



Be Kind, Please Rewind

Adventures in creating a macOS record/replay debugger

LEGAL



This happened at day job, but stuck with us. BTW we work at google, but this talk does not necessarily represent the companies views or opinions. Happy now legal?



PETE: Before we go further we should address the elephant in the room. This talk is about our attempt at constructing a record replay debugger and the challenges and experiments we did along the way, we're not releasing a tool today. The effort is ongoing, and we have many pieces assembled together but its not ready for consumption just yet.

Also brief whoami

Why?

```
NSURLSessionDataTask *task = [_session
    dataTaskWithRequest:request
    completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable response,
        NSError * _Nullable err) {
        ...
        *stop = YES;
    }];
```



```
Thread 2 Crashed:: Dispatch queue:
com.apple.NSXPCCConnection.user.com.google.santa.metricservice.63335
0  libobjc.A.dylib      objc_msgSend + 29
1  Foundation           -[NSError copyWithZone:] + 107
2  santametricservice   -[SNTMetricHTTPWriter write:toURL:error:] +
1372
3  santametricservice   -[SNTMetricService exportForMonitoring:] + 475
```

Why? Dangling stack pointer in a timer-based callback which fire a single bit into a stack frame in which another thread would be running. “Physically impossible” bugs manifesting.

Record/Replay: Prior Art

- PANDA (2020)
 - Whole system!
- WinDbg (2017)
- RR (2014)
- Scribe (2010)
- Jockey (2005)
- Flashback (2004)
- ReTrace
- QuickRec
- Revirt (1999)
 - Whole system!


This is a super old idea. There have been countless talks and papers over the years on this idea of record and replay.

Record/Replay: Prior Art

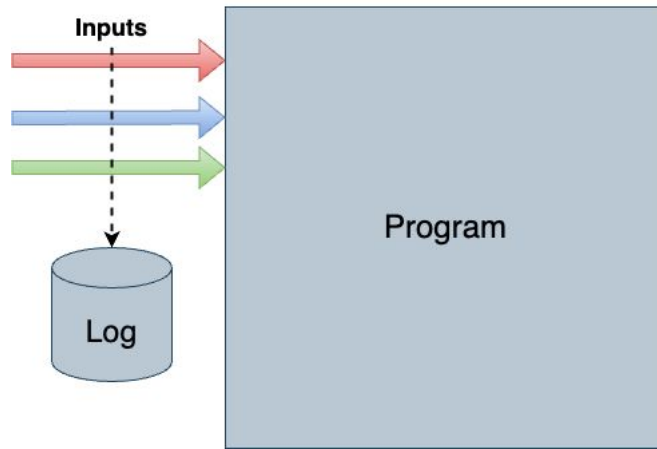
- PANDA (2020)
 - Whole system!
- WinDbg (2017)
- RR (2014)
- Scribe (2010)
- Jockey (2005)
- Flashback (2004)
- ReTrace
- QuickRec
- Revirt (1999)
 - Whole system!



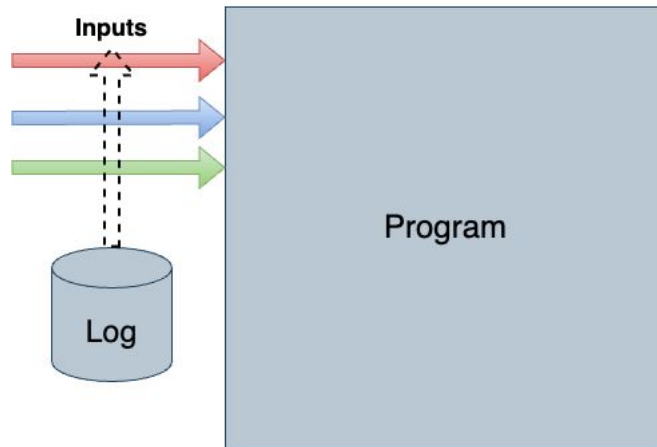
None of them support macOS / XNU.



Record / Replay Basics & Goals



PETE: Basically the idea is that if we can record all inputs and sources of randomness into a process during its execution into a log.



