Subject: MCR10102, Change mbuild for Over-length Source Archive Situations

Author: Gary Dixon

Date: November 24, 2021

Ticket: http://multics-trac.swenson.org/ticket/261

In the Multics Libraries, source files for most bound segments are stored as components of a single archive segment: bound_XXX_.s.archive. Like most segments on Multics, archives are limited in size to 255 pages. When source components grow in number or size, the source archive can overflow this size limit. One of two possible remedies have been used for this problem.

1. Split the source archive into two different source archives: bound_XXX_.1.s.archive and bound_XXX_.2.s.archive.
2. Move some of the source components to a new bound_YYY_ object segment, having its own now-smaller source archive: bound_YYY_.s.archive. The original bound_XXX_.s.archive is then smaller in size.

Remedy 1: Split the source archive

To implement the first remedy, perform the following steps:

- Use library_fetch to get a copy the current source archive. For example:
  lf bound_XXX_.s.archive
- Rename bound_XXX_.s.archive to bound_XXX_.1.s.archive.
- Extract and delete from bound_XXX_.1.s.archive the components to be moved from the first archive. In source archives, the components are ordered alphabetically by name of the component. So components in latter part of the alphabet would be removed from the first archive.

  For example, using archive (ac) command and archive_table (act) active function, extract/delete all components with names beginning with: r, t or v.
  ac xd bound_XXX_.1.s [act bound_XXX_.1.s (r t v)*.*]
- Append those components to a new bound_XXX_.2.s.archive.
  ac ad bound_XXX_.2.s [segs (r t v)*.*]
- Use library_fetch to get the bind file for the bound segment. The bind file is the only component of the object archives not generated by source in the source archives. This bind file is needed in the installation directory so mbuild can add it to bound_XXX_.1.archive when it creates that archive using objects compiled from components of bound_XXX_.1.s.archive.

  library_fetch searches only source libraries by default; so tell it to look instead in object (o) libraries.
  lf -lb o bound_XXX_.bind

The mbuild script for this change involves deleting the original bound_XXX_ object segment (which includes its source and object archives); then adding a replacement bound_XXX_ object segment comprised of the two bound_XXX_.(1 2).s.archive segments and the bound_XXX_.bind file. For example:

```
Description:
 A) Split bound_info_rtns_.s.archive into two smaller source archives.

Installation_directory:  >udd>m>gd>w>bir;

 Build_script:             bir.mb;

 Bound_obj:                bound_info_rtns_                  IN: sss  DELETE;

 Bound_obj:                bound_info_rtns_                  IN: sss  ADD;
     bindfile:             bound_info_rtns_.bind              ADD;
   source_arch:            bound_info_rtns_.1.s.archive       ADD;
   source_arch:            bound_info_rtns_.2.s.archive       ADD;
```

Remedy 2: Move source components to new bound segment

If programs in the over-length source archive comprise several different subsystems, it is often wiser to move components of the largest subsystem to a new bound segment, leaving the remaining components in a smaller source archive. This remedy keeps related components of each subsystem together in a single source archive, rather than having them split by alphabetized name across two (or more) source archives.

To implement the second remedy, perform the following steps:

- Use library_fetch to get a copy the current source archive. For example:
  lf bound_XXX_.s.archive
- Extract and delete from bound_XXX_.s.archive the components to be moved to the new bound object. For example, using the archive (ac) command and archive_table (act) active function, extract/delete all components with names beginning with: c, h, i, l or v.
  ac xd bound_XXX_.s [act bound_XXX_.s (c h i l v)*.*]
- Append those components to a new bound_YYY_.s.archive.
  ac ad bound_YYY_.s [segs (c h i l v)*.*]
- Use library_fetch to get the bind file for the bound_XXX_ segment. The bind file is the only component of the object archives not generated by source in the source archives. This bind file is needed in the installation directory so the developer can manually divide its contents into:
  - descriptions in bound_XXX_.bind for components remaining in bound_XXX_;
  - descriptions in bound_YYY_.bind for components moved to bound_YYY_.

  library_fetch searches only source libraries by default; so tell it to look instead in object (o) libraries.
  lf -lb o bound_XXX_.bind

  Copy bound_XXX_.bind to bound_YYY_.bind. Then edit each bind file to remove descriptions of components no longer part of that bound object.

The mbuild script for this change involves deleting components from the original bound_XXX_ object segment and replacing its bind file; then adding a new bound_YYY_ object segment. For example:

```
Description:
 1) Move all help-related commands and subroutines out of bound_info_rtns_ and
    into a new bound_help_ object segment.

Installation_directory:  >udd>m>gd>w>bh;

 Build_script:   bh.mb;

 Bound_obj:      bound_help_                        IN: sss  ADD;
     bindfile:   bound_help_.bind                    ADD;
   source_arch:  bound_help_.s.archive               ADD;

 Bound_obj:      bound_info_rtns_                   IN: sss  UPDATE;
     bindfile:   bound_info_rtns_.bind               REPLACE;
     source:     check_info_segs.pl1                 DELETE;
     source:     help.pl1                            DELETE;
     source:     help_.pl1                           DELETE;
     source:     help_listen_util_.pl1               DELETE;
     source:     help_request_tables_.alm            DELETE;
     source:     help_requests_.pl1                  DELETE;
     source:     help_responses_.pl1                 DELETE;
     source:     help_util_.pl1                      DELETE;
     source:     info_seg_.pl1                       DELETE;
     source:     info_seg_allocate_.pl1              DELETE;
     source:     info_seg_error_.pl1                 DELETE;
     source:     info_seg_parse_.pl1                 DELETE;
     source:     info_seg_specifications_.cds        DELETE;
     source:     info_seg_util_.pl1                  DELETE;
     source:     info_seg_verify_.pl1                DELETE;
     source:     info_seg_verify_iFile_.pl1          DELETE;
     source:     list_help.pl1                       DELETE;
     source:     verify_info.pl1                     DELETE;
```

The Problems

The mbuild subsystem has several problems dealing with the sample build scripts shown above.

