolute memory protection (i.e., protection against any reference) is accomplished by using \( R_i = 0 \) to mean that no memory is assigned to the page \( i \). Any reference to a cell whose relabeling register contains zero is trapped.

Figure 4 shows a 6K memory allotment distributed in 2K blocks at 24000, 64000, and 14000. The block at 14000 is read-only. It may be seen that references to any location greater than 13777 will point to one of the relabeling registers 3 through 7, causing an out-of-bounds trap. The choice of the combination \( R_i = 0 \) prevents absolute memory locations 00000 through 03777 from being used for user programs, but this is of no consequence since that area is part of the monitor. Note that the user may transfer control, for example, to his locations 10000 through 13777, but an attempt to store information there will cause a trap.

Relabeling is always performed in user mode. It is also possible to invoke relabeling for individual instructions in monitor mode. In accessing memory to obtain the effective address of an instruction, any word encountered with Bit 0 set causes relabeling to apply immediately and for the duration of the instruction. Thus an instruction with Bit 0 set causes relabeling of its address, while an instruction with a chain of indirect addresses produces relabeling at the first instance of Bit 0 set. In the latter case subsequent references come from relabeling memory.

Mode Transitions

During the design of the modifications to the 930 it was felt desirable to make the transitions between modes as simple and as natural as possible. In par-
ticular, it was felt that there should be provided sufficient hardware capability to insure that the interrupt routines could be independent of the mode of the machine at the time of the interrupt and that the system routines explicitly called by the various programs should not require software interpretation of

- the source of the calling program (monitor or user),
- the location of the call,
- the location of the arguments, and
- the specific action requested.

Readers who have some experience in implementing high-speed interrupt or I/O routines will perhaps appreciate the spirit of the above objectives.

Transitions from user to monitor mode occur only upon (1) an interrupt or trap, or (2) execution of a system programmed operator (cf. the following section). The user has no interest or direct concern over category (1) and does not think explicitly of the instructions in category (2) as changes of mode.

The system initiates the transition from monitor to user mode by transferring control to a user program. Specifically, a control transfer calling for relabeling causes a transition to user mode.

In order to provide closure in the above scheme, the previous mode of the machine is stored as a single bit in the subroutine link of both interrupt and system programmed operator routines. The bit used is the same as that used to designate relabel-
Figure 4. Example of a 6K memory allocation.

System Programmed Operators

Input/output instructions are among the privileged instructions not allowed in the user's machine. The system must do all I/O for the user, and he must therefore be able to call the system for such services. Also the system executive requires many complex services, some of which are potentially useful to a user. Such services should be provided by system calls. The system programmed operator (SYSPOP) is the device by which such calls are accomplished.

The SYSPOP is an extension of a normal 930 feature—the programmed operator (POP). POP's are invoked by setting a bit in the instruction word, and they function as a special kind of subroutine call. In the execution of a POP the op code bits are not decoded in the usual way. Instead, they are taken to be the relative address in a transfer vector beginning at 0100, to which control is transferred. At the same time the contents of the program counter and status of the overflow indicator are stored as a subroutine link in location 00000. The indirect address bit of this link is set as well. Single arguments
Table 1. Summary of 930 Modes and Their Effects.

<table>
<thead>
<tr>
<th>Monitor Mode</th>
<th>User Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All 930 instructions may be executed.</strong></td>
<td>All 930 instructions <em>except</em> privileged instructions (I/O, halts, etc.) may be executed.</td>
</tr>
<tr>
<td>Normal addressing applies to references without Bit 0. References with Bit 0 set call for relabeling.</td>
<td>Relabeling applies to all memory references.</td>
</tr>
<tr>
<td>Transition to user mode occurs upon executing a transfer whose effective address is relabeled (i.e., Bit 0 is set).</td>
<td>Transition to routine mode occurs on all interrupts or executions of SYSPOP's. Bit 0 is set at the formation of the first subsequent subroutine link so that memory access to data will be relabeled and so that control will revert to user mode.</td>
</tr>
</tbody>
</table>

or the location of a list of arguments can thus be transmitted to the body of the POP indirectly through the link in 00000. The format of a POP is the same as that of a normal routine instruction, hence the POP is a convenient way of simulating nonexistent machine instructions.