PETE: Then if we replay an application reading the inputs from the log instead of the real system we should get the same execution again which would let you debug hard problems. If we know we can get the same exact execution again we can set breakpoints or watches to triage bugs.





Aside: the bug we

Goals for our tool

- Only focusing on user-space programs
- Easy to use and deploy – needs to support a stock MBP with/M1,M2
- No DBI / code instrumentation
- Small investment of effort to maintain
- Fast enough to use on real programs




Like RR we'd like to build something that's usable. We're only looking for something that's easy to use and debug. In our case we get bugs across a large number of machines and need a user to run the tool. Additionally Apple's kicking everyone out of the kernel, so we don't want to use a KEXT. Ideally long term this should lead to a lower effort to maintain and ultimately we need this to be good enough to work on real programs that we did not compile ourselves.

RR's Requirements for User-Space Replay

Requirement	Does macOS Meet This Out of the Box?
Ability to Record Syscalls	
Ability to Record Syscalls Outside Libc	
Ability to determine if a Syscall is blocking	
Ability to Intercept Signals	

About 3/7 requirements aren't easily met. Specifically I mean that there's an obvious way to do this not that something can't be done e.g. direct syscalls I'm talking about a seccomp or a ptrace sysemu.

RR's Requirements for User-Space Replay (Part 2)

Requirement	Does macOS Meet This Out of the Box?
Ability to pin a process to a single core (cpuset)	
Ability to trap non-deterministic instructions	
Ability to access reliable and deterministic hardware performance counters	

So it's not hopeless, but the 4 strikes and a yellow



Handoff to Nick

Recording: It's not that simple...

- Mach traps
 - Close enough to syscalls, no big deal
 - ... except for a few traps which don't have normal hook points
- Signals
 - The outside world can “asynchronously” poke the target
- mmap
- Multithreading
 - Well-formed programs shouldn't have issues (data races), but...
 - Aside: thanks Apple for not giving us cpuset - no easy way to pin to one core
- Commpage
 - Similar to vvar (normally accessed via vDSO) on Linux
- Non-deterministic instructions – mrs x0, cntvct_el0

In addition to syscalls...

No normal hook points: E.g. gettimeofday

Mmap: both shmem, file writing, kernel filling page (e.g. mach traps), etc.

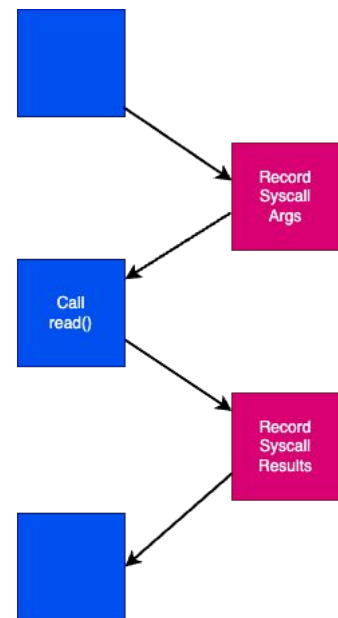
cntvct_el0 = counter



Recording: Syscalls & Traps

Recording: Syscalls on macOS

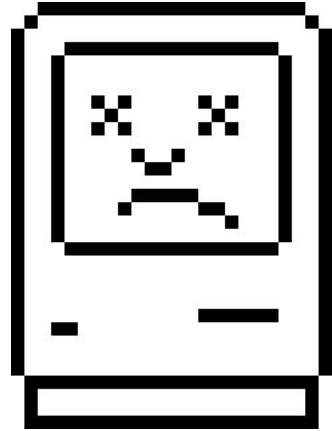
- 3 types – BSD, mach traps, machine dependent
- Need pre- and post-hooks for data gathering



These are our requirements

Thanks Apple

- Gutted ptrace implementation – no sysemu
- No seccomp-bpf equivalent



Basically no equivalent way to intercept syscalls, traps, etc. to another userland program

One Option: dtrace

- dtrace hooks, storage, etc.
- Not enough to capture arbitrary syscall data though
 - No conditionals for example - not possible to `switch` in “multiplexed” syscalls
- Strictly async
 - How to pause so userland can get what it needs?
 - Luckily there are “destructive” actions
 - `signal(STOP)`
 - `stop()` - `mach_task_suspend`
 - These only take effect *after* the syscall is processed though...

