Published: 05/03/68

Identification

Use of the binder under 6.36
J. M. Grochow and N. I. Morris

Introduction

The binder will be run under 6.36 in order to create bound Multics supervisor modules prior to writing them out on a Multics system tape (MST). It will be desirable to run the binder and the Multics system tape generator (MSTG) during the same 6.36 run, and it will also be extremely advantageous to run both the binder and the MSTG from the same control file.

References

The reader is assumed to be familiar with Multics system tape generation (BL.1.03) and segment binding (BX.14.01). BL.1.03A is also recommended.

Overview

In order to integrate binding and the MSTG, the overlay-driver and the binder-driver (to be described individually in more detail later) have been written. The overlay routine is a program designed to run several different programs during one 6.36 execution activity (such as the binding operation and Multics system tape generation). It is, in many ways, similar to the FMS "chain" feature. In this particular application, binder routines will be read-in and control will be transferred to the binder-driver. After all segments specified are bound, the binder-driver will return to the overlay routine. The overlay routine will now delete those segments not needed for the MSTG (e.g. the binder_driver and the binder itself) and read in the additional segments necessary. Control will be transferred to the MSTG and a Multics system tape will be written using, in addition to others, the bound segments just produced.

To make a binding run:

In order for a user to make a binder-MSTG 6.36 run, it is only necessary to create an MSTG header file (see description below and examples) and an appropriate GECOS file. The user need not concern himself with the overlay routine or the overlay data file.

The GECOS file should contain the following.

1. TL's (or assembly/compilation and LD's) for all segments to be bound. T234 CF1
2. INSERT BINDER GECOS (this file contains TL's and LIBE's for the various binding, overlay, and MSTG programs and a MAKETL for an appropriate overlay data file).
3. MAKETL α β MST_SEGMENT_LIST DATA,SLVACC (α β is the ASCII header file to be described below).
4. If the text, link, and symbol segments for any bound segments are required to be returned, the user must have additional MRGEDT commands of the form:

```
FETCH γ TL      (return to CTSS)

DECK  γ         (punched output)
```

where γ is the first name of the file to be returned. γ must be the first six characters (or less if the name is shorter than six characters) of the bound segment name (excluding the "bound_" prefix-see naming conventions below). ASCII underscore ("_") is converted to dash ("-") as is normally done in MRGEDT control files.

BINDER GECOS includes the following statements in addition to those already mentioned:

```
LIBRARY   TAPE    MULTICSEGLIBE

ERROR

LIMITS    95K

FETCH     TL

PGSIZE     0

LSPGSZ     0

NMTBND    20
```

The MRGEDT command is now used in the usual manner. Output will be via files (text, link, and symbol) returned to CTSS or punched output as determined by the FETCH and/or DECK cards.

(Note: a special RUNCOM called BINDER is in T234 CMFL06. This will create all additional links in your directory that are necessary for the binder-MSTG operation.)

MSTG Header File

The standard control file will be quite similar to the old MSTG header file (see Section BL.1.03 and BL.1.03A). The "loadname" option, however, will no longer be available. There will, instead, be a way of dynamically loading a segment from the library into a segment of a different name. Thus, one will be able to load "alloc_" into the segment "xalloc_" by the following statement:

```
name: alloc_ (xalloc);
```

As usual the "path_name" must follow the "name" statement. If the segment is a bound segment and binding is to take place during the run, the next statement must be as follows:

```
bind: yes;
```

If the statement is not supplied, no binding will take place. If the segment is a bound segment, the next statement must be as follows:

```
bind_names: α,β,γ,δ, . . . ;
```

Renaming of a dynamically loaded segment may be specified as in the "name" statement. The binder will create the segment named in the "name" statement by binding segments α, β, γ, δ, . . . etc., if and only if the "bind" statement specifies that binding is to take place. The MSTG will add the names α, β, γ, δ, etc., to the names supplied on the "name" statement.

The name of a bound segment should start with the string "bound_" and should attempt to describe the general content of the segment bound within, e.g., "bound_page_control", "bound_mstg", etc.

If the segments to be bound are obtained dynamically from the library, it is advantageous to have the binding list in alphabetical order (i.e., runs will take substantially less time.)

The Binder Driver

binder-driver has been written to accept a data file listing segments to be bound, call the binder appropriately, and return bound text, link, and symbol segments through the 6.36 return tape mechanism to CTSS.

It is called as follows:

```
call binder_driver (data_file);
```

where data-file is the name of the binder control file (data_file is "mst_segment_list" in combined MST-binder runs).

Since the interim binder requires that text, link, and symbol sections be available for all segments to be bound, a check for this will be made in the driving program. If any of these segments are missing for a particular segment name, a message will be put in the user's error file and the binding operation will continue. The segment in question will, of course, not be included in the bound (output) segment.

Segments assembled using the new EPLBSA assembler will produce symbol segments with the appropriate binding information. The standard "TL" and "FETCH * TL" commands of the merge-editor will automatically handle their movement to and from the GE 645.

The (ASCII) data file should be created as file α β in the standard MSTG header file format, as described above. (In MSTG-binder runs, the same file is used for both.)

The Overlay Routine

The overlay routine is designed to dynamically load those modules needed to run a particular job, transfer to the appropriate entry point, and upon return, delete those modules no longer needed. As noted above, it and its control file are provided, through the BINDER GECOS insert in the merge-edit run. For the benefit of potential non-binder users of the overlay routine, we will briefly deal with it in the abstract. It is called as follows:

```
call overlay;
```

The overlay routine recognizes three types of statements in its (ASCII) control file:

name: α, β, γ, δ . . .;

The above statement indicates to the overlay routine which segments are needed for a particular job. If the segments $\alpha, \beta, \gamma, \delta$, etc. are not in core, they will be dynamically loaded.

entry: α, β ;

This statement causes a call to be made to $\alpha\$\beta$. If β is not given, $\alpha\$\alpha$ will be called. Upon return from $\alpha\$\beta$, all segments found in preceding "names" statements will be deleted.

end;

The end statement will cause termination of a 6.36 run. Comments may be placed in the control file by using EPL comment conventions. The name of the control file segment must be "overlay_list".

Examples

An overlay control file for running the binder and the MSTG might look as follows:

```
names: binder_driver, binder, post_binder;

entry: binder_driver;


names: mstg, mstg_output, mstg_writer;

entry: mstg;


end;
```

A typical entry in a binder and MSTG control file might look as follows:

```
name:                   bound_page_control;

path_name:              >bfs;

bind:                   yes;

bind_names:             page_fault, clean_up, pc_free_core,
                        pctruncate, pc_readseg, setmove,
                        remove_page, removept;


access:                 slvprc, slvacc;

status:                 wired_down;

linkage;

initseg:                yes;

temp_seg:               yes;

status:                 normal;

end;
```

Other examples of a header file entry are as follows (for more complete explanation of syntax see BL.1.03A):

```
name:           bound_string_package;

pathname:       >system_library;

bind:           yes;

bind_names:     catstr_ (xcatstr_),
                movstr_ (xmovstr_),
                stgop_ (xstgop_),
                substr_ (xsubstr_);

end;


/* name with "bound_" prefix must appear first */

name:           bound_pre_linker, scan_linkage,
                force_link;

pathname:       >initializer;

bind:           yes;

bind_names:     pre_link_2, /* must be bound first */
                datmk_ {dbi} ;

end;
```