Practical Record And Replay Debugging With rr

Robert O'Callahan
Debugging nondeterminism

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command Line</th>
<th>Parameters</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux opt</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)</td>
<td>R-e10s(1 2 3 4 5 bc1 bc2 bc3 dt)</td>
<td>R-e10s(C R-e10s) T(c d g1 g2 o s t p) W(1 2 3 4)</td>
</tr>
<tr>
<td>Linux pgo</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)</td>
<td>R-e10s(1 2 3 4 5 bc1 bc2 bc3 dt)</td>
<td>R-e10s(C R-e10s) T(c d g1 g2 o s t p) W(1 2 3 4)</td>
</tr>
<tr>
<td>Linux debug</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)</td>
<td>R-e10s(1 2 3 4 5 bc1 bc2 bc3 dt)</td>
<td>R-e10s(C J R1 R2)</td>
</tr>
<tr>
<td>Linux x64 opt</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)</td>
<td>R-e10s(1 2 3 4 5 bc1 bc2 bc3 dt)</td>
<td>R-e10s(C J R) T(c d g1 g2 o s t p) W(1 2 3 4)</td>
</tr>
<tr>
<td>Linux x64 pgo</td>
<td>B Cpp Jit1 Jit2 Ld Mn Wr X M(1 2 3 4 5 JP bc1 bc2 bc3 dt gl oth p)</td>
<td>R-e10s(1 2 3 4 5 bc1 bc2 bc3 dt)</td>
<td>R-e10s(C R-e10s) T(c d g1 g2 o s t p) W(1 2 3 4)</td>
</tr>
<tr>
<td>Linux x64 asan</td>
<td>Bd Bo Cpp Jit1 Jit2 M(1 2 3 4 5 JP bc1* bc2 bc3 dt* gl oth p)</td>
<td>R-e10s(1 2* 2 3 4 5 bc1* bc2 bc3)</td>
<td>R(e10s(C J R)</td>
</tr>
<tr>
<td>Linux x64 debug</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)</td>
<td>R-e10s(1 2 3 4 5 bc1 bc2 bc3)</td>
<td>R-e10s(C J R1 R2)</td>
</tr>
</tbody>
</table>

Legend:
- **B**: Build
- **M**: Main
- **Wr**: Write
- **Mn**: Main
- **V**: View
- **Ld**: Load
- **H**: High
- **Jit1, Jit2**: JIT compilation
- **Cpp**: C++
- **Bo**: Build options
- **R**: Run
- **-e10s**: -e10s option
- **-p**: Platform
- **-M**: Main
- **-V**: View
- **-Ld**: Load
- **-H**: High
- **-Jit1, Jit2**: JIT compilation
- **-Cpp**: C++
- **-Bo**: Build options
- **-R**: Run
- **-e10s**: -e10s option
- **-p**: Platform
Deterministic hardware
Sources of nondeterminism
Record inputs
Replay execution
“Old idea”

Nirvana
Chronomancer
PANDA
Scribe
CLAP
FlashBack
ReTrace

PinPlay
Jockey
ReVirt
ReSpec
ODR
Echo
QuickRec
rr goals

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

- Deterministic user-space CPU execution
- System call results
- Signals

Linux process boundary

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>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>
<tr>
<td>Cheap block copies</td>
<td>FIOCLONERANGE</td>
</tr>
</tbody>
</table>
ptrace

rr before_syscall

Kernel read() after_syscall

... record results ...
Use modern HW/OS features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Tool/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>
<tr>
<td>Cheap block copies</td>
<td>FIOCLONERANGE</td>
</tr>
</tbody>
</table>
Data races
Data races
# 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>
<tr>
<td>Cheap block copies</td>
<td>FIOCLONERANGE</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>Tool</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>
<tr>
<td>Cheap block copies</td>
<td>FIOCLONERANGE</td>
</tr>
</tbody>
</table>
Accelerating system calls

rr  before_syscall

Kernel read()  after_syscall

... record results ...
Avoid context switches

librrpreload.so
shim_read()

Kernel read()
... record results ...

Suppress ptrace trap
Use seccomp-bpf predicates
Blocking system calls

librrpreload.so

shim_read()

Kernel read()

... record results ...

Blocks?
Blocking system calls

read()  ...  thread 1

kernel  DESCHED perf event  rr

thread 2
Other issues

RDTSC
RDRAND
XBEGIN/XEND
CPUID
rr Overhead

![Bar chart showing overhead relative to baseline for different workloads: cp, octane, htmltest, sambatest. The Y-axis represents overhead relative to baseline, ranging from 0.00 to 2.50. The X-axis represents different workloads. The bars are color-coded: blue for Record, red for Replay, and yellow for Single Core.](image-url)
Reverse-execution Debugging

Chandler Carruth
@chandlerc1024

Debug on Linux at all? Stop and go get `rr`
*RIGHT NOW*. Biggest improv. to debugging
for me ever. H/T Justin Lebar.
Lessons

Replay performance matters

Session-cloning performance matters
→ Cloning processes via fork() seems cheaper than e.g. cloning VM state
Lessons

In-process system-call interception is fragile
→ applications make syscalls in strange states (bad TLS, insufficient stack, etc)
→ in-process interception code could be accidentally or maliciously subverted
→ move this part into kernel?
OS design implications

Recording boundary should:
→ be stable, simple, documented API boundary
→ also be a boundary for hardware performance counter measurement

Linux kernel/user boundary is this (mostly)
Windows kernel/user boundary is not
retry:
LDREX r0,[addr]
ADD r0,1
hardware interrupt???
STREX r1,r0,[addr]
CMP r1,0
BNE retry

→ Need hardware support to detect/compensate
→ Or binary rewriting
Related work

VM-level replay ... **heavyweight**
→ ReVirt, VMWare, QEMU (PANDA), Xen
Kernel-supported replay ... **hard to maintain**
→ Scribe, dOS, Arnold
Pure user-space replay ... **instrumentation, higher overhead**
→ PinPlay, iDNA, UndoDB
Higher-level replay ... **more limited scope**
→ Chronon, Dolos, Chakra, R2
Parallel replay ... **more limited scope, higher overhead**
→ SMP-ReVirt, DoublePlay, ODR, Castor
Hardware-supported parallel replay ... **nonexistent hardware**
→ FDR, BugNet, DeLorean, QuickRec
Conclusions

rr’s approach delivers a lot of value

More research needed for multicore approaches

Lots of unexplored applications of record+replay
Time magic has unlimited potential.