MCR10143
Fix Issue with instr_speed When Run Under Simulator on Slow Hosts
Revision 1.2

Jeff Johnson

February 11, 2025

1 Problem Description

The `instr_speed` test program runs computes the CPU speed by running timing tests on a set of CPU instructions. Currently, it attempts to collect 100 samples for each instruction. However, when it times a given instruction to collect a sample, it handles the case where the sample is considered "too long", by skipping the sample. If all samples for a given instruction are considered "too long", then the program loops indefinitely – never able to collect 100 good samples.

Samples will be considered "too long" on a system (hardware or simulated) that is very slow. Such a situation can result if Multics is being run under the DPS8M simulator on a very slow host. An example of such a host system is a Raspberry Pi 2. On such a system, `instr_speed` will never terminate.

In the past, we fixed `instr_speed` to properly display large values such as could be achieved on very fast systems. This MCR proposes a fix for the opposite problem, that is, allows `instr_speed` to run if the system is too slow.

The proposed change skips a number of "too long" samples, but after a threshold, records those samples until 100 samples, albeit large, are collected.

The ticket describing the original problem as reported is located here: https://multics-trac.swenson.org/ticket/347.

2 Proposed Fix

In the proposed fix, we throw away the first 50 samples that are considered "too slow". After that, we count them, regardless of whether the helper ALM procedure, `test_speed` consideres them slow.

The value fifty is proposed because we determined that the most commonly failing test was expecting a value of approximately 2400 nanoseconds to complete. On the slowest hosts worth using DPS8M on, one can see values like 6000 ns. So, repeating it 50 times before accepting the value gives a value

of 0.3ms. These tests run 100 times, so, if it fails (in that every sample is a large sample, every time) that gives you a test duration of about 30ms.

On a fast host, the test completes in about 0.24ms. This appears "instantly fast" to someone watching/running the command.

I did play with "enhancing" the accuracy by scaling the histogram size, the number of iterations, etc by 300%. The results were slightly more accurate, but probably not enough to update the code.

Even so, on a faster host, `instr_speed` completes in 0.3 seconds or so, wall clock time.

Making it take more samples means it runs in about 1 second. But on a very slow host, the whole process takes about 3 seconds to complete, so having it take 9 seconds to run makes it seem *slow*.

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

Inserted in B:
B24        3) change(2025-02-10,Swenson), approve(2025-02-10,MCR10143):
B25           Updated to work on very slow systems (such as on DPS8M simulator on slow
B26           host system).
Preceding:
A24                                                    END HISTORY COMMENTS */


A135                       else if ls > 0 then ls_aborts = ls_aborts + 1; /* took a large
    sample (probable interrupt) */
A136                       else do;
A137                            successes = successes + 1;      /* another successful run */
A138                            time_total = time_total + float (time);
A139                            mips = float (count) / float (time); /* get mips for this
    run */
A140                            bucket = mips * INDEX_FACTOR;           /* get the index
    into hist for this run */
A141                            bucket = min (bucket, MAXHIST); /* watch out for overflow */
A142                            bucketmax = max (bucketmax, bucket); /* calculate bounds of
    possible values for this type */
A143                            bucketmin = min (bucketmin, bucket); /* .. */
A144                            hist (bucket) = hist (bucket) + 1; /* fill in histogram */
A145                            mips_total = mips_total + mips; /* keep running total for
    final ave */
A146                       end;
Changed by B to:
B138                       else if ls > 0 then do;             /* took a large sample
    (probable interrupt) */
B139                           if ls_aborts < 50 then ls_aborts = ls_aborts + 1; /*
    increment ls_aborts up to 50 */
B140                           else
B141                                call process_success();    /* continue if ls_aborts >=
    50 */
B142                       end;
B143                       else
```

```
B144                            call process_success();        /* handle successful run if
    no page fault or large sample */


A169
A170          end;
Changed by B to:
B167     process_success: procedure();
B168                   successes = successes + 1;      /* another successful run */
B169                   time_total = time_total + float (time);
B170                   mips = float (count) / float (time); /* get mips for this run */
B171                   bucket = mips * INDEX_FACTOR;           /* get the index into hist
    for this run */
B172                   bucket = min (bucket, MAXHIST); /* watch out for overflow */
B173                   bucketmax = max (bucketmax, bucket); /* calculate bounds of possible
    values for this type */
B174                   bucketmin = min (bucketmin, bucket); /* .. */
B175                   hist (bucket) = hist (bucket) + 1; /* fill in histogram */
B176                   mips_total = mips_total + mips; /* keep running total for final ave
    */
B177                   return;
B178     end process_success;
B179
B180     end instr_speed;

Comparison finished: 3 differences, 38 lines.
```

3 Documentation

No documentation changes are needed since this is an internal implementation detail in a test program.

4 Test

The changes will be tested by running this updated `instr_speed` on various Multics system and ensuring correct results.

5 Version History

2025-02-02	1.0	Initial version of the MCR.
2025-02-03	1.1	Added some explanation as to how the value of 50 was arrived at.
2025-02-11	1.2	Updated code to adhere to Multics coding standards.