Technical and human engineering problems in connecting terminals to a time-sharing system

J. F. OSSANNA

Bell Telephone Laboratories, Inc.
Murray Hill, New Jersey

and

J. H. SALTZER

Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

Today, an increasing number of computer systems are used interactively by their user communities. Interactive use of computers, involving more prolonged man-machine contact than non-interactive use, requires a well human engineered user-system interface. The interactive user's performance—his rate of doing work and his ability and desire to utilize system capability—is a sensitive function of the success of this human engineering. In turn, the computer system's effectiveness depends on achieving a satisfactory level of user performance with reasonable efficiency.

This paper will be concerned with the human engineering of connecting typewriter-like terminals to general purpose time-sharing systems. Examples of such systems are Digital Equipment's 10/50 system for the PDP–10, IBM's Time-Sharing System for the 360/67, the Dartmouth Time-Sharing System, and the Multics system at MIT. Such systems are used by a wide range of users doing many kinds of work. Typewriter-like terminals constitute the majority of general-purpose remote terminals in use today; examples are the Model 37 teletypewriter[1] and the IBM Model 2741.[2] Although more complex terminals, such as those providing true graphical capability, are not specifically treated, many of the factors to be discussed apply to them. The special behavior and needs of specialized systems are not treated, but some of the ideas presented will apply in individual cases.

Value judgments about human engineering factors always involve a degree of individual taste which in turn depends in part on individual experience. Many of the ideas expressed here are the outgrowth of experience obtained during the growth and use of Project MAC's CTSS system[3,4] and during the development of Multics.[5]

Good user performance becomes possible when the user can easily and rapidly do what he wants to do. Consequently, many of the human engineering factors to be discussed relate to the user's ability to provide input as rapidly as desired, to control output, and to avoid unnecessary interaction.

First, we will discuss input/output strategies, since they broadly affect most of the other areas to be covered. Then we will discuss in turn, terminal features, the terminal control hardware, and the terminal control software—working from the user into the system. Finally, we will briefly mention character sets and character stream processing.

INPUT/OUTPUT STRATEGIES

The user's input consists of system commands, requests to programs, data, answers, etc. From the user's point of view, input can be divided into components according to whether or not it is expected that the component will cause output to occur. Some input is expected to cause output to occur—for example, a command to list a file directory. Other input may be expected to cause output only conditionally; for example, a command to rename a file may output an error comment only if the named file doesn't exist. Still other input may be expected to cause no output—for example, continuous text input into an editor.

From the system's point of view, the user's input can be considered a character stream containing certain characters indicating that action should be taken. In the common line-by-line input case, a return or new-line character is the only action character. In general, there may be a number of action characters. In certain applications treating all characters as action characters may be appropriate. The user ordinarily should know what action characters are currently in effect, since typing one of them initiates execution, which may in turn cause output.

The human engineering problem in collecting a user's input arises primarily because the user frequently knows much of what his input is to be well in advance. He may know the next several commands or the next several editing requests he wishes to input. In general, the components of this known-in-advance input can fall into all three output relationship classifications. Although the user often knows when to expect output, the system cannot.

The user should not be unnecessarily prevented from providing such input as fast as he can think of it and can type it. By collecting input asynchronously rather than synchronously with respect to the system's utilization of the input, the user and the computer can work asynchronously and in parallel rather than synchronously and serially.

There are four mechanisms that can individually or collectively facilitate providing input.

First, input can be collected whenever there is no output occurring. If the operation is full-duplex,* it is even possible to collect input while output is occurring. The typing of action characters should trigger program execution but not inhibit further input. Such asynchronous collection of input is usually referred to as read-ahead or type-ahead. A number of present day systems[4,5] provide a read-ahead strategy.

Read-ahead permits overlap of input with both system response time and program execution. Also, it permits programs such as text editors to gather text input continuously. Because erroneous input may be encountered, programs must be able to produce conditional output and also discard existing read-ahead to prevent compounding of errors.