In the first remedy, the original bound_info_rtns_ object is being deleted so a replacement with two source archives can be added. But mbuild gets confused when the bound_info_rtns_ object name appears in two Bound_obj statements with different operations: DELETE versus ADD. Those are two separate operations, and mbuild needs to create a BOUNDOBJ structure to describe each of these operations. Right now, it creates only a single BOUNDOBJ structure for the DELETE operation.

Also, when mbuild discovers that a bound object being added consists of more than one source archive, it fails to update the Seg(bindfile) structure to indicate that the bind file should be appended to the bound_XXX_.1.archive (rather than the default bound_XXX_.archive for a 1-archive bound object).

In the second remedy, when mbuild is adding a new source archive for the new bound_YYY_ object, it discovers there are three (or more) directories associated with each library; the sss library has sss.source, sss.object. sss.execution, sss.include, and sss.info directories. It doesn't know in which of those directories to install the new source archive.

Proposed Changes

Change mbuild_data_$get_BOUNDOBJ to use the operation field when checking existing BOUNDOBJ structures for a match. It must continue searching until both the name and operation of the BOUNDOBJ structure match those search specification parameters. Only when the entire list of existing BOUNDOBJ structures has no match should it then create a new BOUNDOBJ structure.

Change mbuild_data_$get_BOUNDOBJ and mbuild_data_$get_Seg to permit the input operation parameter to override the default REPLACE value used in a newly-created structure. Otherwise, a structure with ADD operation can never be created.

Change mbuild_analyze_ to always specify an operation when calling mbuild_data_$get_BOUNDOBJ. This ensures that all references to a bound object are analyzed based upon the operation specified in the build script file.

Change the mbuild_analyze_ internal az_source procedure to preserve any ADD operation specified for a Seg(source) which is a component of a Seg(source_arch) being added.

Change the mbuild_analyze_ internal az_source_arch procedure to modify the name of the object archive in which a bind file is added when installing a new bound segment consisting of two or more source archives. The Seg(bindfile).archive_name defaults to bound_XXX_.archive, but should be bound_XXX_.1.archive when the bound object is comprised of two or more source archives.

These changes are too lengthy to be described here in compare_ascii output.

These changes are made in the segments identified by the following build script:

```
Description:
 1) Change mbuild_data_ to fix Ticket 261: mbuild problems when splitting
    source archive into 2 smaller archives.
 2) Change mbuild to add hidden -debug (-db) control argument.
 3) Change mbuild_display_ to properly distinguish between Seg(Unbound_obj)
    and UNBOUNDOBJ(Unbound_obj) structures.
 4) Change mbuild_archive_ to correctly attach any existing Seg(Bound_obj)
    and Seg(listing) to their BOUNDOBJ structure when that structure  is
    created.  Otherwise, the clean operation won't know these derived
    segments exist and are eligible for clean-up.
 5) Change mbuild_set_ to permit setting <dir> portion of Seg.library
    for Seg structures.

Installation_directory:  >udd>m>gd>w>mb;

 Build_script:             mb.mb;
```

```
Bound_obj:              bound_mbuild_              IN: tools  UPDATE;
    source:             mbuild.pl1                  REPLACE;
    source:             mbuild_Tlist_.pl1           REPLACE;
    source:             mbuild_analyze_.pl1         REPLACE;
    source:             mbuild_archive_.pl1         REPLACE;
    source:             mbuild_clean_.pl1           REPLACE;
    source:             mbuild_compile_.pl1         REPLACE;
    source:             mbuild_data_.pl1            REPLACE;
    source:             mbuild_display_.pl1         REPLACE;
    source:             mbuild_print_.pl1           REPLACE;
    source:             mbuild_scan_.pl1            REPLACE;
    source:             mbuild_set_.pl1             REPLACE;
```

Testing

The two suggested remedies were tested using bound_info_rtns_ as the bound object.

For remedy one, components near the end of bound_info_rtns_.s.archive were moved to a bound_info_rtns_.2.s.archive; and bound_info_rtns_.s.archive was renamed to bound_info_rtns_.1.s.archive. The bound_info_rtns_.bind file was fetched. The sample build script in the Remedy 1 section above was then used to test all phases of an mbuild installation. All completed successfully through generation of a correct update_seg exec_com.

For remedy two, components comprising the help-related subsystems and commands were moved to a new bound_help_ bound segment. The bound_info_rtns_.bind file was fetched, then copied to bound_help_.bind; and descriptions for components no longer in the associated bound segment were edited out of each bind file. The same build script in Remedy 2 section above was then used to test all phases of an mbuild installation. All completed successfully up to generation of a correct update_seg exec_com.

Additional tests were run against more typical installations: installing these changes to bound_mbuild_ (as described by the build script immediately above); and extracting all mbuild components into an mbuild test directory, then using mbuild to recompile all these components with -optimize and -table control args. These tests shows that prior uses of mbuild were not impaired by the changes proposed in this MCR.

Documentation

No mbuild user interfaces are being changed, so no documentation changes are needed for this proposal.

Version History

Date	Revision	Author	Comment
2021-11-15	1.0	Gary Dixon	Initial draft of this MCR.
2021-11-24	1.1	Gary Dixon	Additional changes needed to properly build and install bound objects with overflowing source archives.
2021-11-26	1.2	Gary Dixon	Changes suggested by reviewers.