The 930 was modified so that a POP executed in user mode with Bit 0 set causes transition to monitor mode. The user thus has the facility to jump to a standard transfer vector in the system. Note that the user may still implement his own POP's. The SYSPOP's, however, give the user 64 new "machine instructions" which do not require his memory allocation or other attention.

The reader should note that by having the mode stored in the relabeling bit of the link, all four objectives for system routines listed in the preceding section are accomplished; modes are completely invisible to interrupt and system programmed operator routines. Most importantly, interrupt routines *take no more time* and in fact are no different from similar routines in a non-time-sharing system. Furthermore, the overhead associated with calls to the system (SYSPOP's) is only 4 memory cycles.

**GENERAL DISCUSSION OF SYSTEM FEATURES**

The features of the system described above came into being through a compromise between that which is desirable and that which is feasible, to implement time-sharing on a machine basically not designed for time-sharing. Some of the features have been shown to be surprisingly compact and effective, however. For example, the SYSPOP provides a simple but versatile system call. Also, relabeling is not only useful for dynamic storage allocation but provides the basic means by which common programs can be constructed.

**Method of Writing Common (Re-Entrant) Routines**

By its very nature, a common routine consists of (1) *a pure procedure*—a body of code which is not self-modifying and in which there is no temporary storage—and (2) one or more copies of all temporary storage associated with the routine. To implement a common routine, one allocates all temporary storage—the data block—to a unique block or blocks of memory different from those blocks of the procedure body. Because the procedure is pure the state of a computation at any time is determined by the contents of the data block and the active registers. Thus to interrupt computation for one user and continue computation for another is merely a matter of saving and restoring active registers and changing relabeling for the data block.

The only programming conventions which must be followed in writing the procedure body are those of avoiding self-modification. Avoiding direct self-modification is especially easy in a machine like the 930 which permits combinations of indexing and indirect addressing and which has an executive instruction. The programmer simply avoids storing information within the program. Assemblers
can check programs for such storage automatically.

In addition to the constraints mentioned above, the programmer must avoid the use of the normal SDS 930 subroutine transfer (BRM) since it stores the subroutine link at the head of the subroutine and thus within the procedure body. At the moment, a SYSPOP is provided to steer the link storage indirectly one level into the data block. This is done by placing the address of the subroutine link at the head of the subroutine, where the link itself would normally reside. Thus the SYSPOP SBRM y at location p first looks in y to find a link address z. The value p is then stored in z, which is outside of the pure procedure. Control is transferred to y + 1. Return is accomplished with the normal subroutine return instruction using indirect addressing (BRR~y). The indirect address sequence causes the machine to look first in y to find z and then in z to find p. Control goes to p + 1. It is anticipated that SBRM will be incorporated as a new 930 instruction.

In cases of the occurrence of POP's within a program, it is usually desirable to have the data block at least start within the user's block 0. The use of a POP places a return link in the user's location 0, and this, of course, must be in the data block.

It should be noted that the utility of relabeling in implementing common routines was not fully realized at the outset. It should also be observed that using relabeling for such purposes restricts the common routines virtually to subsystems (compilers, debuggers, interpreters, etc.) since an entire page is reserved for temporary storage. Routines of this magnitude, however, do have the most need to be single-copy.

In our system many functions which might otherwise force users to have copies of the same little routines in their programs are taken over by SYSPOP's. Finally, for those routines that lie halfway between (e.g., packages of mathematical subroutines—SIN, COS, etc.) the read-only facility allows users to share procedure storage. The only abridgment of the users' freedom here is that such routines must be located absolutely in user memory so that they may address themselves properly without asking the system to change relabeling upon entry and exit.