One Option: dtrace



dtrace also has a nasty habit of panicking the system at least in recent kernels

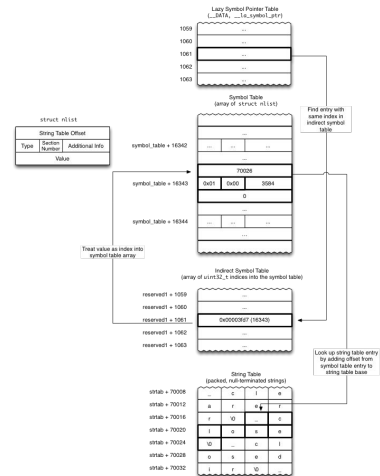
Seatbelt / Sandbox?

- Seatbelt is wired up into every syscall maybe?
- Trace mode for recording
 - Not a good API, minimal log entries
- No way to not kill on replay

There is a trace mode (not just kill on violation). The issue here is the ability to record and replay syscalls values - we can't do arbitrary data fetches

Interposing / Dynamic Interposing / Symbol Rebinding

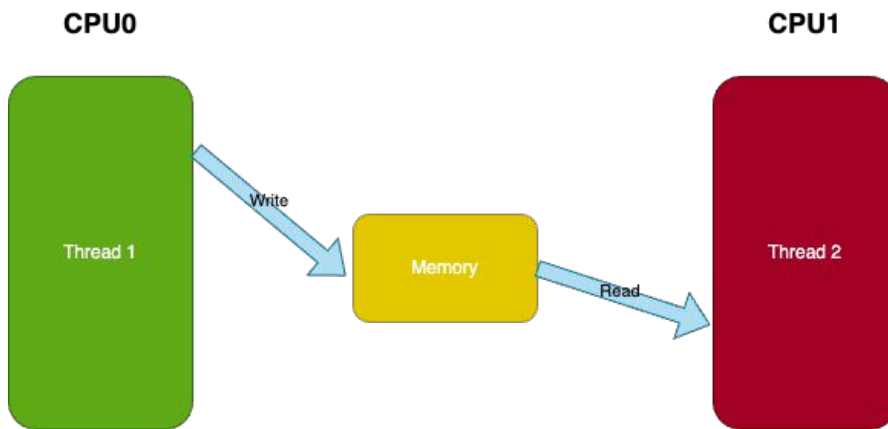
- macOS is a BSD!
 - ABI compatibility is at the libc level not the kernel
 - Can we just hook `libsystem_kernel`?
- We can interpose on the symbol
 - Could use [fishhook](#)
- Doesn't catch direct syscalls...



Go got bitten by the syscall numbers changing a while back. Direct syscalls *should* now be rare, but nothing says syscalls have to come from libSystem



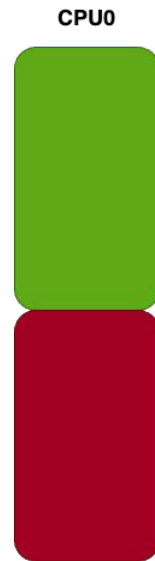
Handoff to Pete



Pete: I copied this diagram from the RR talk as I think it illustrates a data race pretty nicely. Scheduling with multiple threads running in parallel becomes yet another source of non-determinism. We can have a read after write or a write after write instance in which the schedule ordering will make some bugs appear or not. If the cores are clocked differently this can exacerbate the issue. Its something we need to deal with as a lot of macOS apps often use GCD so it's really common to have multiple threads. So how does RR handle this?

How Does RR Handle This?

- Only runs one thread to run at a time (non-parallel)
- Limits threads to the same core using processor affinity
- Schedules threads and records the choice in the log (**can mixup order on replay to find bugs**)



Pete: RR simply avoids this problem. They then deterministically schedule threads and record the choices of which thread ran since this is now a non-deterministic factor that must be accounted for. In the event that they picked wrong and can't reproduce the bug RR has a chaos mode which will randomize the ordering here to try and repro the bug.

