To: Distribution

From: S. H. Webber

Date: April 14, 1980

Subject: Menu Management Software

Purpose

This MTB describes the current state of what has been called the menu management software. It also proposes several additions that should be made so that a complete and consistent product can be produced. A final section proposes further enhancements that I believe are reasonable, but need not be done before the system is made available as a product. The assumption is made that the current software will indeed be turned into an official product in the near future.

Menus in General

For years menus have been used by many computer systems as a means for presenting the various options available to a user of the system at a given time. There are many formats that menus can take but most have been designed with a video terminal in mind.

Basically, a menu is a visual presentation of a set of options available to the user. This set of options is a small finite number at any time and hence all options can be, and are, named on the screen or paper for the user. The basic sequence is: display the set of options; the user selects one; and the associated action is performed.

This technique has as its main strength its simplicity. The users can easily figure out (or be trained) how to use a menu system. From that point on, the user is limited only by specific preset options provided to him in the application he is using.

Many menu systems have dynamic menus, i.e., menus which are generated "on-the-fly" as a function of the data available at the instant the menu is displayed. Emacs presents many such menus in the mail, directory editor, and other subsystems supported. The current menu manager supports such dynamic menus in only a primitive way. Extensions to this are proposed in "What is "Needed" below.

Even without fully dynamic menus, a very useful and effective interface can be presented. I would propose we try to take what we have now and turn it into a product and defer most enhancements to a second or subsequent release.

Advantages of Menus

As mentioned above, one advantage of a menu interface is its simplicity. Other features include:

- The user need not remember anything. This includes command names, formats, control argument names, or system specific constructs.

- The user need not type very much. This is made possible by creating more output in the display of the menu--which for slow line speeds can be time-consuming. Note that the trend is for line speeds to increase. User typing speed is not increasing.

- Menus are less error prone. Typing in erroneous requests nor data is much less likely to cause trouble.

- Menus can be used as a convenient packaging means for tying together many functions. The abbrev processor combined with exec_com are other examples of such packaging as any sophisticated Multics user can attest. Menus provide such "macro" facilities to the totally naive user.

- Menus can be (and in the current implementation are) associated with an effective "help" mechanism to assist all users.

- Menus are easy to learn how to use and easy to use once it is learned how to use them.

- With menus, a consistent interface can be presented to a very large class of application programs. No longer need the user know how to quit out of a given subsystem--all subsystems can be presented in the same way.

- The same user interface can be presented on many different operating systems. This way turn out to be very important in the future as we opt toward more distributed systems.

Disadvantages of Menus

There are a few problems associated with using menus as the interface for application software. Two such problems are:

- Menus take time to display. On terminals with low line speeds, much time can be wasted and the user can become frustrated as he waits for the menu to display. This is a problem only with slow line speeds. Solutions range from faster line speeds to moving much of the menu management "closer" to the terminal (so that line speed is not an issue).

- By their very nature, menus provide only those options the application programmer (or designer) thought appropriate. A knowledgeable user may become frustrated knowing there is more he can do, but not being able to "get at" the features not included.

Features of the Current Menu Manager

The programs developed so far have the following capabilities:

- It is easy and straight forward to create menus.

- It is easy to use nearly all existing Multics user-ring software through a menu interface.

- It is fairly time-consuming to write the input checking software needed to check for the validity, consistency, and format of non-menu-generated input (what is called "bottom window" input, below).

- A very large set of menus can be linked together into a hierarchial network making "parent-offspring" type navigation very easy.

- A help facility is integrally connected with all menu options.

- Screen management facilities are provided.

- The menu software depends on break-all mode and other terminal management facilities developed for emacs.

Terminology

The following terms (underlined as they are defined) were derived during the development of the menu manager and, hence, are based on video terminal type menus. A recent printing terminal capability has been added to the menu manager and the reader is expected to make the appropriate mappings for printing terminal use.

