Is there any meaningful way to compare two Time Travel Debugging positions?

devblogs.microsoft.com/oldnewthing/20220905-00

September 5, 2022

Raymond Chen

Time Travel Debugging uses position objects to represents points in the recorded debugging trace. A position object consists of two numbers (represented in hexadecimal) separated by a colon. The first number is a 64-bit number known as the *sequence number*, and the second number is a 32-bit number known as the *step number*.

So if you have two position objects, is there a way to determine which one occurred first in the trace?

Generally speaking, the position objects can be sorted chronologically by sequence number, and then by step within a sequence number.[1] This ordering is valid even across traces from different processes running simultaneously on the same system. (The consistent cross-process ordering is handy when you are debugging a problem that spans multiple processes.)

Roughly speaking, what you can say is that the *first* step (step zero) of each sequence is chronological by sequence number. The remaining steps in the sequence proceed chronologically within the sequence, but chronological ordering between steps from different sequences is a bit fuzzier because multiprocessing is like that.

For example, consider this timeline:

Thread 1	3:0	3:1		3:2		6:0	6:1

Thread 2	4:0	4:1	4:2	4:3	5:0	5:1

At the start of the trace, thread 1 is assigned sequence number 3, and thread 2 is assigned sequence number 4. Thread 1 then executes instructions 3:0, 3:1, and 3:2. Meanwhile, thread 2 executes instructions 4:0, 4:1, and 4:2. Instructions do not all take the same amount of

time to execute, so you can't compare step numbers between sequences. In the above example, thread 1 executed some slow instructions, so it is stuck on step 2 (within sequence 3). while thread 2 is already on step 3 (within sequence 4).

In the above trace, thread 2 concluded sequence 4 after four instructions, and its next sequence was assigned the number 5. A little while later, thread 1 concluded sequence 3 and it received sequence number 6 for its next sequence of instructions.

So how does the Time Travel Debugger decide when to end a sequence and start a new one?

Sequence boundaries occur at these points:

- The thread issues a synchronizing instruction, such as an interlocked memory access or a memory barrier.
- The thread transitions to kernel mode, say, because of a system call.
- At the Time Travel Debugger's discretion. (Say, because the sequence is getting kind of long.)

Don't be surprised when you see the sequence number jump forward a lot when you step over a system call. After all, that system call could have taken a long time, and other threads got to run thousands of sequences while the thread you're stepping through was stuck in the system call.

Bonus chatter: The Time Travel Debugging folks tell me that the realities of multiprocessing can result in apparent contradictions in the Time Travel Debugging trace. For example, if two threads access the same memory without synchronization, they may observe values that appear to be contradictory with respect to each other. You might observe one thread write a value to memory, and then another thread reads from that memory and gets the old value! This problem can be exacerbated by false sharing.

[1] This is what we did some time ago when we studied the case of the mysterious over-release from deep inside the marshaling infrastructure.

Raymond Chen