This also has the benefit of instructions like CPUID, will return the same values.

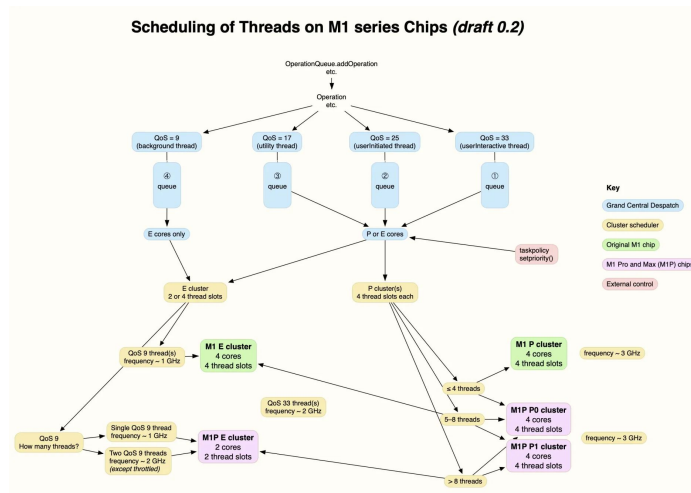
Thread scheduling on macOS not guaranteed

- No `cpu_set(3)`
- Can we use `THREAD_AFFINITY_POLICY`?

```
/*  
 *      thread_bind:  
 *  
 *      Force the current thread to execute on the specified processor.  
 *      Takes effect after the next thread_block().  
 *  
 *      Returns the previous binding.  PROCESSOR_NULL means  
 *      not bound.  
 *  
 *      XXX - DO NOT export this to users - XXX  
 */  
processor_t  
thread_bind(  
    processor_t      processor)  
{  
    thread_t      self = current_thread();  
    processor_t    prev;  
    _
```

Pete: So how can we limit the number of cores our user space program runs on? Can we just pin threads to cores on macOS?

P-cores and E-cores



We also have to contend with the different types of cores on apple silicon, and while we can hint to the kernel which one we want with QOS, we can only prefer P or E, not a *specific* P or E

Can we shutdown cores?

- In the old OS X internals books there was an example showing how to shutdown cores using `processor_exit`
- Can we just limit ourselves to a core?

```
[ user@waterville ~ ]
$ sudo ./print_processors
Password:
Number of processors: 12
CPU: slot 0 (master)
CPU: slot 1
//snipped.
CPU: slot 11
[ user@waterville ~ ]
$ sudo ./processor_xable
processor_exit: (os/kern) service not supported
```

Pete: In the old Amit Singh book, he had an example where you first print out the cores using the `host_port` and `host_priv` ports then had you in a second example shutdown the last core. Can we do the same?

Pete: Nope

So in order to be able to record and replay we need to find a means of limiting the number of cores a process is running on

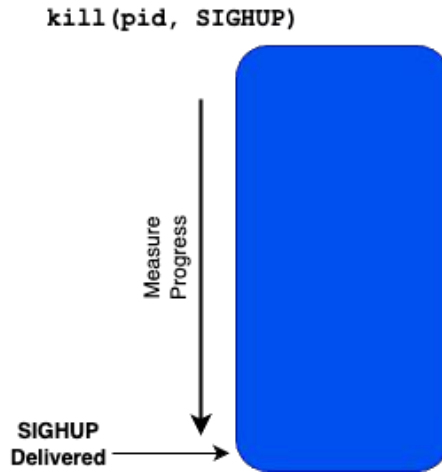
You could

A photograph of vintage VCR and VHS tape components. In the foreground, a black VCR control panel with various buttons like 'PLAY', 'STOP', 'PAUSE/STILL', and 'REWIND' is visible. Above it, a silver VCR unit is partially shown. To the right, a VHS tape is open, revealing its internal reels. The text 'Recording: Asynchronous Events' is overlaid in white on a dark rectangular background in the center of the image.