The screen is the standard user programmable, visible area on the CRT of a video terminal. Most screens have 24 lines, each 80 columns wide. The menu manager divides the screen into two regions, the menu region and the bottom window. The menu region consists of 0 or more header lines, followed by 1 or more lines of options, followed by 0 or more trailer lines. The lines including the options are collectively called the menu.

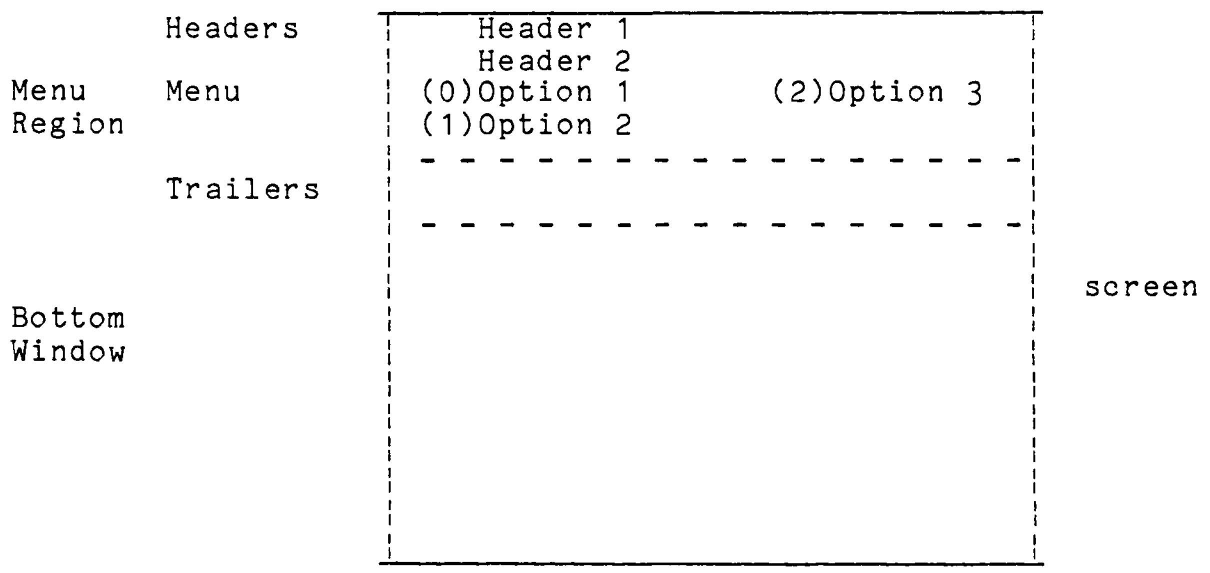

Figure 1. Terminology Used

Menu Input Conventions

Most video terminals have keys called function keys. On some terminals these have fancy names such as "HOME" or "INSERT", "TRANSMIT", or "GOTO". But on most terminals they are named simply "1", "2", etc. or "F1", "F2" ... (some terminals even use color coding to "name" function keys). Basically, function keys are keys usually not found on a typewriter. [It should be pointed out that some terminals have function keys that only have a local effect, i.e.,, they do not cause any characters to the sent over the communications line when pressed. Such keys are fairly useless and all following discussions will assume fully functional function keys.]

The menu manager uses function keys to provide the user with a few simple, universal capabilities. These include:

1. Print help information about an option on the subsystem.

2. Display the first (top level) menu.

3. Display the parent menu to the current menu.

4. Quit from the menu manager.

5. Escape to full Multics command level.

6. Redisplay the screen (to clean it up).

On the VIP7801 terminal, these functions are provided by function key F1 thru F5 and the CLEAR RESET function key respectively.

In addition to these general purpose options, the user can also select an option and have the associated action performed. This is done simply by typing the number or letter next to the option.

