MCR10097
Fix Fixed Point Overflow in disk_usage_stat
Revision 1.1

Eric Swenson

October 26, 2021

1 Problem Description

The system administrator tool, `disk_usage_stat` is used by the `dds.absin` absentee job to compute statistics about the segments and directories in the storage system. When run on a stock MR12.7 release distribution (as well as those of MR12.5 and MR12.6), it gets a fixed point overflow when executed, due to the use of a fixed-size array whose bound is exceeded. When the program writes beyond the end of the array, it corrupts various stack variables used in calculations, leading to the fixed point overflow.

Specifically, there are two automatic arrays of integers:

```
dcl  n_ents_w_i_names (0: 2048) fixed bin;
dcl  n_dirs_w_i_names (0: 2048) fixed bin;
```

that are declared with fixed extents. The first, `n_ents_w_i_names` is used to keep track of the number of entries with a specific number of names on the entry. The second, `n_dirs_w_i_names` is used to keep track of the number of directories with a specific number of names in the directory.

The size of both arrays is fixed at 2048. When run on an MR12.7 cold-booted hierarchy, the maximum number of names found in a directory is 2823, which exceeds the 2048 array entry stack allocation. The directory containing that number of names is `>doc>info_segments`. The maximum number of names on an entry is, in the same test, only 161, so the `n_ents_w_i_names` array updating does not overwrite the stack. On the GHM Multics system, the maximum number of names on an entry is 259, still way below the limit of that array.

The program, as currently written overwrites the automatic variables that follow the `n_dirs_w_i_names` array, causing the fixed point overflow later on.

2 Proposed Fix

The proposed fix is to increase the size of the `n_dirs_w_i_names` array from 2048 to 4096, or double its current value. This is more than enough to prevent stack overwriting. In addition, the `subscriptrange` condition prefix will be added to the main `disk_usage_stat` procedure, so that

if any array write exceeds its bounds, the `subscriptrange` condition will be signalled, rather than overwriting the stack. If this condition is signalled, the caller can use a debugger to determine which array bound should be increased.

In addition, this MCR proposes to add a comment to the start of the program that describes the array limits and use of the `subscriptrange` condition, and what a caller should do if it is signalled.

A `compare_ascii` comparison of the current and proposed changes is below:

```
cpa [lpn disk_usage_stat.pl1] ==

A10
Changed by B to:
B10
B11      /*
B12       * The subscriptrange condition prefix is applied to the whole procedure because
    there are many array
B13       * references to multiple arrays whose indexes might exceed the declared array
    bounds. Rather than allow
B14       * the program to inadvertently overwrite the stack, this condition prefix will
    signal a subscriptrange
B15       * condition, allowing the user to investigate the issue (using probe, for
    example) and to increase the
B16       * array bounds.
B17       *
B18       * One array in particular, n_dirs_w_i_names, currently has an array bound of
    4096, allowing for directories
B19       * containing up to 4096 names. This is more than sufficient to handle most
    currently known storage system
B20       * hierarchies (including that of MR12.7). But should directories arise with more
    than 4096 names, the
B21       * subscriptrange will prevent stack damage.
B22       *
B23       * Should this condition be signalled, the appropriate array should have its
    bounds increased.
B24      */
B25
B26      (subscriptrange):


A55           n_dirs_w_i_names (0: 2048) fixed bin,
Changed by B to:
B71           n_dirs_w_i_names (0: 4096) fixed bin,

Comparison finished: 2 differences, 20 lines.
r 14:03 1.202 19
```

3 Alternatives Explored

Two alternatives were explored and ultimately rejected.

3.1 Dynamic Reallocation

This alternative involved increasing the array bounds for `n_dirs_w_i_names` and detecting when this bound would be exceeded. In that case, the array would be reallocated with a size double its current allocation, the old array copied to to the new array, and the old array freed. The array would move from being an automatic variable to a based variable, allocated in the system free area.

This approach was rejected due to being considered by reviewers as overkill, given the insignificance of the `disk_usage_stat` tool, and the alternative to simply increase the array size well beyond projected needs.

3.2 Increased Size Static Allocation with Control Arguments

This alternative called for increasing the size of the array bounds, and checking for exceeding those bounds, issuing a fatal error if that happens suggesting the user reinvoke the command with a control argument that specifies an increased bound. The initial size allocated to the array would be increased from its current size, but overridable via control argument.

This approach was rejected because it seemed cumbersome to use and difficult for the caller to know what size to specify, and because dynamically reallocating the array would obviate the need for the control arguments.

4 Testing

The fix will be tested on a stock, cold-booted MR12.7 system, as well as on GHM, where the the problem was detected. With the default settings, it should produce no errors and run successfully.

5 Documentation

There are no documentation changes needed, other than the above-mentioned comment to be added to the code.

6 Version History

2021-10-25	1.0	Initial version of the MCR.
2021-10-26	1.0	New proposal after review comments..