MCR10123
Fix Issue with `meter_signal` When Invoked with `-nfaults 2`
Revision 1.1

Eric Swenson

September 4, 2022

1 Problem Description

Ticket https://multics-trac.swenson.org/ticket/123 describes a problem with the `meter_signal` command, where it gets an `illegal_procedure` condition when invoked with the `-nfaults N` control argument, where N is greater than 1.

2 Analysis

If you invoke the `meter_signal` command with the `-nfaults 1` control argument, you get output like this:

```
meter_signal -nfaults 1
The following environment will be established:

1 stack frames will be laid down.
0 dummy interrupt handlers will be established in each frame.
1 zerodivide faults will be signalled.


Following are the times in microseconds for each fault:


      419

Minimum value = 419 Mean = 419
```

However, if you invoke it with a `-nfaults N` control argument, where N is 2 or greater, you get output similar to this:

```
meter_signal -nfaults 2
The following environment will be established:
```

```
1 stack frames will be laid down.
0 dummy interrupt handlers will be established in each frame.
2 zerodivide faults will be signalled.


Following are the times in microseconds for each fault:


      436
Error: illegal_procedure condition by meter_signal$|1461
(>system_library_tools>bound_system_test_)
referencing stack_4|4010 (in process dir)
```

The `illegal_procedure` condition is occurring in `meter_signal` at the DTB instruction shown below:

```
             q = divide (1, 0, 17, 0);

001455 aa  100 004 227 404 dv3d     (ic),(ic),(pr)
001456 aa   000322 01 0002 desc9ls  210,2,0           001777 = 053060000000
001457 aa   000323 01 0002 desc9ls  211,2,0           002000 = 053061000000
001460 aa  6 00156 01 0022 desc9ls  pr6|110,18,0
001461 aa  000 100 305 500 dtb      (pr),(pr)
001462 aa  6 00156 01 0022 desc9ls  pr6|110,18,0
001463 aa  6 00123 00 0004 desc9a   pr6|83,4          q
```

The program intentionally divides by zero to generate the signal to be measured. The program's condition handler collects the timing data and if more faults are to be generated, returns from the condition handler.

The reason for the `illegal_procedure` condition is that the `on` unit that handles the `zerodivide` condition returns, rather than performing a non-local transfer of control. The PL/I Language Specification (manual AG94) page 10-11 asserts that a program is in error if its `zerodivide` on-unit returns to the point of execution. The correct alternatives are to do a non-local goto to resume execution at a different point in the program, or to call the `continue_to_signal_` subroutine to tell the signaler to look for another handler for the condition.

In the specific case with `meter_signal`, the on unit does not perform a non-local transfer, and thus the processor returns to the instruction following the DV3D instruction – the DTB instruction. However, execution of that instruction gets an `illegal_procedure` fault because it illegal to return to the instruction following the faulting instruction in this case.

The fix is quite simple – after the last statement in the `zerodivide` `on` unit, add a `goto` to a label that is inserted after the `divide` builtin invocation. The same fix is required for the `unclaimed_signal` handler, which also must not return.

This, in turn, corrects the issue:

```
meter_signal -nfaults 2
The following environment will be established:
```

```
1 stack frames will be laid down.
0 dummy interrupt handlers will be established in each frame.
2 zerodivide faults will be signalled.


Following are the times in microseconds for each fault:


      424      403

Minimum value = 403 Mean = 413
```

3 Proposed Fix

The proposed fix is to add labels after the calls to the `divide` builtin, and change the `on` units for the `zerodivide` and `unclaimed_signal` handlers to end with a `goto` to the appropriate labels. In addition, since the `unclaimed_signal` condition was changed to the `any_other` condition many years, ago, the all references to `unclaimed_signal` will be changed to `any_other`. Finally, there is a label used to loop over the invocation of the `divide` builtin invocations. Even though there is nothing really wrong with the fact that this label is used twice, keeping in mind the rules of PL/1 scoping, the second such loop will have its label change from `div_loop` to `div_loop2` for clarity.

Because the indentation in the original source code was incorrect, leading a difficulty in reading the original code, the indentation around the `on` unit establishment is updated as well. Because of these indentation changes, the `compare_ascii` output below looks a little daunting, but a careful reader will see that the changes are simply those described above.

