

MCR10182

Fix Initializer Crash When `trace_stack` Is Used in Admin Mode (Ticket #69) Revision 1.0

Eric Swenson

October 10, 2026

1 Problem Description

Running `trace_stack` (`ts`) in admin mode crashes the system (ticket #69). The Initializer's ring-4 stack overflows while a `signal_io_` condition is being handled recursively, the handling of the overflow takes further "null pointer" faults, and the Initializer process is terminated. The same crash occurs with any other tool that uses `list_frame_args_`: `probe` (`probe_print_arglist_`) and `trace` (`display_argument_list_`).

1.1 How admin mode does I/O

In admin mode, and for commands run under system control, the `user_i/o` switch is attached to the `signal_io_` I/O module (`sc_process_command_line_`). `signal_io_` does no I/O itself. For each operation it fills in a `signal_io_info` structure and signals the `signal_io_` condition; a handler established by system control (`sc_admin_mode_`, `system_control_`, `call_ec_`, or `sc_admin_command_`'s own handler) performs the I/O on the real switch, logs it in the admin log, and returns. The handler is found by the normal condition mechanism, so it works only if no newer stack frame claims the condition first.

Programs that establish `any_other` handlers must therefore pass `signal_io_` on. `default_error_handler_` does so ("might be needed to do the work"), and so does `trace_stack_`'s `fault_handler` (`continue_conditions`, added in 1985).

1.2 The defect in `list_frame_args_`

`trace_stack_` calls `list_frame_args_` to print the arguments of each frame. Before its argument loop, `list_frame_args_` establishes an `any_other` handler with `condition_` ("`any_other`", `intproc`), so that a fault while referencing an argument is reported and the argument skipped. `intproc` passes on only `program_interrupt`, `finish` and `quit`, and ignores `cleanup` and quiet-restart conditions (`signal_io_` sets no action flags). For anything else it prints

```
call ioa_$ioa_stream (ss, "--ARG ^2d not accessible. - ^a", j, cname);  
go to skiparg;
```

In admin mode the sequence is:

1. The first `ioa_$ioa_stream` (`ss, "^-ARG ..."`) in the loop writes to `user_output`, which goes to `signal_io_`, which signals `signal_io_`.
2. The `list_frame_args_` frame is newer than the system control frame that would do the I/O, so `intproc` is invoked.
3. `intproc` calls `ioa_$ioa_stream` to report the “inaccessible” argument; that signals `signal_io_` again, and the handler in the same `list_frame_args_` frame is invoked again.
4. This repeats, adding frames each time, until the stack overflows.

The output that `list_frame_args_` produces before its argument loop (the argument list pointer, the command name, and the warnings) is printed correctly, because the handler has not yet been established; the crash comes at the first argument. Even without the recursion, the handler would have discarded the output: it claims the condition, so the I/O is never performed.

1.3 Admin mode is fragile

The underlying weakness is general: any program with an `any_other` handler that does output without first passing `signal_io_` on will recurse in the same way, and in admin mode that terminates the Initializer. Such handlers exist throughout the system and in site and user tools, and they cannot all be found and fixed.

2 Proposed Fix

Two independent changes. The first fixes the reported problem; the second keeps any similar handler from crashing the system.

2.1 `list_frame_args_`: pass `signal_io_` on

`intproc` sets `cont = "1"` for `signal_io_`, as it already does for `program_interrupt`, `finish` and `quit`:

```
/* signal_io_ is how the signal_io_ I/O module (user_i/o in admin mode and
   under system control) asks an older handler to do the I/O. It must be
   passed on: handling it here would discard the output, and printing the
   "not accessible" message would signal it again, recursing until the
   stack overflows. */

/* format: ^delnl */
if cname = "program_interrupt"
    | cname = "finish"
    | cname = "quit"
    | cname = "signal_io_"
then do;
    cont = "1";
    return;
```

```
end;
```

The condition then reaches whichever handler owns the I/O: in admin mode that is `sc_admin_mode_'s` `sc_signal_io_handler_`; under `sc_admin_command_` (the `x` and `admin` requests from the message coordinator) it is that procedure's `IO_SIGNAL_HANDLER`. Calling `sc_signal_io_handler_` from `list_frame_args_`, as suggested in the ticket, was rejected: it would tie a general library routine to system control and bypass `sc_admin_command_'s` handler.

Because `list_frame_args_` is shared, this also fixes `probe` and `trace` argument listings in admin mode.