A second mechanism is to allow more than one independent input component between action characters. For example, a system using new-line as an action character should permit more than one command on a line. Editors in such a system should permit more than one editor request per line. This outlook should pervade every level of programming.

Third, commands and other programs should be designed to avoid unnecessary interaction. One aid in doing this is to allow the typing of arguments to a command on the same line as the name of the command. For example, typing "edit zilch" is preferable to typing only "edit" and later answering the question, "Filename"? Default parameter values can frequently be assumed in the absence of typed arguments. Permitting both multiple commands and arguments enables various schemes for inputting factored command and argument sequences.[5]

Fourth, it is convenient if the user can create a file containing potential input and subsequently cause the system to take input from this file.

The use of these mechanisms can also improve system efficiency by reducing the number of separate program executions, since the program may find more input and be able to do more work during each execution.

The user should have reasonable control over his output. For example, whenever a stream of unwanted output occurs, it should be possible to stop it without undesirable side effects, such as losing too much of the results of immediately previous interactions. An interrupt mechanism, such as that detailed later, can be used to stop the output, cause execution to halt, and discard any read-ahead. If the system allows an interrupted program to catch the user's interrupt signal, a program desiring an extra degree of sophistication can be designed to recover from various conditions such as unintended execution loops or unwanted output due to unwise input. User control over output code conversion is desirable and will be discussed later. The ability for the user to direct program output to destination(s) other than his terminal is quite useful. For example, the output from a program which generates a large volume of output can usefully be directed to a file for later printing.

REMOTE TERMINAL CHARACTERISTICS

An excellent treatment of features desirable in typewriter-like terminals can be found in Reference 6. We will treat here certain important terminal design features which strongly affect the system designer's ability to human engineer the system-user interface.

A typewriter may be viewed as a collection of data sources—the keyboard, the receive-data lead of the modem or data set, and possibly a paper-tape reader—and data sinks—the printer, a control detector, the

* In full-duplex operation, transmission can occur independently in both directions. This requires independent keyboard and printer operation at the terminal, as well as independent input and output at the computer. The modems (or data sets) typically used to connect the kind of typewriter being discussed to the telephone line ordinarily provide full-duplex transmission.

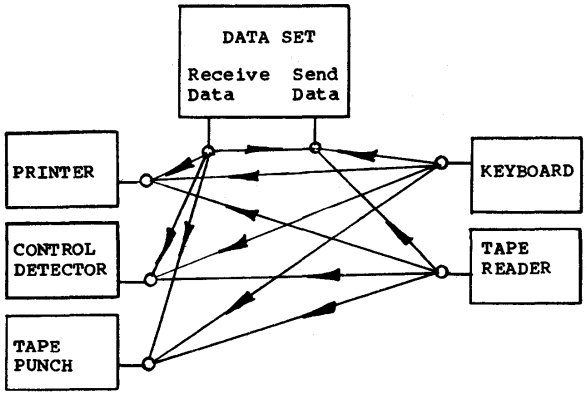

Figure (1a)—Typewriter data sources and sinks and possible interconnections

send-data lead of the data set, and possibly a paper-tape punch. Figure (1a) shows such a collection and possible interconnections. Flexible user and/or system control over these source-sink interconnections permits implementation of various input/output strategies.

As a specific example, Figure (1b) shows the interconnection control of a Model 37KSR teletypewriter. Control of the switches occurs by detection of control character sequences by the control detector associated with the printer. The interrupt detector and generator are discussed below. When the keyboard-to-printer connection is closed the terminal is in half-duplex mode and local printing of keyboarded data occurs. When this connection is open the terminal is in full-duplex mode, and the relationship between keyboarded data and printed copy is under control of the computer system. One common use of the full-duplex mode is to collect passwords without printing them. The full-duplex mode allows the printed characters to be simple mappings, or even arbitrarily elaborate functions, of the keyboarded characters. The ability to lock and unlock the keyboard allows the system to constrain the user to type only when input is able to be collected by the system.

