How do Time Travel Debuggers Work?

Greg Law
Most programmers spend most of their time debugging.
Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

Brian Kernighan
THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY printf's COMPILING."

Hey! Get back to work!

Oh. Carry on.

VS

STEP BY STEP DEBUGGING

Many steps later

I am going to kill you!

MONKEYUSER.COM
How did that happen?
Expectations

Reality
What makes bugs hard to fix?

Non-deterministic

Heisenbugs’

Time between bug and failure
What happened… in detail

See what *happened*

- Printf logging

Time travel

Interrogate program without recompiling

Debuggers
C++ projects and products

**Linux**
- Undo - UDB & LiveRecorder
- rr (rr-project.org)
- GDB (ish)

**Windows**
- TTD

**Embedded**
- Lauterbach “TRACE32”
- Green Hills TimeMachine
### Non C++

- JavaScript / React: replay.io
- Java: Undo
- Rust, Go: Undo / rr
- Python: Undo*, PyPy*
Design Decision 1.
How to remember
What was the previous state?

Two options:

1. Save it.
2. Recompute it.

$$a = a + 1$$

$$a = b$$
Deterministic re-execution

\[ f(x) \rightarrow x' \]
Deterministic re-execution

- Snapshots.

- Event log.
Non-determinism

- What is unpredictable?
  - System calls.
  - Thread switches.
  - Asynchronous events (signals).
  - Shared memory accesses.
  - Some machine instructions.
How the technology works

Non-deterministic code

Here’s some machine code in a program that we want to be able to record and replay.

\[
\begin{align*}
0x000055555555515e & \text{ mov } -0x4(\%rbp),\%eax \\
0x0000555555555161 & \text{ mov } \%eax,\%esi \\
0x0000555555555163 & \text{ lea } 0xe9a(\%rip),\%rdi \\
0x000055555555516a & \text{ mov } $0x0,\%eax \\
0x000055555555516f & \text{ sub } \%rdx,\%rsp \\
0x0000555555555171 & \text{ mov } \%rcx,\%r10 \\
0x0000555555555174 & \text{ mov } $0x11,\%eax \\
0x0000555555555179 & \text{ syscall }
\end{align*}
\]

deterministic

nondeterministic
Design Decision 1

Re-execution is the winner
Design decision 2: what exactly to record?

- A process?
- A thread or function?
- A sandbox environment (e.g. JVM or v8?)
- Virtual machine?
Recording at process/OS ABI boundary

- **Process 1**
  - libc
  - libm

- **Process 2**

- **Process 3**

- **Kernel**
Design Decision 2 - winner!

Re-execution at the process boundary is the winner
Design Decision 3: How to track time?
Design Decision 3: How to track time?

To JIT or not to JIT?
How the technology works

Translation and events

Program

A-code

Binary Translation

B-code

Instrumentation Cache

CPU

Engine Library

Undo Instrumentation Engine

Event Log

System calls

Kernel

Record time
How the technology works

Translation and events

Program

A-code

Binary Translation

B-code

Instrumentation Cache

CPU

Engine Library

Undo Instrumentation Engine

Event Log

Replay time
Perf counters

- Can be very fast.
- Simpler (less code).

JIT

- Works anywhere
- With anything*
- Solid
Design Decision 4: Rely on memory determinism?
Memory Determinism

- (Much) smaller event logs
- Faster single-threaded

Not

- Unknown syscalls
- Parallel record
Thread serialization

Multithreaded code

write()

read()
Non-determinism

- What is unpredictable?
  - System calls.
  - Thread switches.
  - Asynchronous events (signals).
  - Shared memory accesses.
  - Some machine instructions.
Design decisions

- At what boundary to capture
- Binary rewriting instrumentation
- All/some/no memory recording
- Separate record/replay phases
- Parallel thread recording

<table>
<thead>
<tr>
<th>Undo</th>
<th>rr</th>
<th>WinDbg</th>
<th>replay.io</th>
<th>ODB</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc</td>
<td>proc</td>
<td>proc</td>
<td>proc</td>
<td>JVM</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>some</td>
<td>none</td>
<td>all*</td>
<td>none</td>
<td>all</td>
</tr>
<tr>
<td>yes/no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Debuginfo

(gdb) print foo
<value optimized out>
(gdb)