Taming Nondeterminism

Robert O'Callahan
Mozilla Corporation
### Nondeterminism at scale

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linux opt</strong></td>
<td>B Cpp Jit1 Jit2 Mn Mn-e10s Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R1 R2 Ru) R-e10s(C R-e10s) T(c d g1 g2 o s tp) W(1 2 3 4)</td>
</tr>
<tr>
<td><strong>Linux pgo</strong></td>
<td>B Cpp Jit1 Jit2 Mn Mn-e10s Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R1 R2 Ru) R-e10s(C R-e10s) T(c d g1 g2 o s tp) W(1 2 3 4)</td>
</tr>
<tr>
<td><strong>Linux debug</strong></td>
<td>B Cpp Jit1 Jit2 Mn X M(1 2 3 4 5 JP bc1 bc2 bc3 dt1* dt2 dt3 dt4 gl oth* p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R1 R2) R-e10s(C R-e10s1 R-e10s2)</td>
</tr>
<tr>
<td><strong>Linux x64 opt</strong></td>
<td>B Cpp H Jit1 Jit2 Ld Mn V Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R) R-e10s(C R-e10s) T(c d g1 g2 o s tp) W(1 2 3 4)</td>
</tr>
<tr>
<td><strong>Linux x64 pgo</strong></td>
<td>B Cpp Jit1 Jit2 Ld Mn Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3 dt) R(C J R) R-e10s(C R-e10s) T(c d g1 g2 o s tp) W(1 2 3 4)</td>
</tr>
<tr>
<td><strong>Linux x64 asan</strong></td>
<td>Bd Bo Cpp Jit1 Jit2 M(1 2 3 4 5 JP bc1 bc2 bc3 dt* gl oth p) M-e10s(1 2* 2 3 4 5 bc1 bc2 bc3) R(C J R*)</td>
</tr>
<tr>
<td><strong>Linux x64 debug</strong></td>
<td>B Cpp Jit1 Jit2 Mn S X M(1 2 3 4 5 JP bc1 bc2 bc3 dt1 dt2 dt3 dt4 gl oth p) M-e10s(1 2 3 4 5 bc1 bc2 bc3) R(C J R1 R2) R-e10s(R-e10s1 R-e10s2)</td>
</tr>
</tbody>
</table>
Deterministic hardware
Sources of nondeterminism
Record inputs
Replay execution
“Old idea”

- Nirvana
- Chronomancer
- Scribe
- CLAP
- ReTrace
- PinPlay
- Jockey
- PANDA
- FlashBack
- ReVirt
- ReSpec
- ODR
- Echo
- QuickRec
rr goals

- Easy to deploy: stock hardware, OS
- Low overhead
- Works on Firefox
- Small investment
**rr design**

- System call results
- Signals

**Deterministic user-space CPU execution**

**Observable effects**

**Record and replay**
No code instrumentation

mov EDX, tls.ebp
mov ECX, tls
call MemReadCallback
mov EAX, [EDX]
### Use modern HW/OS features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>System call results</td>
<td>ptrace</td>
</tr>
<tr>
<td>Signals</td>
<td>ptrace</td>
</tr>
<tr>
<td>Shared memory data races</td>
<td>Limit to single core</td>
</tr>
<tr>
<td>Asynchronous event timing</td>
<td>HW performance counters</td>
</tr>
<tr>
<td>Trap on a subset of system calls</td>
<td>seccomp-bpf</td>
</tr>
<tr>
<td>Notification when system call blocks in the kernel</td>
<td>DESCHED perf events</td>
</tr>
</tbody>
</table>
Event timing: HW perf counters

- `alarm()`
- `SIGALRM`

- Measure progress
- Instructions executed!
- Retired conditional branches (Intel)
- Zero overhead
Use modern HW/OS features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>System call results</td>
<td>ptrace</td>
</tr>
<tr>
<td>Signals</td>
<td>ptrace</td>
</tr>
<tr>
<td>Shared memory data races</td>
<td>Limit to single core</td>
</tr>
<tr>
<td>Asynchronous event timing</td>
<td>HW performance counters</td>
</tr>
<tr>
<td>Trap on a subset of system calls</td>
<td>seccomp-bpf</td>
</tr>
<tr>
<td>Notification when system call blocks in the kernel</td>
<td>DESCHED perf events</td>
</tr>
</tbody>
</table>
Accelerating system calls

Before syscall:
- Kernel read()

After syscall:
- ... record results ...

Before system call function rr:
- before_syscall
Avoid context switches

librrpreload.so
shim_read()

Kernel read()

... record results ...

Suppress ptrace trap
Use seccomp-bpf predicates
Reverse execution

(gdb) watch -l mRect.width
(gdb) reverse-continue
nsIFrame::SetRect
(this=0x2aaadd7dbeb0, aRect=...)
718       mRect = aRect;

(gdb) reverse-next
Results

21:38 mstange roc: there's somewhat of a competition going on here at the office about who can use rr the most
21:38 mstange roc: it's so good
21:39 roc :-)
21:39 roc who's using it?
21:39 mstange roc: jeff, myself, jeff's interns
21:40 mstange roc: and we're telling everybody else to use it whenever we get the chance
This is only the beginning
Parallel dynamic analysis
Omniscient debugging

<table>
<thead>
<tr>
<th>Time</th>
<th>Address</th>
<th>Written value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0x40030</td>
<td>127</td>
</tr>
<tr>
<td>12</td>
<td>0x40034</td>
<td>512</td>
</tr>
<tr>
<td>15</td>
<td>0x40030</td>
<td>221</td>
</tr>
<tr>
<td>18</td>
<td>0x7FFFFFF0</td>
<td>0</td>
</tr>
</tbody>
</table>

“At time 14, what was the most recent write to 0x40030?”
Record everything

Diagnose every bug

Fully analyze every security incident

Recover all lost data

A “black box” for every computer!
We can do it

- Hardware support for race detection
- Secure persistent storage
- Hypervisor support for recording
- Privacy issues!