Here is the `compare_ascii` comparison of the old and new version of the code:

```
Inserted in B:
B11
B12      /****^ HISTORY COMMENTS:
B13        1) change(2022-09-04,Swenson), approve(2022-09-04,MCR10123):
B14           Fix improper on unit definitions and consequent illegal_procedure faults
B15           that occur when meter_signal is invoked with -nfaults 2.
B16                                                  END HISTORY COMMENTS */
B17
Preceding:
A11      /* Originally coded by Paul Karger August 17, 1971 */


A24          (unclaimed_signal, zerodivide) condition,
Changed by B to:
B31           (any_other, zerodivide) condition,


A36           nuncl fixed bin init (0),                        /* frame number for the
    unclaimed_signal handler */
Changed by B to:
```

```
B43          nuncl fixed bin init (0),                          /* frame number for the
    any_other handler */


A126              if unclaimed then call ioa_ ("An unclaimed signal handler will be in
    stack frame ^d.", nuncl);
Changed by B to:
B133              if unclaimed then call ioa_ ("An any_other signal handler will be in
    stack frame ^d.", nuncl);


A131                  on zerodivide
A132                  begin;                                     /* set up zerodivide
    handler */
A133                  newtime = clock_ ();                       /* read the clock */
A134                  diff = newtime - time;                     /* get the difference */
A135                  call ioa_$nnl ("^10d", diff);              /* print it out */
A136                  j = j + diff;                              /* accumulate the sum */
A137                  l = min (l, diff);                         /* get the minimum fault
    time */
A138                  i = i + 1;
A139                  if mod (i, 4) = 0 then call ios_$write_ptr (p, 0, 1); /* put out
    newline every four */
A140                  if i >= nfaults then go to all_done;
A141                                                             /* check fault counter */
A142              end;                                           /* return to fault to
    permit resignalling */
Changed by B to:
B138                  on zerodivide begin;                                /* set up
    zerodivide handler */
B139                      newtime = clock_ ();                            /* read the clock
    */
B140                      diff = newtime - time;                          /* get the
    difference */
B141                      call ioa_$nnl ("^10d", diff);                   /* print it out */
B142                      j = j + diff;                                   /* accumulate the
    sum */
B143                      l = min (l, diff);                              /* get the minimum
    fault time */
B144                      i = i + 1;
B145                      if mod (i, 4) = 0 then call ios_$write_ptr (p, 0, 1); /* put
    out newline every four */
B146                      if i >= nfaults then go to all_done;            /* check fault
    counter */
B147                      goto AFTER_DIVIDE;
B148                  end;                                            /* return to fault
    to permit resignalling */


A146                      on condition (unclaimed_signal)
A147                      begin;
A148
```

```
A149                    newtime = clock_ ();
A150                    diff = newtime - time;
A151                    call ioa_$nnl ("^10d", diff);
A152                    j = j + diff;
A153                    l = min (l, diff);
A154                    if mod (i, 4) = 0 then call ios_$write_ptr (p, 0, 1);
A155                    i = i + 1;
A156                    if i >= nfaults then go to all_done;
A157                end;
Changed by B to:
B152                        on any_other begin;
B153                            newtime = clock_ ();
B154                            diff = newtime - time;
B155                            call ioa_$nnl ("^10d", diff);
B156                            j = j + diff;
B157                            l = min (l, diff);
B158                            if mod (i, 4) = 0 then call ios_$write_ptr (p, 0, 1);
B159                            i = i + 1;
B160                            if i >= nfaults then go to all_done;
B161                            goto AFTER_DIVIDE;
B162                        end;


Inserted in B:
B172    AFTER_DIVIDE:
Preceding:
A167                    go to div_loop;                         /* loop back - the handler
    will turn it off at the right time
\c */


A179                            on condition (unclaimed_signal)
A180                            begin;
A181
A182                        newtime = clock_ ();
A183                        diff = newtime - time;
A184                        call ioa_$nnl ("^10d", diff);
A185                        j = j + diff;
A186                        l = min (l, diff);
A187                        if mod (i, 4) = 0 then call ios_$write_ptr (p, 0, 1);
A188                        i = i + 1;
A189                        if i >= nfaults then go to all_done;
A190                    end;
Changed by B to:
B185                            on any_other begin;
B186                                newtime = clock_ ();
B187                                diff = newtime - time;
B188                                call ioa_$nnl ("^10d", diff);
B189                                j = j + diff;
B190                                l = min (l, diff);
B191                                if mod (i, 4) = 0 then call ios_$write_ptr (p, 0, 1);
B192                                i = i + 1;
```

```
B193                                if i >= nfaults then go to all_done;
B194                                goto AFTER_DIVIDE2;
B195                            end;


A197     div_loop:
Changed by B to:
B202     div_loop2:


A200                        go to div_loop;
Changed by B to:
B205     AFTER_DIVIDE2:
B206                        go to div_loop2;

Comparison finished: 10 differences, 88 lines.
```

---

4 Testing

Invoking this little-used program multiple times with multiple control arguments shows that it no longer issues the `illegal_procedure` condition.

5 Documentation

No changes to the documentation are required.

6 Version History

2022-09-03	1.0	Initial version of the MCR.
2022-09-04	1.1	Updated with the correct fix to the problem.