Published: 08/25/67

Identification

GECOS master-mode entry simulator
gecos_mme
D. B. Wagner

Purpose

The procedure gecos_mme is used to simulate master mode functions called by a 635 program jobbed into Multics using the gecos_seg command (see BX.17.01). It simulates most GECOS functions and allows the user to handle any which it cannot simulate.

Usage

Before passing control to the "jobbed over" 635 procedure segment, the user must make several initialization calls. The first is:

```
call gecos_mme$init;
```

Then calls may be made to specify how various file-codes should be treated by the simulated IO functions; these are described later.

Then any GECOS functions which the user may wish to handle himself are specified through calls of the form:

```
call gecos_mme$callme (name, routine);
```

where name is a character-string giving the name of the function (e.g. "GETIME") and routine is an entry to be called to handle the function. The call to routine has the form:

```
call routine (p);
```

where p is a pointer to machine conditions at the time of the fault.

Finally calls may be made to activate some simple debugging mechanisms inside gecos_mme.

Easy GECOS Functions

The following GECOS functions seem reasonably easy to do in terms of Multics and need not be discussed further here:

GEFADD	(Physical file address: return nonsense since it makes no difference)
GEFINI	(do a signal finish)
GEBORT	(do a signal error)
GESETS	
GERETS	
GETIME	
GEMORE	(for memory requests only - no problem)
GEENDC	(Used with "courtest calls" in GEINOS: see later)
GEFCON	
GEROAD, GERELC, GEMREL, GELOOP, GESYOT	(Simply accept and ignore the call)

GECOS Functions not Simulated

The following are hard to do and will not be simulated unless a good reason presents itself. GECALL is used in one or two places to call the loader to load special things from libraries (GMAP, for example, loads system macro definitions this way). It should be special-cased by any user who needs it.

GESNAP	
GELAPS	
GEMORE	(other than memory requests)
GERELS	
RECALL	
GESAVE	
GERSTR	
GECHEK	
GEROUT	

GECOS Input - Output Functions

GECOS I/O will be simulated initially only for the files known as "linked", that is, disk, drum, and tape files which can be treated as linear sequences of records, like tape.

The user must specify what segments are to be read or written when various GECOS file codes are operated on. He does this through calls of the form:

```
call gecos_mme$linked_file (code, segp, length);
```

The parameters are declared,

```
dcl code char (2), segp ptr, length fixed bin;
```

Here code is the 2-character file code, segp points to the base of the segment, and length is the length in bits of the segment.

If gecos_mme is updated to provide for simulation of random-access devices, card equipment, the printer, or the on-line console, initialization entries must be provided as follows:

```
call gecos_mme$drum (code, segp, length);
call gecos_mme$disk (code, segp, length);
call gecos_mme$typed_input (stream);
call gecos_mme$typed_output (stream);
call gecos_mme$printer (segp);
call gecos_mme$card_reader (segp, length);
call gecos_mme$card_punch (segp);
```

Most of these calls should be reasonably clear, and will not be discussed further. Stream is a Multics I/O-system stream-name.

Whenever a file is written in "decimal" (BCD) mode, it is converted to ASCII before transmission. Similarly, when a file is read in decimal mode, the segment specified is converted from ASCII lines to 80-column GE-Hollerith card images.

Some programs now in use (EPLBSA and TMG) read and write ASCII input out of and into 28-word column-binary card images. For these programs, the following calls are provided:

```
call gecos_mme$ascii_input (code);
call gecos_mme$ascii_output (code);
```

Each specifies that the necessary conversion from (to) ASCII lines must be made to (from) binary card images.

If the user would rather handle a file himself, he may simply fail to tell gecos_mme about it. When a call is made to operate on a file which gecos_mme does not know, it does a

```
call signal (code, "1"b, p);
```

(see BD.9.04 for the signal procedure) where code is the file code and p points to the machine conditions at the fault. Thus if the user has earlier called condition he will catch all calls for this file.