The cursor associated with a video terminal is used to indicate to the user what "state" he is in. If the cursor is next to one of the options in the menu, the user is "in the menu". Otherwise, the cursor and the user are in "in the bottom window". When the cursor is in the menu, the only thing he can do is to select on of the 6 generic functions (tied to the function keys) or select one of the options. When the cursor is in the bottom window, either information is being displayed to the user (help info or output resulting from selecting an option) or the user has been asked to type in some specific text or data as a result of selecting an option.

Bottom Window I/O Conventions

When in the bottom window, the user is presented with a slightly non-standard Multics terminal interface. In particular, end of screen processing and primitive input line editing features are provided. [Both of these were added because all I/O in the bottom window had to be treated specially anyway in order to prevent the menu from being "scrolled off" the screen.]

The specific editing features are:

- kill and erase are done dynamically
- word kill is provided
- upper-case word, lower-case word and capitalize word features are provided
- interchange the last 2 characters
- redisplay the current line
- clear the bottom window and display the current line being input at the top pf the bottom windows, and
- retrieve the last word or line that was killed (including the entire previous input line if no other "killing" was done)

The end of screen processing (analogous to what you can do when you get "EOP" printed at the bottom of a video terminal) consists of the following options:

- Continue printing by "scrolling" in the bottom window (this option requires the "delete lines" capability in the terminal being used)
- Clear the bottom window and continue printing at the top of the bottom window, and
- Abort all output until the next read request (either from the menu or from some program executing as the result of selecting an option). [Aborting <u>all</u> output up to the next read is <u>not</u> a good solution in that event call output and indeed simple ready messages will lost.]

Terminals Supported

The menu manager software was designed to work primarily with video terminals. A printing terminal capability has been provided and a "glass tty" capability is being planned. The printing terminal version expects the backspace function to be provided. For the video terminals, however, much more information about the terminal is needed. This additional information is maintained in a new version TTF (terminal type file). A new interface to the TTT is provided which the menu manager uses to figure out how to manage the particular terminal being used. The new TTF structure and the programs to manage it (cv_ttf, display_ttt, etc.) will all be described in another

MTB. The principal program of interest to the menu manager is video_control which returns the terminal specific escape sequences needed to effect specific actions for the terminal.

The TTF changes represent a good deal of work and the menu manager will depend on them.

[These TTF changes were started by Bernie Greenberg in an attempt to provide emacs with a standard way of driving terminals. It is expected that emacs will use the new facilities.]

The Basic Strategy of the Menu Manager

The basic strategy is to intercept all read requests to the terminal done by the process and to respond to them according to information found in the compiled screen object segment. In particular, by the time a read is done a menu has already been displayed and the user is expected to merely type in the number of the option he wants executed. This is simple "menu input" scenario. By responding to an option the user has implicitly selected a command line to be executed, or as in the case of subsystems such as read_mail, an input string to be returned to the subsystem.

In order for this technique to work, it might be easy to intercept all terminal I/O. This, of course, is the case with Multics and the user-ring I/O system, iox_.

Non menu input capabilities are also required. This is because application programs must gather information from the user dynamically about what the user wants done. For this kind of input, which I call bottom window input, new software, was also required. This is because the entire video screen must now be managed (lines and characters counted, etc.) in order to prevent the menu from being scrolled off of the screen.

[The final version of this bottom window I/O software has not been written. I believe a new and general-purpose "forms" mode input program should be written to aid application programmers but this is not presented here.]

Data Structures and Program Interaction

The following diagram indicates the rough inter-connection of the programs and data used by the menu manager.

