Subject: MCR10163, exec_com Can Fail to Release Temporary Segments

Author: Gary Dixon

Date: 2026-06-22

Multics Ticket: https://multics-trac.swenson.org/multics-trac/ticket/393

Introduction to the Problem

Ticket_393 describes a problem with the exec_com command not releasing temporary segments it acquired to run an exec_com program when execution of the program ends.[1] If an exec_com script includes one or more `&on-units`[2], if any of those on-units are activated by a signaled condition then a new exec_com runtime environment is created to run the LINES of the condition handler. This is the equivalent of a PL/I program pushing a new stack frame onto the user stack to run an on-unit activated by a signaled condition.

The new environment inherits access to variables and statement labels defined by the original exec_com that established the on-unit. But any faults, interrupts or signaled conditions that happen while running in the handler's runtime environment do not affect the original program's environment.

Lines of the original program are read using an I/O switch attached through the ec_input_ I/O module. An abs_data structure tracks all information pertaining to the original program's runtime environment. This includes attach and open information for the ec_input_ stream reading lines of the program. And abs_data also includes an area (called abs_data.work_area) in which the environment can make allocations as it interprets lines read from the exec_com script. This is an *extensible area*: if free space is not available when an allocation is attempted, the allocate operator defines a new area in a temporary segment stored in the user's process directory.

The list_temp_segments command can display ownership information about all temporary segments created in the process directory. The list_temp_segments output in the ticket shows many temp segments holding an area created by ec_input_, even when exec_com is no longer running. These unreleased areas were created as I developed an exec_com script, and invoked it many times to test its operation. This led to the hypothesis that areas used by the ec_input_ I/O module are not being released when exec_com ends execution of a script.

---

[1] The same problem can also occur when an absentee job ends. But it is less noticeable because the process running the absentee job gets destroyed when the job ends.

[2] An &on-unit is inserted into an exec_com script by a group of statements in the form:
&on CONDITION_LIST &begin LINES &end

Description of the Problem

The main area used by ec_input_ starts as a small area element declared in the abs_data structure. The area is only 800 words in size, but it is created as an extensible area by calling the define_area_ subroutine to reformat the area.

```
declare   1 abs_data              aligned based (abs_data_ptr),
            2 version             fixed bin,
            2 io_module_name      char (32) varying,
             ...
            2 work_area           area (800);
```

If the allocation operator finds insufficient free space in an extensible area, it adds a new component segment to the area. This component is linked to the original area by an extend_block allocated in the original area when that area gets defined by the define_area_ subroutine. This extend_block is located by a pointer in the component's area_header structure. The block contains a pointer to the most recently-added component of the extensible area. From area_structures.incl.pl1, the declaration of the extend_block is shown below.

```
dcl 1 extend_block aligned based (extend_blockp),
    2 first_area ptr unal,
    2 next_area ptr unal,
    2 sequence_no fixed bin,
    2 name char (32),
    2 pad fixed;
```

When a new component is added to the extensible area, an extend_block is also allocated in the new component area. Data from the original area's extend_block is copied to this new extend_block. Then the extend_block in the initial area component is updated to point to the new area component. In this manner, all the temporary components added to the extensible area are linked to the initial area's extend_block, and can be released when the area is no longer needed. Also, the component with the most free space is always chained immediately off the initial area component.

This method of linking components of an extensible area becomes important when diagnosing the failure to release all of the temporary segments composing the extensible area. When reading lines of a normal exec_com, ec_input_ allocations are made in the abs_data.work_area under complete control of the allocate operator, which calls upon the define_area_ subroutine when more free space is needed.

However, when ec_input_ reads a script that establishes on-units, each time one of those on-units gets signaled, the absentee_listen_$execute_handler routine gets called to establish a new runtime environment for executing lines of that on-unit. It allocates a new abs_data structure in the system_free_area, then calls abs_io_$initialize_abs_data to create the new runtime environment. This new abs_data structure provides links to the variables, labels and on-units in the original environment.

Initialization code for abs_data overwrites the work_area element of the new abs_data structure by copying the 800 words of the original environment's work_area into the new work_area. This permits the on-unit runtime to access items allocated by the original runtime; and also allows it to make its own allocations in an already operational extensible area. The following statement from abs_io_$initialize_abs_data shows this overwrite operation.

```
    abs_data.work_area = P_abs_data_ptr -> abs_data.work_area;
         /* use the same area; copy it back when done */
```

The comment attached to that code line provides an important insight about the failure to release the temp segments. The overwritten work_area still retains the complete allocation history of the extensible area, as controlled by the allocate operator and define_area_.