The program interrupt ability previously mentioned can be achieved by full-duplex operation of both the terminal and computer, which permits an interrupt-implying character to be typed at any time. Another method, which does not require full-duplex operation, is the "line-break" technique,* where an always generatable unique signal can be transmitted. In addition, the ability of the terminal to respond to a break or interrupt signal from the computer regardless of its state provides a method of restoring the terminal to a desired state—typically ready to receive control or text information. As an example, the Model 37 responds to a break by locking the keyboard; the Model 37 break generator and detector are shown in Figure (1b).

The system should be able to maintain knowledge of and control over the states of the terminal. In particular, the system should be able to force the terminal into a state where the system can print on the terminal without user interference. As many terminal actions as possible—for example, those causing carriage and paper motion, color shift, source-sink interconnections—should be initiated by character sequences whether terminal or computer generated. This implies that the character set used should be sufficiently rich in control characters.

The terminal should not inherently hinder implementation of a read-ahead strategy. For example, the keyboard should not lock automatically after the typing of what the terminal assumes is an action character, such as at the end of a line; such terminal behavior is a violation of a general rule that the terminal shouldn't try to "outguess the software."[6] When a system controls input by keyboard locking the user should know when the keyboard is usable without having to test it. For example, the Model 37 lights a "proceed" lamp when the keyboard is unlocked. Using a "new-line" function (combined carriage-return and line-feed) is simpler for both man and machine than requiring both functions for starting a new line; the American National Standard X3.4–1968[7] permits the line-feed code to carry the new-line meaning. The terminal should have adequate functions for speeding up both input and output. Horizontal tabs are essential, form feed and vertical tabs are useful. They are the most useful when the user can easily set the stops himself

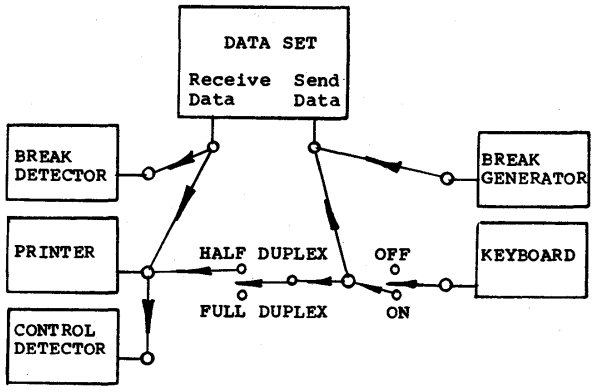

Figure (1b)—Model 37KSR teletypewriter interconnections

* The "line-break" or "break" signal usually consists of approximately 200 milliseconds of "space" ("0" bits). This is distinguishable from ordinary characters and is easily detected independently without the necessity of being able to receive characters.

using control character sequences; this is possible in some present day terminals.[1,8]

When a terminal has reached the system via a switched telephone network, the system may not *a priori* know anything about the calling terminal, and it can be useful if the terminal can send an identification sequence to the system upon demand. This sequence can be used to uniquely identify the terminal, to determine the terminal type, and to indicate terminal options. The Model 37 answer-back scheme is an example of a more than adequate identification. The economic advantage of having different terminal types statistically share computer ports is a strong motivation for the system to be able to experimentally determine the terminal type. It is necessary only that each terminal to be supported be able to respond to a transmission from the system and that either the transmission or the response be unique. Multics currently supports four types of terminals and determines which type by performing an experiment involving identification responses.