Fault Simulation

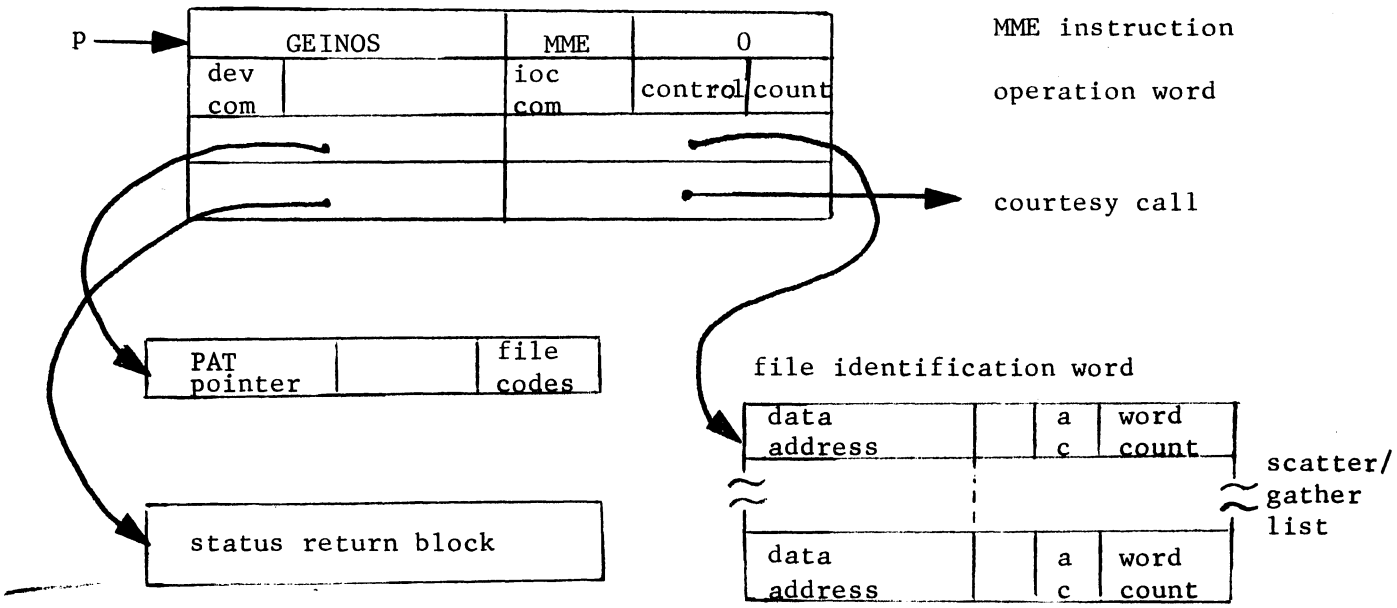
GECOS allows a user to specify his own fault handlers for:

1. Memory (Multics out-of-bounds fault)
2. Divide check
3. Overflow
4. Command
5. Illegal op-code
6. Fault tag (Multics fault tag 1)
7. Derail
8. Connect

Gecos_mme, at initialization, sets up fault-handlers for all of these except command and connect, and when they occur makes the appropriate transfer of control to the 635 program.

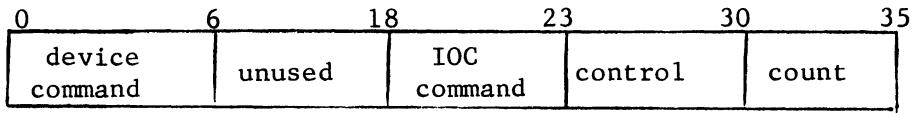
Handling of GEINOS

After a pointer, p, to the faulting instruction has been obtained, as discussed elsewhere, we find that it normally points to something like this



The call for disk or drum I/O is slightly different, and can be seen in the GECOS manual. The basic principles discussed here still apply.

The operation word is described in some detail in CPB-1195, pp. 145-147. Briefly, it looks as follows:



Device Command is a number meaning such things as "Request status" (REQS), "Write card binary" (WCB), etc.

IOC Command is rather peculiar, and appears to be more-or-less irrelevant to gecos_mme. See CPB-1195, p. 147.

Control is zero except in a call to write an end-of-file mark. Since gecos_mme accepts and ignores such calls, this field is irrelevant.

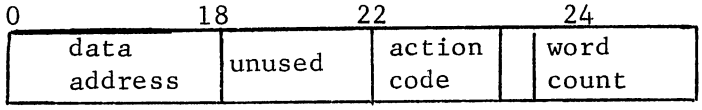
Count serves several purposes:

1. In calls to backspace or forwardspace records, it indicates the number of records to be spaced over.

2. In any other non-data-transfer operations, it is required to be 1.

3. For disc or drum I/O, a seek operation word contains the value 2 in the count field. This indicates that another operation word follows.

The scatter/gather list, also called the DCW list, is a not-necessarily-contiguous list of the blocks of locations involved in a read or write operation. The list is described in CPB-1195, pp. 148-149. Briefly, each word (DCW) in the list has the form:



Action code is a 2-bit number indicating:

00 Transmit [read or write] and disconnect [end of list]

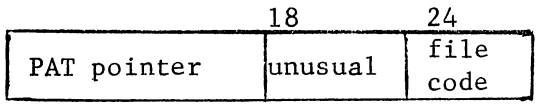
01 Transmit and proceed [go to next DCW]

10 Transfer to DCW

11 Non-transit [skip] and proceed.

Data address and word count have [obvious] meanings determined by the action code. See CPB-1195, p. 149.