If the on-unit's runtime needs to allocate more storage than the free space in the existing area components, then a new component will get threaded onto extend_block in the abs_data.work_area component. So data in that area must be copied back into the original runtime's abs_data.work_area when execution of the on-unit completes. This copy-back preserves any changes made to the abs_data.work_area's extend_block (or other blocks in that area) when returning to the original runtime environment.

When execution of the on-unit LINES finishes, absentee_listen_ jumps to the EGRESS label which invokes a finish_up subroutine. This routine has the job of freeing the on-unit's abs_data structure, and restoring use of the original abs_data structure. According to the comment highlighted above, contents of the 800 word the abs_data.work_area is supposed to be copied back into the original abs_data.work_area.

But finish_up frees the new abs_data structure WITHOUT copying it back. Thus any changes to the extend_block made in the on-unit's abs_data.work_area are lost. This explains why an extension component of the work_area would never get released. Data about existence of some of these temporary area segments does not get preserved by the finish_up operation.

Code below is from absentee_listen_.pl1. This shows the steps involved in freeing the on-unit's runtime abs_data structure.

```
EGRESS:
   call finish_up ();

   return;
    ...

finish_up: procedure ();
 ...
 if initialized then
  if abs_data_ptr ^= null then
   if entry_point_name = "execute_handler" then do;

    P_continue_to_signal_sw = abs_data.on_info.continue_to_signal_sw;

    if abs_data.goto_sw then do;
     allocate goto_label in (saved_abs_data_ptr -> abs_data.work_area)
                              set (P_goto_label_ptr);
     P_goto_label_ptr -> goto_label =
                                 abs_data.goto_label_ptr -> goto_label;
     P_goto_label_len = abs_data.goto_label_len;

     saved_abs_data_ptr -> abs_data.goto_statement_pos =
                 abs_data.goto_statement_pos
               + charno (abs_data.input_string.ptr)
               - charno (saved_abs_data_ptr -> abs_data.input_string.ptr);
     saved_abs_data_ptr -> abs_data.goto_statement_len =
                                           abs_data.goto_statement_len;
     end;

    saved_abs_data_ptr -> abs_data.output_file = abs_data.output_file;
    saved_abs_data_ptr -> abs_data.variables_ptr = abs_data.variables_ptr;

    free abs_data;              /* free the &on unit's private abs_data */
                                /* and restore the parent's.            */

    abs_data_ptr, ec_info.switch_ptr -> attach_data_ptr =
                                                    saved_abs_data_ptr;
    end;

 return;

end finish_up;
```

Proposed Changes

```
Description --
1) Repair Ticket_393 in absentee_listen_.pl1


Changes --


 Build_script: ec.mb;

 Bound_obj:    bound_exec_com_                    IN: sss  UPDATE;
   source:     absentee_listen_.pl1                REPLACE  compiler: pl1 -ot;


mbuild:  compare

----------  absentee_listen_.pl1

compare_ascii >ldd>sss>source>bound_exec_com_.s.archive::absentee_listen_.pl1 ==
…

Inserted in B:
B31         5) change(2026-06-22,GDixon):
B32            Ticket 393:
B33             A) In finish_up, copy abs_data.work_area back to the starting abs_data
B34                structure so all extensible area components get released when the
B35                script ends.
Preceding:
A31                                                        END HISTORY COMMENTS */


A772                if initialized then
A773                  if abs_data_ptr ^= null then
A774                     if entry_point_name = "execute_handler" then do;
Changed by B to:
B777                if initialized then
B778                  if abs_data_ptr ^= null then
B779                     if entry_point_name = "execute_handler" then do;
B780
B781                          saved_abs_data_ptr -> abs_data.work_area =
                                                              abs_data.work_area;
B782                      /* copy work_area back to the starting abs_data to ensure */
B783                      /*   any changes made to its extend_block (which tracks   */
B784                      /*   all extensible area components) are visible in the   */
B785                      /*   starting abs_data.  This ensures all these area      */
B786                      /*   components are released when the exec_com or         */
B787                      /*   absentee job ends.       (Ticket 393)                */
```

Testing

The exec_com program which originally encountered this problem was used to verify correctness of the repair above. The list_temp_segments command was used before and after each test run to determine how many ec_input_ area segments were not released during the run.

A. Using the installed version of bound_exec_com_, 2 ec_input_ areas were listed by list_temp_segments after the exec_com ended. None existed prior to running the exec_com.

B. Using the repaired version of bound_exec_com_, 0 ec_input_ areas were listed by list_temp_segments both before and after the exec_com was run.

Documentation

No interfaces were changed by this repair, so no documentation needs to be updated.

Version History

Date	Revision	Author	Comment
2026-06-22	1.0	Gary Dixon	Initial version of this MCR.