The Model 37 teletypewriter and the General Electric TermiNet–300[8] (Registered Trade Mark of the General Electric Company) provide nearly all of the above-mentioned features. Consider the standard version of IBM's Model 2741[2] terminal, which is widely used as a time-sharing terminal. This terminal can only be used in the half-duplex mode, so there is no way to inhibit direct local copy or to exploit full-duplex operation. The terminal cannot be interrupted by the system while the keyboard is unlocked; thus the system can't force the terminal to accept output while the user is able to type. This property makes read-ahead a somewhat dangerous strategy, since conditional output is impossible while the user is able to type. The keyboard locks as a result of typing "return" (new-line), and requires the system to respond and unlock the keyboard before the user can proceed. Even with instant system response, the delay before typing can continue (caused by the transmission of control characters) is noticeable, so that any read-ahead strategy is degraded. No keyboard-unlocked indication is provided for the user. Adding an identification mechanism, enabling interrupt to be always generatable and receivable, adding a local-copy suppress mode, and eliminating the automatic keyboard lock, are possible modifications; unfortunately, as is characteristic of post-initial design changes, they add significant cost.

COMPUTER SYSTEM TERMINAL CONTROL HARDWARE

The terminal control hardware used today broadly falls into two categories. One is the peripheral stored-program controller; the other is the hard-wired controller operated directly by the main computer. The major difference between these in practice is in the way the control software is modularized. The various functions to be performed by the terminal control hardware and software together can be divided between them almost arbitrarily. The decisions made when allocating logic between a main machine control program and a hard-wired or stored-program controller involve a variety of economic and other management considerations; it is not our intention here to discuss relative virtues of hard-wired and stored-program controllers. In either case, if the controller provides a primitive but complete set of functions, the terminal control program in the main computer can assume primary logistic control over the terminals. Such a controller is assumed in the following discussion, which describes suitable controller functions.

Because it may be safely assumed that new and better terminals will continue to be introduced, the terminal controller should be flexible enough to permit operating these new terminals with minimum modification. Specifically, parameters such as the number of bits per character, the character parity, and the bit rate should be program controllable or at least field modifiable. At any given time, there are usually several terminal types worth supporting. The controller must be able to handle the corresponding variety of line control

```
READ START SEQUENCE
1. SET TRANSMIT MODE.
2. TRANSMIT LITERAL "EOT" CHARACTER.
3. SET READ MODE.
4. READ ONE CHARACTER (SWALLOW "EOA" CHARACTER).
5. SET ACTION CHARACTER LIST TO (JUST) NEW-LINE.
6. TRANSFER TO READ SEQUENCE.

READ SEQUENCE
7. READ INTO BUFFER 1.
8. READ INTO BUFFER 2.
9. TRANSFER TO KEYBOARD-LOCKING SEQUENCE.

KEYBOARD-LOCKING SEQUENCE
10. SET TRANSMIT MODE.
11. TRANSMIT LITERAL "BREAK" SIGNAL.
12. STOP.
```

Figure 2—Command list to read the keyboard of an IBM 2741

requirements without undue programming effort and without undue main processor intervention; this implies suitable controller command chaining, which is described later.

When terminals reach the system via a switched telephone network, the system needs to be fully aware of call-ins, disconnects, and line failures. Thus the controller should make available to the software all status available from the modem or data set, and allow the system to enable interrupts for status changes. Similarly, the controller should allow the system to set all the control leads of the data set, so the system can control data set answering, make lines in hunt groups appear busy, and initiate disconnects. Such control allows the system to disable improperly working lines and to exercise system load control.

Certain terminal functions (tabs, form-feed, new-line, etc.), require that a delay sufficient for completion follow its initiation. If this delay is provided by the inclusion of "fill" characters (causing no terminal action), only the needed number should be transmitted. Experience suggests that accurate delay calculation, providing only the actual delay necessary, speeds up output and gives the system a smoother and speedier image.* Preferably, delays should be calculated to the nearest bit time rather than to the nearest character time.