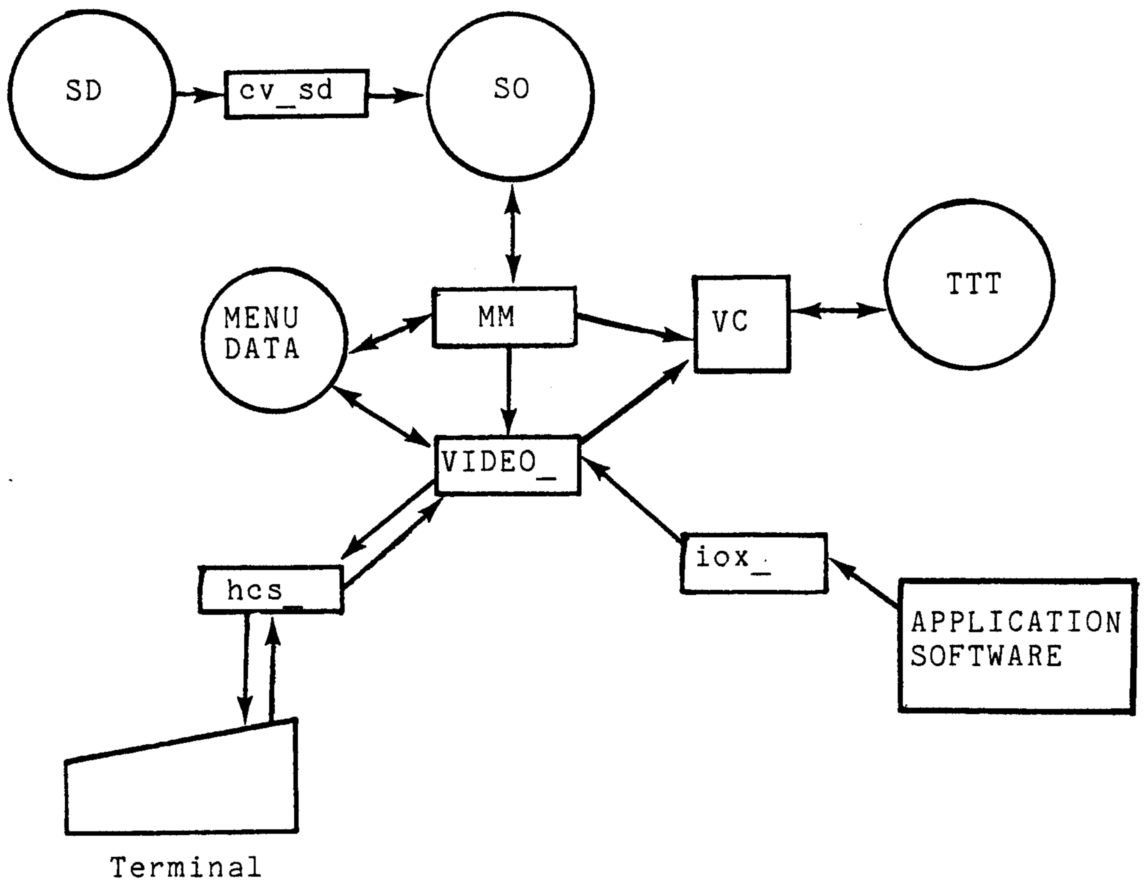

Figure 2. Data Structures and Program Interaction

<u>The Actual Program Structure</u>

The software developed so far can be broken down into five categories as follows:

1. The menu_manager itself,

2. The screen definition compiler,

3. The TTF programs,

4. Programs to help "put up" applications with, and

5. Application programs.

I will briefly describe much of the main programs in the above five areas with the intent of letting the reader see how they fit together and hence how they can or might be extended.

The menu_manager software ...

menu_manager (mm)

This is a Multics command that accepts one argument, the pathname of a screen object segment created by the cv_sd compiler. It sets up the process environment by attaching the standard terminal I/O switch (user_i/o) through the video_ I/O module. In so doing this it initializes the input line editing program (mm_line_editor_) and the program that interfaces to the new TTT.

Once the appropriate initialization is done it displays the first menu and awaits input. (The input must be the selection of some option, or high-level function-key driven function such as "help".)

The menu_manger command sets up cleanup handlers as appropriate and cannot currently be called recursively.

There is an alternate entry point into the menu manager that does nothing more than attach the I/O via video_. This entry, ring_4_video, can be used to get the line editing and "more" processing available in the bottom window by treating the entire screen as the bottom window. (This feature has been pretty much obsoleted by Multics-mode in emacs.)