Recording: Asynchronous Events

Signals & Scheduling

- Need to be able to intercept signals and record register state of where the signal was delivered or program interrupted for scheduling.
- Need to know where you are in the programs execution so you can inject your signals in the right place during replay
- Replay: when using something interrupt driven must account for late firing interrupts



PETE: Basically we need to record exactly where and when an signal or interrupt happened. This is a superhard problem. If we can do this then we can handle scheduling as if we injected an interrupt at a given time.

Using PMUs from macOS

- RR works on Asahi Linux and uses the PMU can we?
 - Uses the count of retired conditional branches as progress indicator (0x8c)
 - Can reset for an interrupt when replaying
- macOS does not have an interface for setting PMUs from ELO

```
[ user@waterville ~/src/pmu_counters ]
$ sudo ./counter_test
loaded db: a15 (Apple A15)
number of fixed counters: 2
number of configurable counters: 8
counters value:
    cycles: 41865278
    instructions: 91998218
    branches: 21071096
    branch-misses: 53779
[ user@waterville ~/src/pmu_counters ]
$ sudo ./counter_test
loaded db: a15 (Apple A15)
number of fixed counters: 2
number of configurable counters: 8
counters value:
    cycles: 41946121
    instructions: 92093331
    branches: 0
    branch-misses: 0
```

PETE: Can we access the in macOS PMUs? This is available from kperf and kpc

- Sorta – Crashes!
- Unreliable scheduling
- No way to set interrupts for replay

```
panic(cpu 5 caller 0xffffe0017c66cd8): kperf: timer fired  
at 2793246644070, but sampling is disabled  
@kptimer.c:328  
Debugger message: panic
```

PETE: It's also pretty common to get a delayed interrupt firing when you're turning these on and off again in XNU which can result in a kernel panic. See me after the talk if you want to watch me make my mac angry.

Supporting Nondeterministic Execution in Fault-Tolerant Systems*

J. Hamilton Slye

Dept. of Electrical and Computer Engineering
Carnegie Mellon University
ham+@cmu.edu

E.N. Elnozahy

Department of Computer Science
Carnegie Mellon University
mootaz@cs.cmu.edu

Abstract

We present a technique to track nondeterminism resulting from asynchronous events and multithreading in log-based rollback-recovery protocols. This technique relies on using a software counter to correlate

with the end users [11]. Efficient tracking of nondeterminism is thus crucial to supporting interactive applications [14].

Different flavors of logging have been suggested with different performance and resilience character-

Decrement Register

Branch to handler if register = 0

PETE: So what do we do if we don't have the PMU? Not having access to a PMU is an old problem, that's been solved for a while. In this case the classic solution proposed is to reserve a general purpose register and rewrite the branches so that they decrement the register, when taken.

While recording you first set the counter to the max uint64 value. then when you get a signal you simply record the counter value and the register state into the log then reset and repeat, no need to worry about underflow because. During replay you set the counter to

Options without PMU or DBI

- We can count the number of syscalls and then single step forward then inject the signal (set a breakpoint and invoke the signal handler)
- Do what scribe(10) does and simply deliver the signal at the next syscall and replay interrupted syscall (special case for signals like SIGSEGV that originate in user space.)
- If we need to go further than say 10,000 instructions we can use an high res clock (e.g. pacman) to trap back to us

```
// set up the registers for our counter
asm!(
    "mov x3, {running}",
    "mov x4, x0", // store backed up copy of
    "ldr x0, [x0]", // load the value of coun
    "1:",
    "isb",
    "ldr x2, [x3]", // load the value of run
    "sub x0, x0, #1", // decrement the counter
    "cbz x2, 456f", // break the loop if we'
    "cbnz x0, 1b", // if we've not underflo
    "123:",
    "mov {interrupted}, #1", // set interrupte
    "456:",
    "str xzr, [x3]", // set running to false
    "str x0, [x4]", // store the value of co
    in("x0") count_addr,
    interrupted = out(reg) interrupted,
    running = in(reg) running_addr,
    );
```

If we have to use the timer we simply set it lower than we need and single step



So after running into all of these issues we looked around to see if others had implemented something similar to what we were hoping for and thats when we found darling.

