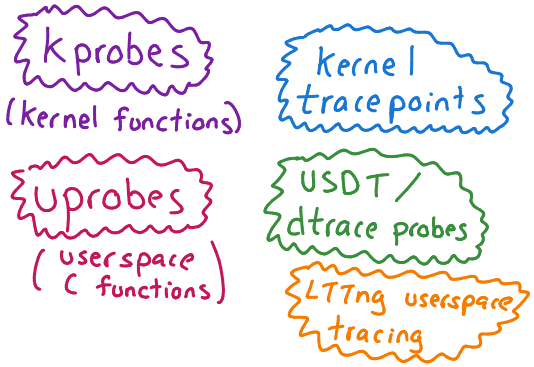
♡

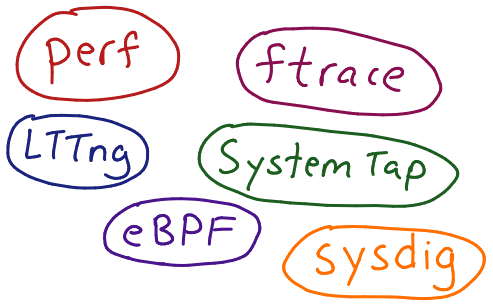
like this?
you can print more!
for free!
http://jvns.ca/zines

Linux tracing systems & how they fit together

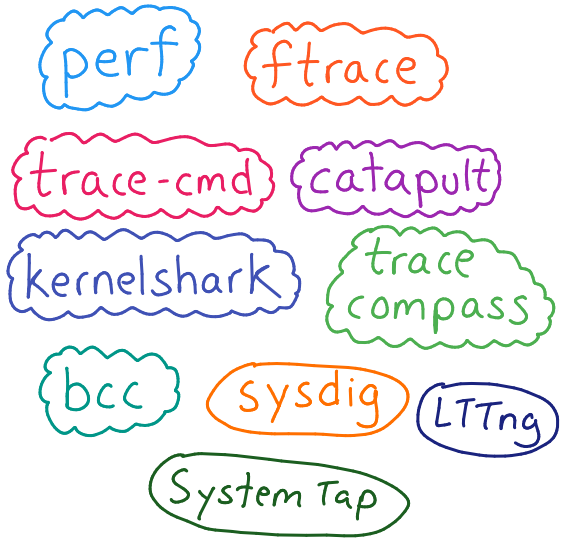
Data sources:



Ways to extract data:

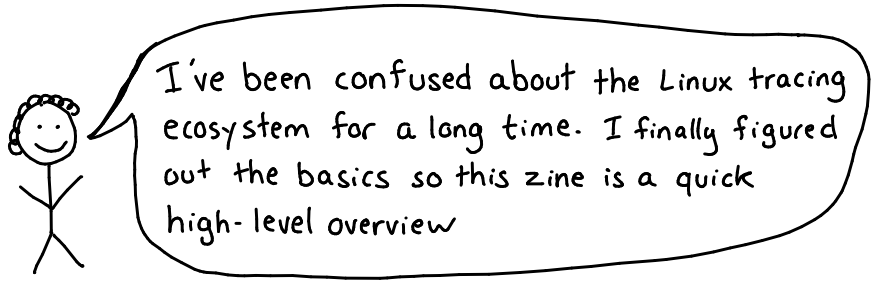


frontends:



by Julia Evans

what's this?



JULIA EVANS
@b0rk
https://jvns.ca

thanks for reading

To learn more:

- brendan gregg's blog
- the kernel docs on kprobes / ftrace, in the Documentation folder
- LWN has a bunch of useful articles on ftrace

Why eBPF is exciting

→ it supports a ton of data sources (kprobes / uprobes / USDT probes / tracepoints

→ you can write your own programs and insert them into the kernel so it's high performance and flexible

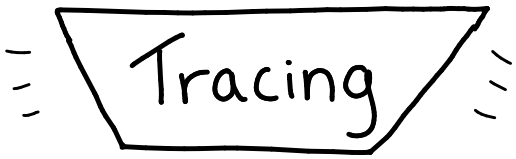
→ it's pretty safe: what eBPF programs can do is strictly limited by the kernel (no loops! no arbitrary memory access). Every program runs through a verifier before it can run.

→ people are building cool easy to use tools with it (strace built with eBPF? yes please!)