A further entry in the menu_manager routine is used if the terminal being used does not have function keys. The affect of this entry, mm$no_function_keys, is automatically done for "printing" terminals which rarely have function keys.

video_

This is an iox_ I/O module that sets up special routines for the get_line and put_chars entry. All others are passed through. These entries are needed to avoid scrolling the menu off of the screen when I/O is done at the bottom of the bottom window.

mm_line_editor_

This entry is called by video_$get_line (which is actually implemented in menu_manager). It reads characters from the terminal, in raw and breakall modes, and does real time editing on the partially collected input line. When a CR is received, the input line is returned to the caller of iox_$get_line.

The screen definition compiler ..

cv_sd
: This is the compiler for the screen definition source language. It is implemented with the reduction compiler and behaves in the standard way compilers behave on Multics. The error recovery logic is, however, fairly primitive.

display_sd
: This command takes the pathname of a screen object segment and generates a listing of the various screens defined therein. Two output formats are available. the first lists the various options for each screen and the second, gotten by saying display_sd_$long, generates a picture of each menu as it would appear on the screen--with headers and trailers filled in appropriately.

The TTT programs ...

[Note that the entire TTF issue is currently being acted upon by Larry Johnson and that the programs listed here represent an interim solution only. The long-term solution will clearly involve most or all of these programs but the details and functionalilty provided will be the subject of another MTB.]

cv_ttf
: Compiles a ttf into a ttt. the major difference between the new ttf and the old ttf is the inclusion of video information with each terminal type. Currently this information includes items such as the escape sequence needed by terminals of that type to do, say, a "delete lines" function.

Eventually, the ttf will also include information keys (how many? program settable? if so, how many characters?), color, and other video terminal characteristics.

display_ttt, ttt_info_
: Upgraded to understand and report the new video information.

video_control_
: This is a new program that interfaces the new ttt. There are many entries which return the specific character string needed to effect a particular behavior on the terminal. For example, to cause a line to blink (if the terminal has the capability) one might:

```
char_string = video_control_$blink();
call ioa_ ("^a", char_string);
```

after positioning the cursor to the beginning of the line -- with another pair of calls to video_control_ and ioa_.

The new video_control_ interface being developed by Larry Johnson expands upon this overly simple interface to include buffering, padding, and error reporting.

Programs to help "Put up" applications ...

Most of the programs used to set up applications are part of the standard Multics product: exec_com, abbrev, do, and the host of active functions available. I mention here a few (not so private) tools used to develop the applications to date.

assq
: An association list/look up program. This program is a very convenient replacement for segment property lists as well as a simple table lookup mechanism.

setf, valf
: These are replacements (standins) for the long awaited "exec_com variables". Some such capability is needed. The value command has string limitations. The MIT memory command may be satisfactory, but the best solution is an integrated "variable" capability.

getstr, valid_numberp, valid_datep, ...
: Several new active functions were implemented mainly, to aid in error checking of user-supplied input but also for the various side-effects that interacted with the setf and valf variables.

build_segment
: A program to collect input into a file. This program may be extended to be a simple version of an emacs-like (video) editor.

Application programs ...

Most of the application programs used already existed before pulled into the various menu_driven packages. The most important are read_mail, send_mail, memo, lister programs, and various text editors. [EXL or even newer versions of some of these were used.] A few new applications were developed ...

calendar_add

A "day-at-a-glance", "week-at-a-glance", "month-at-a-glance" type desk calendar program. This is planned to be expanded to include a multi-calendar segment scheduling capability. (Problems exist in that extended access is needed on a more general class of segments than ring-1 message segments.)

doc-file, doc-file editor

A fairly extensive emacs extension that provides much of the power of emacs but uses a simpler (same say) function-key interface. Associated with the emacs extension is a simple text-formatter that is used to generate the final printable output as well as generate "table of contents" and "index" output.

mini

A subset of the doc-file editor, using the same function keys, but for use in reading/sending mail, creating and editing simple text segments, and the like.