An important controller feature is the ability to act on a list of queued "commands" from the control software. The command repertoire should include commands to set controller and data set modes, obtain controller and data set status, transmit from a buffer, read into a buffer, transmit a literal bit string, and transfer to another command. The tandem execution of two or more read *or* write commands is usually called "data chaining." The tandem execution of a list of mixed read and write commands is usually called "command chaining." A transfer command allows the list to be conveniently built of sublists and dynamically threaded together. The ability to transmit literal bit strings allows the transmission of delays (all 1s), breaks (all 0s), and canned control character sequences.

The ability to data chain while reading is an important help in allowing continuous input, because it allows a more relaxed software response to an exhausted buffer. To simplify buffer management, the controller should be able to interrupt on an action character but continue reading sequentially into the same buffer; an interrupt should also occur on data-chaining to alert the software of an exhausted buffer. It is useful if the action character(s) detected can be dynamically set by the software. If the action character(s) can be associated with each individual read command and the action to be taken individually specified, the ability to chain a list of mixed read and write commands permits handling a variety of terminal types and the design of good read-ahead strategies. The detection of a received "break" signal should halt the controller and cause an interrupt.

Figure 2 shows a hypothetical command list similar to lists implemented in Multics. The list illustrates reading the keyboard of an IBM 2741 (modified to accept break signals), and employs several sublists. After an interrupt from the controller indicating the exhaustion of buffer one, the control software would ordinarily replace the transfer in step 9 with a transfer to another read sequence. The keyboard-locking sequence stops input should the system fail to obtain another buffer prior to exhaustion of buffer two.

General Electric's General Input/Output Controller (GIOC) used with the GE 645 system (on which Multics is implemented) is an example of a communication controller that provides most of the above-mentioned controller functions. Reference 9 describes the design of the GIOC.

TERMINAL CONTROL SOFTWARE

The following discussion will be concerned with terminal control software in a main computer using a flexible terminal controller. We will discuss the need for flexibility of design and operation, the implementation of input/output strategies, some of the responsibilities to other system software, and a little about the interface to user programs.

The major areas where flexibility is important in terminal control software are the ability to operate various terminal types, and the ability to adapt to the variable behavior and needs of users.

The advantages of being able to operate a variety of terminals are: (1) freedom from dependence on one terminal supplier; (2) ability to take advantage of newer terminals; (3) user access to terminal features not all found on one terminal; and (4) satisfaction of individual user needs and preferences. The ability to operate various terminals and to easily extend operation to new terminals requires a flexible and convenient method for converting between internal system character codes and physical device codes, and for handling the different kinds of terminal control.

If the terminal control software is designed to be driven by a collection of tables, it should be possible to

* This effect was noticed during the early development and use of Project MAC's CTSS. Subsequently on both CTSS and Multics, users quickly noticed longer-than-needed delays on new terminals or due to untuned new software.

embed device differences and perhaps user options in the tables rather than in the harder-to-change program. Flexibility and extensibility can be achieved by sufficient ingenuity in choosing what information is to be relegated to tables. The generality required in such tables depends on the range of terminals to be controlled. Control driving tables can include the following:

1. Input and output code conversion tables.
2. Device parameter tables.
3. Tables of controller command sequences for identifying and operating the various devices.

The system-device code mappings contained in the code conversion tables would include suitable "escape" character sequences for handling system-defined characters not present on some terminals.* Also, additional tables could be provided for alternative conversion modes on the same terminal,** and to accommodate, for example, the user who wants to use a non-standard print element on an IBM Model 2741 or an extended-character type-box on a Model 37 teletypewriter.

The device parameter table would contain such information as default action characters, default output line overflow length, default code conversion table name, carriage return speed for delay calculations, character parity, etc.

The operating command sequence information includes sequences for initiating a write, writing, terminating a write, initiating a read, etc. The identification command sequences are the ones used for terminal type determination; often the terminal identification code is obtained as a by-product of type determination.

