Published: 08/24/67

Identification

Using GECOS programs in Multics
D. B. Wagner

Discussion

There will be various occasions on which it is desirable to make programs written for the GE 635 run under Multics. The two such programs which are most important are:

1. The EPLBSA assembler (See BN.8)

2. The TMG table interpreter (See BN.4)

The EPLBSA assembler is almost all written in FORTRAN IV, so the final answer to using it in Multics will be to do some minor editing and compile it using the Multics FORTRAN IV compiler. Unfortunately the compiler will not be ready at the time EPLBSA is needed, so an interim measure is required.

The TMG table interpreter is a more difficult case. It was written originally in CDC 1604 assembly language, edited over to IBM 7040 assembly language, converted to IBM 7090 BEFAP by defining macros for the 7040 opcodes, and finally converted to GE 635 GMAP by further defining macros for the 7090 op-codes. The program has gradually lost almost all its commentary and any trace of coherence it may ever have had. There is little likelihood that any further mechanical translation of this program is even possible.

In addition Multics will be a much easier system for people to get used to if we can easily carry over:

3. Most ordinary scientific programs written in FORTRAN IV and GMAP.

The basic method which has been chosen for carrying all these programs over to Multics takes advantage of the fact that the GE 645 is simply a GE 635 with the appending hardware and a collection of new instructions loosely hung on. If a program is restricted to operating within a single segment on the 645 it can easily be made to believe it is running on a 635. The gecos_seg command (see BX.17.01) accepts as input a collection of GECOS relocatable and absolute program decks and produces a single (unfortunately impure) procedure segment. Means will be available for interfacing with ordinary Multics procedures in accordance with Multics standards.

Hard-core GECOS functions (such as IO) which are called with MME instructions as discussed later, are performed by the special MME-catcher procedure gecos_mme discussed in BX.17.02.

These GECOS object decks can be obtained by assembly or compilation off-line while Multics is not running. However it should be possible to use the same method to carry over the following, which would make maintenance much easier:

4. The GMAP assembler

5. The GECOS Fortran IV compiler.

GECOS Program Interfaces

Programs which run under GECOS communicate with each other and with their environment using any of the following mechanisms. GE documentation is cited by document number below. The documents cited are:

CPB-1004D	GE 625/635 Programming Reference Manual (includes GMAP)
CPB-1195	GECOS manual
CPB-1008	GELOAD manual

1. Normally all subroutine calls (including all calls compiled by FORTRAN IV) are made using the standard GMAP CALL macro (described in CPB-1004D, p. II 76). This macro provides a special mechanism for error returns and some special error linkage information.

2. References to external entries are made through the GMAP pseudo-ops SYMDEF and SYMREF (described in CPB-1004D, pp. III 46-47). These provide direct linkage at load time: there is no "transfer vector" as with the BSS loader. Data is often shared among subroutines using this mechanism also: in particular TMG accesses its driving table using SYMREF's.

3. Data is more often shared among subroutines using the BLOCK pseudo-op of GMAP (described in CPB-1004D, p. 57). The "labelled common" and "blank common" of FORTRAN IV are implemented using this mechanism, for example.

4. Some GECOS functions are invoked using the MME (master mode entry) instruction, which causes a fault to the supervisor. A list of these functions is given in CPB-1195, pp. 103-124. They include overlay processing, the basic I/O functions (normally used only through the library I/O package GEFRC), and a collection of simple system functions such as getting the date and time, terminating the job, etc. The FORTRAN IV compiler apparently does not compile any MME instructions, but some of its run-time library subroutines must use them.

5. Some special GECOS debugging functions are invoked using the DRL (derail) instruction. See CPB-1008D, pp. 33-37.

6. Programs written in GMAP can use a great many dirties communicating with each other. Some parts of EPLBSA, for example, depend upon otherwise irrelevant details of the GECOS loader such as the fact that it loads from the top of allocated memory down.

An example: the method for EPLBSA

EPLBSA consists of 35 relocatable subroutines, 27 written in GMAP. Two (INPUTS and OUTPUTS) are GMAP- written interfaces to GEFRC input-output functions. A few FORTRAN library subroutines are used. The technique used to carry EPLBSA over to Multics will be roughly as follows; it is expected that a similar technique will be useful in carrying over TMG.

The simplest way of creating an EPLBSA command in Multics is as follows:

1. Get the 635 object decks for the various pieces of EPLBSA into Multics. Use gecos_seg to make a procedure segment from these. Give this segment some strange name.

2. Write in EPL an interface program (called EPLBSA) which sets up the GECOS master mode entry simulator as fault-catcher for MME's, calls the various initialization entries in this fault-catcher (described in BX.17.02), and makes a standard call into the segment created by gecos_seg.

This simple method leaves something to be desired in the way of efficiency. In particular a lot of unwieldy machinery is involved in simulating GECOS input-output. Further, we would like to minimize the amount of impure procedure floating around, especially in a program as heavily used as EPLBSA. Therefore two more steps may be desirable.

3. Write EPL versions of EPLBSA's two master I/O routines, INPUTS and OUTPUTS. Use the call_out control line in gecos_seg to specify that calls to these routines should be translated into Multics standard calls to the EPL versions.

4. Rewrite in appropriate languages any pieces of EPLBSA which look easy to do. Then these can be used as in (3) to reduce the amount of impure procedure.

Note that step 3 means that EPLBSA does not need the rather ugly GECOS I/O simulation machinery. If EPLBSA were the only program we were interested in, this would be a vastly easier approach. TMG is less tractable in these matters, however.