2.2 signal_io_: limit the signalling depth

The internal procedure `signal`, through which every `signal_io_` operation signals the condition, keeps a per-process count of the `signal_io_` conditions currently being signalled. If the count has reached `MAX_SIGNAL_DEPTH` (20), it returns without signalling, leaving `returned_error_code` at its initial value, `error_table.$action_not_performed`, so the operation fails with that code. A `cleanup` handler decrements the count when a handler leaves with a nonlocal `go to` (as `intproc` does).

```
dcl    MAX_SIGNAL_DEPTH      fixed bin int static options (constant) init (20);
dcl    signal_depth         fixed bin int static init (0);
...
signal:
  procedure;

  declare counted           bit (1) aligned;

  if signal_depth >= MAX_SIGNAL_DEPTH
  then return;              /* returned_error_code is still
    action_not_performed */

  counted = "0"b;
  on cleanup
    begin;                  /* a handler may exit with a nonlocal go to */
    if counted
    then signal_depth = signal_depth - 1;
    end;

  signal_depth = signal_depth + 1;
  counted = "1"b;
  call signal_ ("signal_io_", null (), addr (sgii));
  counted = "0"b;
  signal_depth = signal_depth - 1;
  return;
end signal;
```

With this change alone, the old `list_frame_args_` no longer crashes the system: each argument's output recurses 20 levels, the innermost `ioa_` call fails, `intproc` goes to `skiparg`, and the listing continues without that argument's line. Output is lost, but the Initializer survives and the operator

can see that something is wrong.

2.2.1 Choice of limit

Legitimate nesting is one level per `signal_io_` operation that is still pending further down the stack. In practice that happens when a quit is taken while admin mode is waiting for input: the interrupted `get_line` stays on the stack and the new command level's `get_line` is one level deeper. The I/O performed by the system control handlers goes to the real switch and to the admin log, not through `signal_io_`, so it does not nest.

Each cycle of a recursion like the one in ticket #69 uses on the order of a thousand words of stack (`signal_io_`, `signal_`, the handler, `ioa_` and `iox_` frames). A limit of 20 stops the recursion well before the stack is exhausted while allowing 19 nested quit levels in admin mode, far more than are ever used.

The count is per process, not per switch. A handler that does output on a different `signal_io_` switch while handling one is still counted; that is the conservative choice, and such nesting is shallow.

2.2.2 Behavior past the limit

Past the limit every `signal_io_` operation fails with `action_not_performed` until the nesting unwinds. If an operator really had 20 pending quit levels in admin mode, the listener's `get_line` would fail; the remedy is to release quit levels, which reduces the depth. This is far better than the present behavior, where runaway recursion terminates the Initializer.

3 Documentation

No documentation changes. The `signal_io_` I/O module is internal to system control and has no info segment.

4 Test

To be done on a test system (GHM-TEST), not in production.

1. **Reproduce.** With the current software, enter admin mode and run `ts`. Expected: the Initializer crashes (confirms the test system shows the problem).
2. **signal_io_ alone.** Install the new `signal_io_` with the old `list_frame_args_`. In admin mode run `ts`. Expected: no crash; the trace is printed with the argument lines missing. Then `ame` and confirm the system is healthy.
3. **Both changes.** Install the new `list_frame_args_`. In admin mode run `ts`: the full trace, including every frame's arguments, is printed on the console and appears in the admin log.
4. **probe and trace.** In admin mode, `probe` a frame with arguments and use its argument-listing request; use `trace` on a procedure and call it. The argument listings are printed.
5. **Normal use.** In admin mode run ordinary commands, an `exec.com`, and an interactive program, and take one or two quits and `pi/` `release` them. Run `x` commands from the message coordinator. All behave as before.

6. **Outside admin mode.** In a user process run **ts**, **probe** and **trace**: unchanged.

5 Files Changed

bound_trace_stack..s.archive	list_frame_args..pl1: intproc passes signal_io_ on
bound_trace_stack..archive	list_frame_args_ (recompiled)
bound_misc_io_modules..s.archive	signal_io..pl1: signalling depth limit
bound_misc_io_modules..archive	signal_io_ (recompiled)
>sss	bound_trace_stack_, bound_misc_io_modules_ (rebound)

A new Initializer process (a reboot) is needed for **signal_io_** to take effect in system control; **list_frame_args_** takes effect in any process that has not yet used it.

6 Version History

2026-10-10	1.0	Initial version of the MCR.
------------	-----	-----------------------------