TO: MSPM distribution
FROM: K. J. Martin
SUBJECT: BX.0.00, Overview: Use of Commands in Multics
DATE: February 2, 1967

Section BX.0.00 has been updated to incorporate changes which have taken place elsewhere in Multics. These changes are: specification of symbol tables, specification of the segment housekeeping module, replacement of meta-arguments with interjected commands, and specification of system options.

Published: 03/04/67
Supersedes: BX.0.00, 01/27/66

Identification

Overview: Use of commands in Multics
D. Eastwood, G. Schroeder, R. Sobecki

Purpose

This paper attempts to define the various modules which make up the Multics command structure. The various responsibilities and features of these modules are discussed in the light of console usage and absentee usage.

How to Get Started

After a user is logged in (see BQ.3.02 for a discussion of logging in and BX.3.01 for usage of the login command) he begins in a procedure called the listener. The listener is equivalent to a read loop on the device from which the user wishes to issue commands - usually this device is a 1050 or Teletype console. In this paper console will be used to denote the command input device. When the listener reads a complete message, it calls the shell procedure.

When the listener is the current active procedure in a user's process, the user is said to be at command level.

The shell is the procedure that receives a completed message from the listener. This message is interpreted by the shell as a call to a procedure.

Functions of the Shell

The shell takes the character string of a message from the listener and breaks it into substrings according to syntax conventions described in BX.1.00. The first substring is assumed to be a procedure name. If the name is of the type segment$entry (the shell recognizes $), the linker is called to establish linkage between that <segment>$[entry] and the shell. The linker returns a pointer to a fault word pointing to the newly established linkage section. If the first substring of characters has no $ in it, the assumption made by the shell is that the user is calling a segment and entry of the same name. Linkage is established then for segment$segment.

The Shell also reads the symbol table associated with the specified segment for a description of the parameters for the requested entry within that segment. See BD.1.00 for a description of the symbol table format.

The rest of the substrings of the user's message are assumed to be arguments to the procedure requested. Using the data description of the parameters associated with the procedure entry, the shell attempts to convert the argument strings to the form expected by the procedure. If at this point the shell detects an invalid argument, it signals its caller that an error has occurred.

After the arguments have been set up by the shell in the specified form for the procedure, a call is made to that procedure by performing the instructions which save the bases and registers, establish the argument list, and save the return, then transferring to the fault word returned by the linker when linkage was established.

The segment is located by the segment housekeeping module, usually with the help of the search module. The data base that controls the search module can be provided by the user; the algorithm for finding needed segments is, therefore, under user control. Of course there will be a default search algorithm provided by the system. See BX.13 for a discussion of user control of searching.

After a call to the requested procedure is successfully made by the shell, that requested procedure is in control of the user's process. The shell does not gain control again until that procedure returns to it, or calls it. The requested procedure may perform many functions for the user, accepting input and giving output to the user. Usually this input and output is done on the device from which the user issues commands, although he may specify other devices. The procedure uses stream names for I/O with the user; the actual device being used is usually, therefore, of no concern to the procedure. When the requested procedure completes its task, it returns to the shell. The shell returns directly to the listener; the user is then at command level.

The shell, then, can call any procedure for the user, provided:

1. The request for the call is in a form understood by the shell (i.e., the request must obey the syntax rules of command language),

2. The arguments can be properly converted by the shell,

3. The procedure can be located by the segment housekeeping and search modules.

System Options

One of the facilities of the command system is the definition and maintenance of system options. System options are a grab-bag of features which control certain aspects of the behavior of Multics toward an individual user. For instance, the user may or may not get listings from compilers and assemblers depending on how he sets a particular system option.

System options are set by using a special command. They are maintained in a special data base belonging to the user, and they are checked by those system commands to which they apply. System options may be set permanently or for the duration of a console session, i.e., until the user logs out; furthermore options can be temporarily reset for the duration of a procedure through the use of a feature of the command language called the interjected command (see BX.1.00). For a further discussion of the use of system options and for a discussion of how this feature can be extended to user-defined options, see BX.12.

The Special World of System Commands

Although the shell may call any procedure specified by the user, it will be used mostly for calling system commands. A system command is a procedure which is maintained by system programmers in a special directory, the Multics Command and Subroutine Library. The system commands are utility procedures which allow the user to use the facilities of the Multics system with minimum difficulty. One way to put it is that these procedures describe the Multics system to the user; they are written by professional programmers for the convenience of the average console user.

Since the system commands are independent procedures written by many individuals, it is an administrative task to make the commands a coherent unit with convenient standards. There are some obvious rules to follow in doing this task.

1. There should be a standard editing facility for text. This editing program should be callable by any system procedure that wishes to use it.

2. There should be standard handling of all common errors.

3. All commands must respect system options.

4. All commands must be able to accept input from and give output to all feasible I/O devices.

For a more complete list of rules, see BX.0.01.

Perhaps to summarize this section one can say that users must receive consistent, courteous, and reliable service from system commands.

Absentee User Processes

In Multics, it will be possible to run absentee user processes. An absentee user process is one which has attached to it no I/O device operated by a person. There are many problems attendant to operating such a process correctly. The most obvious problem is perhaps input and output for the user's requested procedures and for the listener. The solution to this problem is to designate standard stream names for user input and for user output. These stream names can be attached by the user to files in his directory to which he has proper access.

Another difficulty is that many procedures wish to ask questions or make requests and expect answers. Sometimes the questions or requests may be unexpected, and a user preparing an absentee job would be hard put to prepare for them. A proposal for solving this problem is to require all procedures that wish to ask questions to respect a no-questions option the user may set. This means that all system commands that wish to ask questions must define a default action requiring no input from the user in case this option is on.

Summary of the Multics Command Structure

The shell is exactly what its name implies, i.e., a thin layer between the user and the procedure being called. For all practical purposes it is invisible to the user. It handles some errors for the commands and it controls segment housekeeping.

What the shell cannot do is control the execution of commands; the shell merely passes control to commands. The responsibility for a coherent command system, therefore, lies with the design of the command language and the commands themselves.

The next section of this manual attempts to outline rules for command writing that hopefully serve as guidelines for building a coherent and useful set of commands.