The Structure of SYSPOP's

The SYSPOP mechanism is basic to the overall system and is used extensively by programs at all levels. SYSPOP routines run in monitor modes. If called by a user the execution of a SYSPOP is part of his program, and this is the only instance of a user-controlled program running in monitor mode. The absence of normal protections during such intervals imposes constraints on the program structure of SYSPOP's as follows:

1. SYSPOP's must be written so as not to cause disaster if erroneously called. This feature calls for a certain amount of software interpretation, but it is on a different level from the interpretation spoken of in the section on Mode Transitions.

2. SYSPOP's are normally small and of short duration. Because they share the same link it is difficult to make SYSPOP's re-entrant without time-consuming maneuvers. The SYSPOP's are therefore not re-entrant and contain their own temporary storage.

3. Since SYSPOP's are not re-entrant and since they are shared by all users and by most parts of the system itself, program interruption is handled by allowing a SYSPOP in process to go to completion. This is done by having all SYSPOP's return control through a common (1-instruction) routine.

The reader should note the merit of the mode-changing scheme discussed in the section on Mode Transitions, as reflected in resulting simplicities in SYSPOP's. Recall that the mode of the calling program is stored in Bit 0 of the link, and that the SYSPOP accesses the calling parameters indirectly through the link. If a user calls a SYSPOP, relabeling will be applied to accesses of calling parameters, otherwise not. When returning control the SYSPOP executes a return instruction. If Bit 0 of the link is set, relabeling is applied and the mode is set back to users'. Thus modes are completely invisible to SYSPOP's.

SUMMARY

The project goals discussed at the beginning of this paper have been set. The time-sharing system involving memory relabeling, common routines, and duplex teletype operation has been in operation since April, 1965. The system is highly flexible and

From the collection of the Computer History Museum (www.computerhistory.org)
can provide, for users who require it, a response
time of less than one second.

It should be noted that memory relabeling* is
accomplished with no increase in access time. The
number of processor modes is small (two), and
mode transitions are done in such a way as to en­
able interrupt and user-called system routines to be
independent of mode.

The user machine is clean and well defined. In­
put/output is simpler, more foolproof, and device­
independent. The user is given a variety of other
services ranging from generalized file-handling
capability to string processing to assemblers, com­
cilers, debuggers, and editors.

ACKNOWLEDGMENTS

The authors would like to acknowledge the direc­
tion and valuable advice of Professors Harry D.
Huskey and David C. Evans, co-principal investi­
gators. Mr. W. J. Sanders made many valuable con­
tributions to the early system design and SDS 930
modifications. The time-sharing executive system⁹
and most of the subsystems were written by Mr.
Peter Deutsch and Mr. Butler Lampson, who suf­
fered through seasons of balky, fidgety hardware
and primitive input/output to produce an excellent
result.

REFERENCES

puter Utility Approach,” IEEE Spectrum, vol. 2,
no. 1, pp. 56-64 (Jan. 1965).

2. J. I. Schwartz, “A General Purpose Time­
Sharing System,” Proc. AFIPS Conf., 1964, vol. 25,
pp. 397-411.

3. W. W. Lichtenberger, M. W. Pirtle and W. J.
Sanders, “Modifications to the SDS 930 Com­
puter for the Implementation of Time-Sharing,” Doc­
ument no. 20.10.10, Project GENIE, University of
California, Berkeley (Jan. 1965).

inary Reference Manual,” Document no. 20.70.20,
Project GENIE, University of California, Berkeley
(Mar. 1965).

5. G. J. Cullen et al, “TWR Two-Station On­
Line Scientific Computer,” vols. II and IV, TWR
Space Technology Laboratory (July 1964).

cations,” Document no. 20.60.10, Project GENIE,
University of California, Berkeley (Jan. 1965).

Tablet: A Man-Machine Graphical Communica­
part I, pp. 325-331.

8. J. B. Dennis, “Program Structure in a Multi­
Access Computer,” Technical Report MAC-TR-11,
Project MAC, Massachusetts Institute of Tech­
ology (1964).

Time-sharing System Preliminary Reference Man­
ual,” Document no. 30.10.10, Project GENIE, Uni­
versity of California, Berkeley (Apr. 1965).

*The technique of relabeling was developed by M. Pirtle
in April 1964 and was implemented in two weeks on the 930
upon delivery the following November.