Implementing Forth for the Multics Operating System

Michael A. Pandolf
Principal Software Engineer
Cambridge Information Systems Laboratory
Multics Development Center
Honeywell Information Systems
4 Cambridge Center
Cambridge, MA 02142

Like many tools and subsystems on Multics, Forth resulted from a wish of one of its users (in this instance one of its developers) to add to Multics's range of computation. After a period of development and some local exposure, such spontaneously generated software enters a stage of peer review and wider exposure that enables it to be considered for product review. Currently, a version of Forth that follows the design as described in the "fig-FORTH INSTALLATION MANUAL" by the FORTH INTEREST GROUP is in the local exposure state at MDC/CISL and is not yet scheduled for peer or product review.

Multics has traditionally been a very willing host for the simulation or emulation of other programming environments. One reason for this is a library rich in programming tools that is easily accessible when programming in the Multics native language, PL/I. However, programming at a lower level than PL/I (and its subroutine library) results in low level support structures that are not altogether friendly without high-level support. While implementing a version of Forth for Multics, the author found several of these support structures that directly influenced the design of the resultant subsystem, including the virtual memory itself, Multics object segment structure, and the native I/O system.

The Forth dictionary is a read/write/execute structure. Although Multics allows a segment to have all three access modes simultaneously specified, write permission and execute permission are, as a rule, mutually exclusive. All language translators, including the assembler, produce pure code; Multics runtime expects this rule to be in effect. The use of pure code is one of the assumptions made by the storage system as it makes a segment known to a process; the actual segment (not a copy of the segment) is added to one's address space. This permits nearly all code in the system to be completely shareable. Modification of the dictionary contents while executing Forth discourages the dictionary from being shared. To circumvent this feature, the minimal default dictionary is copied into a per-process temporary segment and used there.

A more critical problem related to the dynamic nature of the dictionary and the Multics expectation of pure code is Multics runtime support. Multics runtime support expects that executable segments have a particular structure. Although Forth can provide all data manipulations using code within the dictionary, it must invoke the supervisor to perform I/O. In Multics, the supervisor is invoked using the same mechanism as any unprivileged call. The ring of the calling code segment is compared to the ring of the target; the hardware recognizes the call as one that invokes the supervisor and adjusts several registers accordingly. This call/return mechanism depends upon the existence of several memory-resident data structures, which includes the code segment's external reference definition section. It takes the addition of only a few words to the dictionary to overwrite the definition section. Multics does not employ a special trap instruction to invoke its supervisor thereby requiring that I/O be performed with a call at some point. One method of accomplishing this is provided by the PL/I environment's condition handling mechanism. This method simulates the more traditional supervisor trapping construct by having Forth execute one of the unused trap instructions causing a PL/I on-unit to be activated. Though rather elegant, a mechanism with less overhead is desired because condition handling would cause the supervisor to be invoked to verify that the fault was not system critical before passing it on the the user ring environment where further processing would take place. A faster, more direct bit of hackery is employed, instead. In Multics Forth, a gateway written in assembler provides an interface with Multics runtime on one side and with the Forth environment on the other side. The Forth interface of the gateway reserves a pointer register which provides the only way back into Multics from Forth. The gateway also reserves one pointer for its own storage, cutting only slightly into the available register compliment for Forth. The PL/I interface is used by the Multics "forth" command after first creating an address space for the Forth runtime structures. The gateway stores register contents upon invocation from either side, essentially swapping two different environments.

The remaining design issues are, primarily, esthetic concerns but there is a Multics-like way of doing things that tends to flavor its facilities. From the point of view of a Multics programmer, it is desirable to exploit the segmentation offered by Multics. While the model used for Multics Forth runs in linear memory, the protection offered by separate segments for stacks and I/O buffers encourages one to divide up the linear address space. The data structures are intuitively distributed to one of four segments: a data stack segment, a return stack segment, a dictionary segment, and a utility storage segment. Each segment has a capacity of 256K words (1Mbyte) and is

physically isolated from any other segment. In this way, stack overflows, stack underflows, and some types of invalid addressing are hardware detectable and do not effect other data, either related or unrelated. Multics can support Forth running in either segmented or nonsegmented modes with equal ease; there is no system imposed bias on the choice.

The only important design issue in address architecture concerns the address itself. When running in a single segment, addresses are eighteen bit quantities that reference a location in the segment. When running in segmented mode, addresses are at least 33 bits wide. Forth manipulates addresses as integers, so the address must be capable of being used in addition and subtraction. A complete Multics pointer value is not suitable for use as a Forth address because it is two machine words long, must be even-word aligned, and its segment number and word offset fields are not easily manipulated. A Multics packed pointer is one word long and useable with one restriction: the bit-within-word value, found in the high order six bits must be set to zero thereby limiting the address to specify word boundaries only. The offset field is in the low eighteen bits of the word and lends itself to simple numeric manipulation. There is a generic problem with storing a virtual (segmented) address in Multics: segment numbers are dynamically assigned and are valid only within process boundaries. Permanent storage of these addresses requires a conversion method at startup and/or shutdown time that is not necessary when referencing offsets only. Maclisp on Multics stores segmented addresses and performs an address conversion. Forth could do the same, although the need for such a facility has not arisen. The complexity added by running Forth in a segmented environment is minor compared to the advantage obtained with partitioned data; therefore, the Multics implementation is segmented.

Terminal access within Forth is based upon a system developed for the Multics Emacs text editor. Both receive input one character at a time and perform their own editing. This is in contrast to Multics's line-at-a-time processing where a line of text is built in the communications processor and sent to the central system. Character-at-a-time I/O to a program has been shown to have a deleterious effect on Multics's response time when used by several processes simultaneously. This is because character processing is intended primarily to be performed by communications processors, whereas character-at-a-time processing involves the central system. A solution developed for Emacs instructs the communications processor to echo and build a buffer of uninteresting characters and inform the central system when an interesting one comes across. This mode of communications, called "echo

negotiation," is the standard mode of communications for Forth.

Disk access for Forth can be considered to be simulated. Forth does not reference a disk drive directly. When invoking the forth command, one is allowed to specify a segment that is used as a one megabyte disk. Reading in a disk block actually reads in part of the segment. Remembering that the system makes no distinction between main memory and disk storage, the changes made to the segment in the course of the Forth session will be reflected on a physical disk. The group of segments used as Forth disks can be arranged as a library of disks at the Multics command level.

The runtime structure of the Multics Forth environment begins with the invocation of the forth command. The command program is written in PL/I and serves as the monitor base for the rest of the system. It optionally takes as its arguments the names of Multics segments to be used as virtual disks. After obtaining temporary storage segments for its own structures and for the four segments Forth requires, it loads a copy of the default dictionary into one of the segments. Finally, it calls the gateway program: forth_bootstrap_. This program is written in assembler and has as its basic functions the saving of register contents from the PL/I environment, the transfer of addresses from automatic storage in the forth command to hardware registers (most notably pointers to the stacks and utility area), and transfers to the dictionary. The dictionary is coded in assembler and is roughly 1.5K words long. It contains the words necessary to load the rest of Forth from the virtual disks identified at command time.

To summarize, under the Multics operating system Forth runs as a low level program contained within four segments. Its interface with Multics is through a gateway program that transfers back to the PL/I environment. The PL/I program base forwards its I/O requests to the Multics supervisor through standard subroutine calls. Forth communicates to the user through the process's terminal I/O channel and uses segments as virtual disk storage. The system is currently under development.