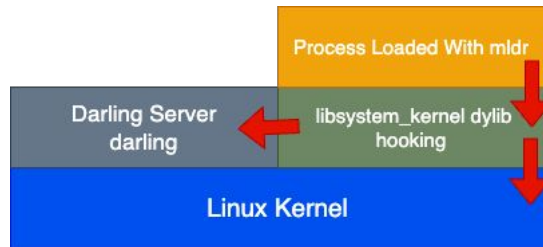
Darling

- “A Translation Layer that lets you run macOS software on Linux”
- Uses a custom loader, interposing of libsystem_kernel, a lot of duct tape code and userland a server to translate macOS syscalls to Linux syscalls
- Can run software like xcode on Linux



PETE: We looked around for something

High Level: How Darling Works



Essentially they use a custom loader to bootstrap dyld in their own process address space and then call back into their hooked `libsystem_kernel`. For things that need `mach_traps` and callback notification etc. they use a user land server they talk to over a unix socket. Together this produces a workable solution on linux for a larger number of command line programs.

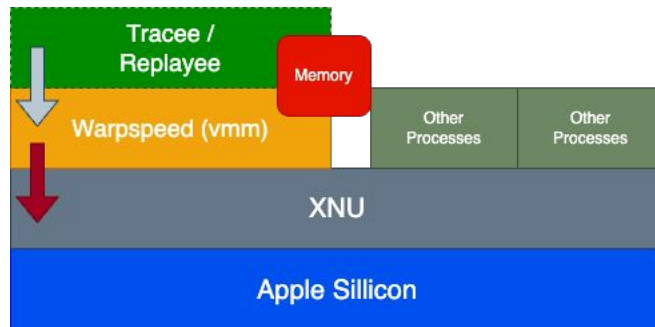


Handoff to Nick

Because everything has to have a good codename

Warp speed

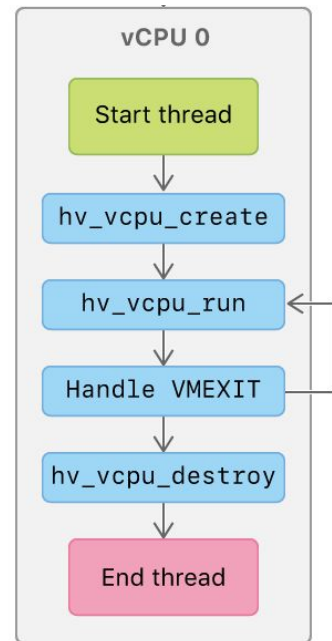
- Isolate target inside a VM with 1 core
- Proxy syscalls
- Both signal slide + SoftPMU to approximate program progression
- Manual thread scheduling



NB this is *not* full macOS in a VM, just the program at EL0
... one of the key things that enables this is Hypervisor.framework

Hypervisor.framework

- Super light-weight framework
 - Little as possible in the kernel
- Usage:
 - Create a VM
 - Map memory (from hypervisor address space)
 - Create vCPU
 - Set regs
 - Run
 - Trap out to userland on VM exit
 - GOTO 5
 - *That's it*



Map memory from the hypervisor's VA space, *not* a new address space. Similar to KVM, but even simpler in ways

Warp speed: VM/Hypervisor

- Use modified darling's loader (mldr) to map in target program and dyld
- Load in shared cache
- "Share" an address space with the guest
 - 1:1 map the regions of the loaded target into VM at the same virtual address
- Trap out and forward syscalls
- All based on Hyperpom (Rust!)
- Lets us control the execution of the program perfectly
 - Only have one virtual core
 - Manually schedule threads



This is a VM, if there is any way to get info from the host without trapping that is a bug in the framework

Load in shared cache != fake out all of what dyld does.

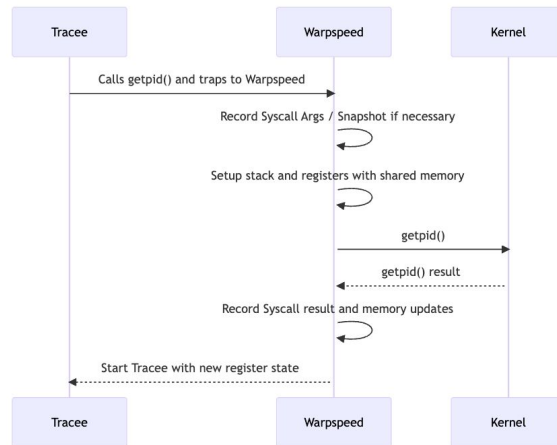
Dyld is not a simple thing, as you'll see shortly even a simple program ends up doing *hundreds* of syscalls as part of dyld init

Hyperpom is a research fuzzer for m1, provided the base for MMU setup, has snapshotting, etc.

