# HUMAN FACTORS INFLUENCING THE COMMAND, CONTROL AND COMMUNICATION FUNCTIONS OF A TIMESHARING SYSTEM

by

Jon A. Stewart
Department of Computer Science and Statistics
University of Southern Mississippi

## ABSTRACT

The intent of this paper is to explore various human factors which influence the design of a timesharing system and its processors; to examine characteristics of existing systems; and, to propose solutions which more nearly suit the requirements of the timesharing user. Most illustrations will be drawn from extensive personal use and observations of users of two particular systems -- the XEROX CP-V system on a Sigma 9 and TOPS10 on the DECSystem-10. Both these systems are characterized by extremely flexible command capabilities, almost if not complete compatibility with their batch facilities and very crisp response (for both the CPU resource and file system functions) when under proper administrative controls.

## INTRODUCTION

The timesharing neophyte of a few years back was making a transition from batch and had a specific set of learning problems. Today, especially in educational institutions, the new user will probably have his first contact with a computer through remote terminal access and may then at a later date become familiarized with other aspects of the system. Most concern here will be with that initial orientation phase in which the user is first learning the characteristics of the terminal, how to use various processors and is having his first "experience" with the "personality" of the system through the messages communicated and the response of the system to various commands.

My own first experience with a timesharing system dates back to late 1968 with a PDP-10 in DEC's plant in Maynard, Massachusetts. We were involved in an evaluation process which eventually led to selection of the PDP-10 for an academic-only computer operation at the University of New Orleans (then LSUNO). During the same period use was made of RACS (on a 360/40 in the New Orleans data center) and BTM (on an SDS Sigma 7 in El Segundo). Contacts with all these systems led to increased confirmation that timesharing was the way to do business in an academic computing environment but there were many peculiarities of these relatively undeveloped systems. Initially, all these systems produced an impression of instability -- terminals misbehaved, files disappeared, etc. Causes for this "mysterious" behavior were later determined to be: 1) misuse of the system (incorrect commands); 2) line noise and accidental key depression which caused the terminal to change state; and 3) actual bugs in the systems. Today there has been a vast improvement in stability and the "feel of security"; but, observation of users at USM in their contacts with CP-V and the Sigma 9 still produces many a surprise!

Most frequently, user problems result from an unexpected response which may be explained by one of several possibilities: 1) changes of terminal state (various command and input/output modes exist) deliberately induced by a processor or accidentally by the user; 2) a delay due to CPU or file system response which cannot, in most cases, be anticipated by the user who usually has little knowledge of the computational resources required to perform his task, or the actual current demand on these resources; 3) an error message which, though it may be descriptive and not just an error code, does not usually indicate the nature of the corrective action

which should be taken (unless the user has had considerable experience with the particular system component); or, 4) an incorrect I/O assignment, missing file or incorrect file type (or device type) for a particular process.

## TERMINAL INTERFACE DESIGN

Problems caused by unexpected terminal state changes can be alleviated or almost defeated by making the "combination" required a difficult one to produce accidentally. With limited available key combinations, and in keeping with the objective of minimizing key strokes and allowing sufficient flexibility, this goal seems to be rather elusive. But another solution would be to have a simple "panic" procedure, uniform across all processors, which would initialize the terminal status.

The question of prompt characters and the transparency of command levels is another important concern in terminal interface design. Timesharing utilities attempting to provide command uniformity and ease of use tend toward the one language/one level concept -- often some extension of the Dartmouth BASIC type of command facility. More sophisticated systems allow the user to re-design his command interface -- substituting synonyms or even changing syntax. Vendors in their standard offerings seem to take the middle ground with recognition prompts keyed to the processor and essentially a two level command structure -- namely, the operating system (SYSTEM) level or the processor (subsystem) level (e.g., BASIC). Another possible solution would be to have an assistance function which could easily establish the processor currently doing the command interpretation. Distinguishing prompts, which CP-V uses, are really not that necessary since: 1) the advanced user knows what processor he has invoked, 2) the inexperienced user will not know what the different prompts mean anyway, and 3) an assistance feature, if made available, could establish the identity of the processor in any eventuality. Having different command levels and terminal states controlled by user and/or processor seems desirable for flexible and efficient use of a system; however, the user should be able at any time to re-establish a basis for understanding the responses of the system and continuing his command/response dialogue with the system.