Unresolved Problems and Needed Extensions"

The following paragraphs name and briefly describe some of the important problems that need to be addressed. The list is, of course, not exhaustive but most of the problems I have encountered are included.

Performance

The performance of the menu manager itself, although not yet measured, is probably not a problem. The performance of the bottom window input line editing may well be a problem--it uses breakall mode and although well coded, much of the function can probably be moved to ring 0 or the FNP and thereby improve response time as well as CPU time overhead.

The solution to this breakall mode performance problem should be applicable to emacs as well. A study/design project should be initiated as I believe both emacs (including its extensions) and the menu manager will be important products in the future.

Lister

The lister programs are a very attractive vehicle for generating many applications. Needed, however, are several extensions including:

-- record IDs

-- active funtion interface

-- new commands to assign to fields of records and add records to lister files.

A new interface which uses some version of forms processing for filling in the fields of a record is almost a necessity. (A forms interface to MRDS has been developed by FORD -- this should probably be looked into.)

New Menu Features

A "GO TO MENU" capability that avoids clear adherence to the strict hierarchial ordering of menus would be very useful, particularly on slow speed terminals.

A session metering/logging capability would also be very valuable to help us determine how menus are being used, which applications are being used, and how effective the menu manager is in bringing Multics to a place where it can solve real-world problems.

The ability to have dynamic menus would be very useful.

The use of the RETURN key is inconsistent and continues to cause problems with new users (due to the type-ahead features of Multics). Other menu systems have an explicit "EXECUTE" key that might be better.

The ability to display two menus at once, the first being a set of "action" options and the second being a set of "operand" options, would be very desirable. Several (or none) operands could be selected for a simple action.

A new way to abort output at "end-of-screen" time (a problem of Multics in general) is needed. The QUIT action aborts too much. Possibly, a flag (in the terminal output data structures) could be provided. This flag gets turned on at "abort output" time and back off when a subsystem request loop is reentered (qedx, read_mail, listen_).

It has been proposed that the various options of a menu be distinguishable depending on whether a new menu will be displayed if the option is selected.

A method of scrolling back output in the bottom window is needed. [One solution is with the use of the mini emacs extension.]

The current limitation of 2 columns of options should be removed (the documentation already indicates 4 columns are possible).

Optimizations to the menu redisplay function are needed.

A better method of "more" processing is needed so that input typed at end-of-screen is not discarded (this works now in full Multics).

Although the handles are there, the current menu manger will not operate correctly on terminals that are not 24X80 in screen size.

Support of "glass tty" terminals is needed. These terminals work much like printing terminals but need the ability to do "end-of-screen" (more) processing.

Application Support Systems

Variables in exec_com would be extremely useful. Nested if-then-else clauses and begin-end constructs would also be quite useful. An exec_com active function would be very useful.

Many new active functions should be provided to help application programmers verify the validity of input.

A "FORMS" package is needed both to generate forms and also for data-base update applications.

A dictionary for spelling checking and hyphenation would be valuable (the current dictionary is too small).

Extended Support for Video terminals"

A mentioned earlier, a ring-0 or FNP input line editing and screen management facility is probably needed to satisfy response-time requirements, as well as to improve performance. This is, of course, a large design and implementation project -- maybe we can benefit from the CP-6 effort or possible the Level-6 efforts.

Video features that need to be addressed are: redisplay, scrolling, input editing, erase and kill processing, video attribute management (blinking, inverse video, low intensity color, etc.) multiple character sets, protected fields, and many more.

The above video support might well be in ring 0 and hence largely in PL/I. It might then be possible to implement an inextensible subset of emacs in PL/I and thereby achieve considerable performance gains.

Extended Access for Application Data Bases

A version of the ring-1 extended access for message segments should somehow be made available to lister files and MRDS data bases. There are many possible solutions -- the need is repeatedly pointed out.