Of course it's rust, we're hipsters from Brooklyn

Perfect control even without ptrace

Warpspeed: VM/Hypervisor



dyld



If you want to get a sense of how complex dyld (and any program pre-init) is, this is taken from Levin's **OS internals* book

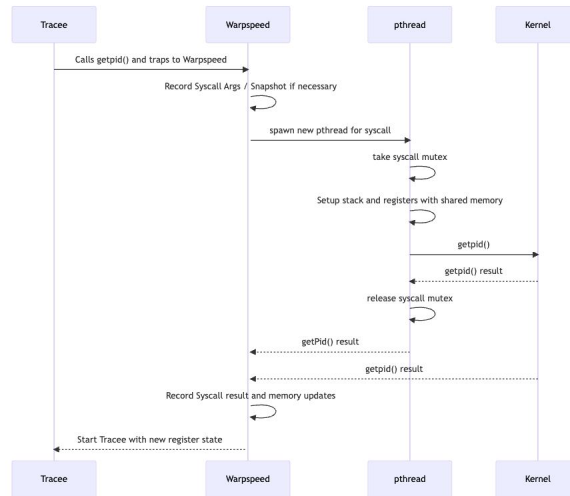
Warpspeed: Unimplemented Features

- LLDB/GDB interface
- Optimizing/compressing log format
- The hypervisor itself is responsible for performing the syscalls
 - What happens on a blocking call?
 - Could deadlock on mutex wait
- Handling blocking syscalls
 - Manually enumerate and perform some non-blocking alternative
 - or...

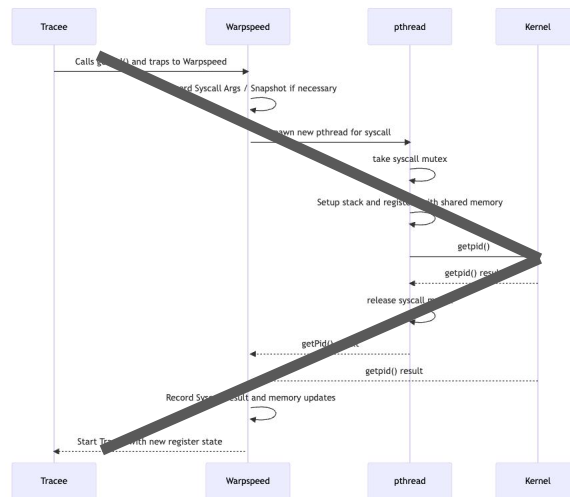
We obviously want this to be usable under a debugger, so we need some way to interface with lldb/gdb to receive rwatch, rstep, etc.

Log format is simple right now

Warpspeed: VM/Hypervisor



Warpspeed: VM/Hypervisor



Totally totally why this is called warpspeed (seq diagram)

Warpspeed: Outstanding Issues

- MMIO
- Entitlements

RR takes the approach of “wontfix” for MMIO. Entitlements is a real unanswered question though. Hardened runtime isn’t an issue though since we do our own loading



Replay

- If you can figure out recording, replay is much simpler
 - Set breakpoints where something happened in recording
 - Mimic side-effects
 - Continue
- SoftPMU needed here in case we end up with an async event in a hot loop

Replay: GUI

- UI is core to macOS
- How can we “pass through” events on replay to the OS (to see the app running) while not introducing nondeterminism?
 - *In theory* it will “just work”
 - No (easy) way to show the UI on replay though

Not part of the original motivation, but of course people are going to wonder
The replay should behave as expected since the mach ports don't actually get
created and messages get put down just as they did in the recording
But of course you won't see anything

Basically: TODO

Recap

- Tool is WIP
- But principles work!
- Stay posted for more



Questions?