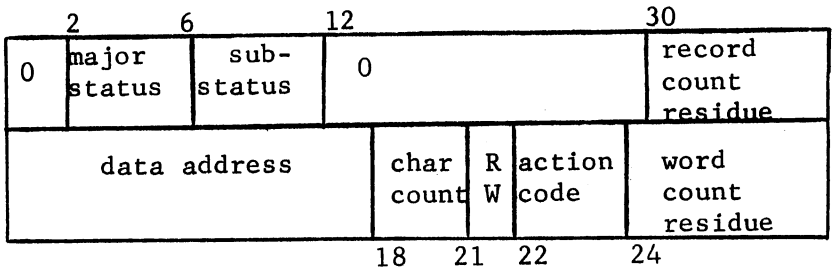
The File Code Word is described in CPB-1195, p. 148. Briefly it looks like:



Here PAT pointer is a user-supplied place that GECOS uses for its own purposes. Gecos_mme ignores it. File code is two 6-bit GE-Hollerith characters.

The Status Return Block is two words of bits giving miscellaneous information about the IO, including all manner of interesting statis like "card-punch chad box full". See CPB-1195 pp. 150-151, 175-215. Most of the status information is irrelevant to gecos_mme, of course. The following is what gecos_mme needs to know.

The status block can be returned as follows:



Major Status and Sub-Status are the only really important parts. Since gecos_mme will accept input-output only on "linked" files, those which look like tapes, only the following are ever returned:

Major status = 0000 Substatus = 0001xx

(All OK. Not clear what xx should be.)

Major status = 0100 Substatus = Single data character ≠ 77(8)

(End of file. Substatus gives the identity of the end-of-file. Since end-files are hand-waved by gecos_mme, we will have to choose a code to return. (Zero should do.)

If card-equipment and printer simulation is ever added to gecos_mme, the following additional status returns will be relevant.

Major status = 0000, Substatus = 000000

(All OK)

Major status = 0100, Substatus = 000000

(End-of-file on card reader.)

If drum and disk simulation is ever added to gecos_mme, the following additional status returns will be relevant:

Major status = 0000 Substatus = 0XXXXX

(All OK on drum. XXXXX = block address at this instant. Gecos_mme should return a random number, like 6.)

Major status = 0000 Substatus = 000000

(All OK on disk.)

If console typewriter simulation is ever added to gecos_mme, the status returns can be seen in CPB-1195, pp. 212-215: "Channel ready", "Message length alert", and "Operator distracted" seem to be the only pertinent returns.

Record Count Residue applies only to backspacing and forwardspacing operations, and gives the number of records not passed over because one end or the other of the tape has been reached.

The second status word gives the condition of the last DCW word processed. It is clearly described in CPB-1195, p. 151.

The Courtesy Call Address is the address of a place to be called when the operation is completed. It is not clear from the GECOS manual what form a courtesy call takes but it seems to be a simple transfer. The user program then returns by doing a MME GEENDC (see CPB-1195, pp. 113).

Working with the Fault Data

A fault-handler like gecos_mme is called by signal with an argument p which is a pointer to machine conditions at the time of the fault. The machine conditions are stored in a block of 23 words as follows:

words	0-7	stored bases
	8-15	stored registers
	16-22	stored control unit.

This can be expressed in the following EPL declaration:

```
dcl 1 mach based (p),
    2 stb (0:7) bit (36),
    2 sreg (0:7) bit (36),
    2 scu,
      3 tbr bit (18),
      3 appending_status bit (18),
      3 computed_address bit (18),
      3 control_frame_status_1 bit (18),
      3 pbr bit (18),
      3 fault_data bit (18),
      3 ict bit (18),
      3 indicators bit (12),
      3 control_frame_status_2 bit (6),
      3 even_instruction bit (36),
      3 odd_instruction bit (36),
      3 ring_no bit (18);
```

There are several documents which give information on the stored control unit. The best are:

645 EPS M50EB00107 pp. A49-A54
CU format G0046

Then the following EPL statement will fish out a pointer to the faulting instruction:

```
q = ptr (ptr$baseptr (p→mach.scu.pbr), p→mach.scu.ict);
```

(See BY.14.00 for the pointer-manipulation routines used here.)

In order to return control to the faulting program in the proper place, a small modification must be made to the stored control unit. Gecos_mme fabricates a transfer to the desired return location and puts it into either of:

```
p→mach.scu.even_instruction
p→mach.scu.odd_instruction
```

depending on the "odd" bit in the "appending unit status", bit 23 of the second word of the SCU data. This bit can be gotten at by:

```
b = substr (p→mach.scu.appending_unit_status_1, 6,1);
```