Minimization of key strokes is certainly a desirable objective since touch typists with good speed are the exception, rather than the rule, in most general user communities. Commands should be simple in form and, ideally, uniform in syntax across all processors. Abbreviations should be allowed and an easy rule for forming an acceptable command abbreviation is desirable (such as, first three letters, initial letters of multi-word commands). Keying input is, at best, a poor solution. Until practical voice or optical character recognition replaces the ASCII terminal it seems that we have all agreed to be stuck with the functional characteristics of this device; but we don't have to be committed to its graphics set. Perhaps a good interim solution here would be the establishment of standard commands (RUN, SAVE, LOAD, CAT, whatever...) implemented through software interpretation of available CONTROL/SHIFT key combinations. With the full duplex protocol handled by many systems the echoed response could visually confirm the command typed -- such as EDIT, RUN, SAVE, etc. -- but the typing time for inputting the commands would be substantially reduced. Better yet, of course, would be to re-design the terminal to include a "function" pad which would allow single key input of the necessary system commands.

A final word about terminal design: the question of full or half duplex, line at a time or character at a time transmissions (with echoplex protocol) seems resolved for all but a few vendors (they may never come around!). Observation of many casual or occasional users would certainly indicate that full duplex is not their mode of operation, whether or not the system supports it. Interrupt overhead for the single character at a time systems can be significant for the higher speed displays; but, at the same time, can be easily absorbed in frontend mini- or micro-processors. The user might even be allowed to select a preferred mode of terminal operation -- even getting a rate cut for selecting the lower overhead method of operation. But there is another side to the full duplex story. The most sophisticated and, indeed, most effective and prolific timesharing users make good use of the type-ahead capability afforded by the full duplex protocol. These users seem completely frustrated when faced with the "primitive" systems which: 1) lock keyboards after transmissions (return or EOT); 2) refuse to accept type-ahead; or, 3) simply can't respond fast enough because of software interlocks and consequent processes which have "inertia" -- i.e., must complete before additional command interpretation or action can proceed, or

even recognition of "escape" or "break" requests.

Editing capabilities are also closely tied to terminal support and must be given special attention because successful use of the editor -- which encompasses not only file building and editing but examination of output and many other functions -- is often the key to successful use of the timesharing system. By placing a terminal in non-echo mode the editor can accept interspersed command and data information. For example, the SOS editor (from the Stanford AI Lab) has very effective within line editing capability. In the "alter" mode this editor uses spaces and rubouts as cursor positioning commands in order to move to positions within a line where deletions, insertions or changes are to be made; at that point particular single letter commands indicate switching to a data input function and a special character is used to terminate data input and return to command inputs. Of course, editing functions do not have to be supplied by the central (main) computer system; micro- or mini-processors in either communication frontends or in the terminals may actually perform editing without any regard for the nature of the final transmission to the central system where, presumably, the file system exists in which the information from the editor is ultimately deposited.

## UNEXPECTED DELAYS IN SYSTEM RESPONSE

This is perhaps the most mysterious aspect of a timesharing system, not just for the new user, but for the long time, relatively sophisticated applications programmer. For one thing, most applications programmers have insufficient knowledge of either hardware or software performance characteristics of the system they are using; and, in fact, are not generally allowed any (or much) information about the current demand (load) placed on the system. This latter information gap sometimes (usually?) extends even to those demands placed by the user himself. It would not place an inordinate demand on any good system (that is, one with sufficient built-in performance monitoring capability) to allow the user to display at any time: 1) an indication of the total demand on the CPU resource and his job's predicted (through analysis of past history) current demand on that same resource; 2) an indication of the total demand on the disk system (sectors read and written and seeks per some time interval) caused by both file transfers and page/swap traffic; and, 3) display of all pertinent cumulative and last snap (small interval observation) statistical parameters characterizing his session (connect time, CPU time, disk I/O, terminal interactions, average line lengths, etc.).

Information such as the above mentioned should be available on demand at any time without interfering with the current process for the user. There should be a simple summary from selected data geared to the requirements of different user categories. This information would assist the user in: 1) deciding whether to continue, suspend (saving for future re-activating) or abort the current task; 2) predicting resource demands for estimating time and cost requirements of the job; and, 3) deciding whether the system (if not the user task) is performing as it should. This would also contribute to: 1) educating the user generally about the system; 2) more efficient usage of the system (fewer invalid runs due to "early" error detecting and unnecessary reruns caused by premature aborts); and, 3) better program design by supplying the more advanced user a powerful tool for analyzing and improving program performance through detection of bad design characteristics.