If the hardware controller can interrupt on an action character and otherwise continue, then only a small fixed buffer space need be associated with each active terminal—that needed for current physical input/output by the controller. All other buffer space can be pooled and assigned to individual terminals on demand. A simple read-ahead strategy can be implemented by copying input characters from physical collection buffers at action character interrupt time into a linked list of input buffers obtained dynamically from the buffer pool. When the user program requests input, the input is taken from the user's input buffer list. Similar buffer schemes have been long used for handling devices such as magnetic tapes, but are not often seen used for terminal control.

Similarly, a user program's output can be copied into a dynamically grown buffer list. Physical output occurs by refilling from the output list the physical buffer associated with each terminal every time its contents have been output. With half-duplex operation, emptying the output list should reinstate read-ahead. Letting a user program emit substantial output before suspending its execution (referred to as permitting write-behind) usually improves system efficiency by reducing the number of separate program executions. Physical output should be initiated as soon as there is any, and not delayed perhaps waiting for a buffer to fill. Aside from distorting the sense of program progress, such output delay can make program debugging very difficult. For example, debugging often involves inserting additional output statements in various program branches to obtain flow information. It is misleading not to see this flow information prior to a program entering an unintended loop, because of inappropriate output delay.

Of course, reasonable limits must be put on how much read-ahead and write-behind is permitted, lest a single user or his program seize all available buffers. Adequate total buffer space should exist to cover reasonable fluctuations in total demand. Algorithms to limit the buffer space that can be claimed by one user should be generous when conditions permit to avoid losing the advantages of read-ahead and write-behind. During peaks in total demand that tax the available space, these algorithms should be gracefully restrictive. Some successful limiting algorithms[4] involve allowing each user to accumulate a fixed fraction of either the total buffer space set aside for all such terminals, or of the current remaining space. Because the average output character rate is typically ten times the average input character rate,[11] the limiting algorithms must prevent write-behind demands from completely depleting the available buffer space, so that some space is kept available for collecting input.

The terminal control software is responsible for blocking further execution of the user's program when it requests input and none is available, and whenever it exceeds the write-behind limit. In the waiting-for-input case, the program must be restarted when an action character is detected. In the waiting-for-output case, the program should be restarted when the back-logged output has dropped to an amount whose physical output time will approximately correspond to the restart delay (system response), so that the physical output can occur continuously.

Another responsibility of the control software is to detect and report disconnects and the user's interrupt (break) signals. Disconnects should be reported to the

* For example, the sequence "¢<" could be used to represent a "[" on an IBM 2741.[10]

** If the default mode utilizes escape sequences for missing characters, an alternative mode could print blanks for such characters to permit inking them in.

system module responsible for reclaiming the communication line and making it available to other users. The interrupt should be reported to the system module responsible for suspending the execution of the user's program, pending input from the user indicating the reason for the interrupt.

The subject of the interface between the terminal control software and a user program is too large to be covered thoroughly in this paper. The flexibility built into the control software should be available to the user program. It should be possible, for example, to request a different code conversion table, specify a new line-overflow length, discard existing read-ahead input, turn off and on the terminal's local copy, disconnect the terminal (if it is on a phone line), request the terminal's identification code, etc. A particularly bad interface example occurs in some systems in use today, in which it is not possible to simply read from the terminal. The user program can only issue a write-read sequence. Output is forced to occur between each line of input. Consequently, the user program is scheduled and executed to perform this obligatory output. The overall effect is to degrade system efficiency as well as seriously slow down the user at the terminal.

The typewriter control software in the Multics system is almost completely driven by tables organized along the lines described above. A single control program currently operates the Model 37 teletypewriter, IBM Models 1050 and 2741, and the General Electric TermiNet–300. Full read-ahead and write-behind are implemented with a maximum limit which corresponds to about 700 characters for both the read and write buffer lists. A buffer pool of 250 14-character block has proven adequate in a 35 user system. In addition each active typewriter has physical read and write buffers of about 100 characters each. After a program exceeds the write-behind limit and is blocked from execution, it is restarted when the write-behind has dropped to about 60 characters.