## CONTROLS FOR PREVENTING ACCIDENTAL AND/OR MALICIOUS MISUSE

As various timesharing systems evolved, more sophisticated user (usage) controls were added; many at the request of users who had experienced difficulties. It is not the intent here to examine the more usual account privileges and limits but to address those controls (usually lacking) which could be used to prevent accidental and malicious misuse by alerting both the user and the system administrator to program "behavior" which is outside of expected or acceptable limits. For example, especially in student environments, processes often fall into accidental loops and needlessly waste valuable computing time. CPU "consumption" per elapsed time (connect time) interval could be monitored by the system to suspend activity on such a "suspect" job. Different CPU consumption limits could be established by account, since some users have a much higher expected consumption rate than others. Actual decision to abort could still remain with the user but the system might simply refuse to continue the suspended task until the average rate of CPU usage for the session (say CPU minutes per connect time minute) had dropped below the acceptable account limit. If the user insisted in continuing the CPU-bound task the limit would soon be exceeded again and the task again

suspended. Some users would wish to set temporary usage limits in order to be able to detect abnormal program behavior earlier than otherwise possible. These same ideas could be usefully extended to other resources -- disk space, main storage job requirements, etc. CPU, disk and main storage controls of this type would supply the system administrator with very effective load control based on job (account) profiles. The often asked question "How many users can the system support?" might then be given a reasonably accurate answer -- and clearly the answer would vary with the setting of the controls.

## MAN/MACHINE COMMUNICATION GAPS AND POSSIBLE SOLUTIONS

Commands given by the user to the system often have been chosen to mimic English in syntactic form and meaning (COPY A OVER B or RUN X, etc.). Sometimes they convey unintended meaning which traps the user into invalid use of the command. For example, RUN X should work for program x, no matter what its form -- but it won't on CP-V or many other systems. X which is really a file might be a Fortran source, a BASIC source, a relocatable object module, a core image save file, a load module, a compressed file or many other possibilities. Thus, it is perhaps expecting too much that this command work for all forms of X but an error message to the user, in this case, could clearly specify why the command didn't work -- "Program does not exist.", "BASIC sources only allowed under BASIC.", or "Linking X, load module does not exist." might be some of the possible responses to different situations. To perform such services successfully the system has to know the distinguishing attributes of each file and, ideally, be able to associate sets of related files (source, object, load module for example). Inevitably, ambiguities will arise -- for example, both a Fortran and BASIC source might exist and a load module produced from the Fortran source -- all by the same name X. What then does the user intend when typing RUN X? The protective system would not execute the load module without comment but would alert the user to the existence of the two different sources.

Messages given by the system to the user are open to many forms of misinterpretation. These generally are caused by: 1) misleading choice of message by the designer -- often accepting a default when more specific analysis should be performed; 2) poor error (trap) control often originating in the runtime support systems for various processors and resulting in loss of information which might otherwise point to a particular problem situation; 3) lack of understanding of terminology, unawareness of existing documentation which could explain a message, or inconsistent use of terminology by the designer; and, 4) shortcuts taken in the name of system efficiency (cutting overhead). Most of these can be corrected by good design of the message system in the first place, which then allows continual improvement as users have trouble with the system and feedback information about various problem areas. Lack of uniformity and preciseness in terminology is a serious industry-wide problem which can't be corrected by a particular vendor or educational institution; but a given installation can do much toward stating the "official" meaning of messages issued by a specific system and change the messages as required to be more useful given the level of understanding (and error propensities) of the user community.

A timesharing system should be responsive in many ways, not just in achieving the efficient execution of properly submitted user tasks. It must inform the users of their problems and, inevitably, of the system's problems. It must encourage the good tendencies of the users and discourage or prevent the bad tendencies. To do these things involves careful study of the man/machine interface and other factors which influence effective on-line computer usage. Many good systems do exist but neither of the two which were the basis for much of this discussion come close to all the objectives stated here; however, they both have potential for relatively easy change and did improve immeasurably over the years.