CHARACTER SET AND CHARACTER STREAM CONSIDERATIONS

The choice of a suitable character set and suitable processing of the input and output character streams are extremely important human engineering issues which can affect the user's view of the system as much as any of the factors already discussed. An earlier paper[10] contains a detailed treatment of these issues; it includes discussion of character set choice, input and output code conversion, input text canonicalization, and input line editing.

CONCLUSIONS

The total effectiveness of a time-sharing system and its user community depends a great deal on the human engineering of the system-user interface seen by the user from the vantage point of his terminal. We have concentrated on the factors affecting the user's ability to provide input at the rate he wishes and to control output. Suitable input/output strategies can allow the user to work in parallel with the computer. We have maintained that a coordinated design of the terminal, the terminal control hardware, the terminal control software, the system's command stream interpreter, the commands, and other programs, are all necessary to achieve the desired goal.

Many of the individual factors discussed, of course, have been recognized as important in the design of various systems. It is rare, however, to find a sufficient set of these factors implemented to a satisfactory extent. One reason for this is that the system designer is often faced with using previously designed terminals and terminal control hardware, and even previously written software. Another reason is that even with experience using a variety of interactive systems it can be difficult to assess the sensitivity of the human interface to differences in design. Too often, this lack of complete design control together with insufficient experience results in a system design lacking some important features.

ACKNOWLEDGMENTS

Many of the techniques described here were developed over a several year time span by the builders of the 7094 Compatible Time-Sharing System at MIT Project MAC, and by the implementors of Multics, a cooperative research effort by the General Electric Company, the Bell Telephone Laboratories, Inc., and the Massachusetts Institute of Technology. Among those contributing to the understanding of how to effectively implement typewriter input/output were F. J. Corbató, R. C. Daley, S. D. Dunten, E. L. Glaser, R. G. Mills, D. M. Ritchie, and K. L. Thompson.

Work reported here was supported in part by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01). Reproduction is permitted for any purpose of the United States Government.

REFERENCES

1 *Model 37 teletypewriter stations for DATA-PHONE* (Registered Trade Mark of the Bell System) *service*

Bell System Data Communications Technical Reference American Telephone and Telegraph Company September 1968

2 *IBM 2741 communications terminal*
IBM Form A24-3415-2 IBM Corporation New York

3 P A CRISMAN
The compatible time-sharing system: A programmers' guide
Second Edition MIT Computation Center The MIT Press Cambridge Massachusetts 1965

4 J H SALTZER
CTSS technical notes
MAC Technical Report No 16 Project MAC Massachusetts Institute of Technology March 15 1965

5 *The multiplexed information and computing service: Programmers' manual*
MIT Project MAC Cambridge Mass 1970 (to be published)

6 T A DOLOTTA
Functional specifications for typewriter-like time sharing terminals
Computing Surveys Vol 2 No 1 March 1970 pp 5-31

7 *American National Standard X3.4-1968*
American National Standards Institute Oct 1968

8 *Programmers' manual for the General Electric TermiNet-300 printer*
No. GEK-15002 General Electric Company 1969

9 J F OSSANNA L E MIKUS S D DUNTEN
Communications and input/output switching in a multiplex computing system
AFIPS Conference Proceedings Vol 27 Part 1 1965 (1965 Fall Joint Computer Conference) Spartan Books Washington D C pp 231-241

10 J H SALTZER J F OSSANNA
Remote terminal character stream processing in multics
AFIPS Conference Proceedings Vol 36 1970 (1970 Spring Joint Computer Conference) AFIPS Press Montvale New Jersey pp 621-627

11 P E JACKSON C D STUBBS
A study of interactive computer communication
AFIPS Conference Proceedings Vol 34 1969 1969 Spring Joint Computer Conference AFIPS Press Montvale New Jersey pp 491-504