Brendan Gregg's blog has a TON of posts about eBPF, and

https://github.com/iovisor/bcc

has lots of tools written using it, and makes it easier to write your own



Let's say you want to

- see every time a certain function is called (and its arguments)
- see every time an 'event' happens (like the CPU switching which process it's running - that event is called sched_switch)
- define your own tracing events
- aggregate (to see exactly how much time was spent in a function)

to do this, we need to:

- define tracing events (either at compile time or at runtime). aka **data sources**

- a way to collect delicious tracing data and send it to userspace. Usually something in the kernel collects tracing data.

- a frontend to use!

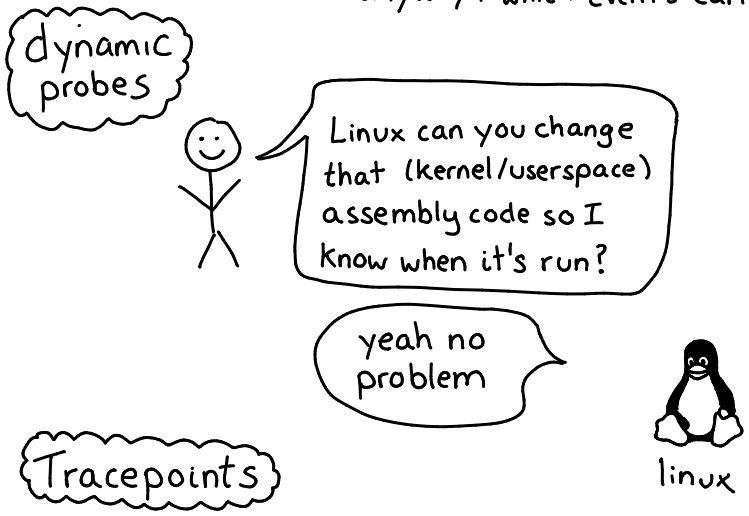
let's go see what the options are →

(the ecosystem is a little fragmented ¯_(ツ)_/¯)

data sources

There are 2 basic kinds of data sources:
(not quite the right terminology but I'm not sure what is)

- 'dynamic probes': change your assembly code at runtime to instrument it
- 'tracepoints': choose at compile time (or in advance anyway) which events can be traced.



① Compile a tracepoint into your program
(you can also often define them at runtime)

② as long as nobody activates it, ~no overhead!

③ Your users can activate the tracepoint (with tools like ftrace/dtrace + friends) to get info about what your program is doing.

more frontends

bcc for eBPF

Python framework to help you write eBPF programs. Also tons of examples!

https://github.com/iovisor/bcc

catapult for ftrace

Can draw graphs of sched_switch events recorded by ftrace.
(and maybe more things? unsure.)

Kernelshark for ftrace

graphical trace-cmd frontend
haven't tried it yet

LTTng / sysdig / SystemTap

all frontends for their respective data collectors

Frontends

tools to help you:

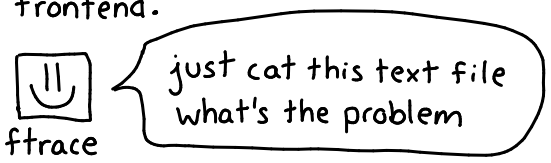
- tell the kernel what data to collect / programs to run
- display the data in a useful way

perf trace
for perf + ftrace

'perf' can use perf_event_open (surprise) and also ftrace to record tracing data. I use 'perf trace' to trace syscalls.

ftrace

ftrace by itself doesn't really have a frontend.



trace-cmd
for ftrace

A command line frontend to ftrace, a lot easier to use.

perf-tools
for perf / ftrace

A collection of scripts by Brendan Gregg. The kprobe/uprobe scripts are fun to play with!

Here are the 5 data sources the tools in this zine use:

kprobes
kernel

let you trace any instruction / function call / function return in the kernel. kprobe.txt in the kernel docs says more.

uprobes
userspace

like kprobes, but for userspace programs!

tracepoints:

kernel tracepoints
kernel

these are defined by a TRACE_EVENT macro. For example there are 2 tracepoints (enter/exit) for every syscall

dtrace probes aka USDT probes
userspace

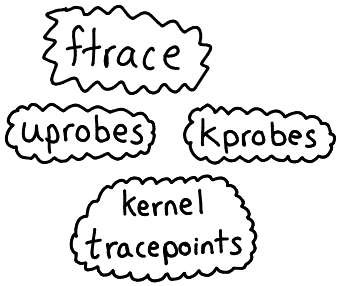
dtrace isn't a Linux program, but lots of programs (like python/mysql) can be compiled with dtrace probes. And there are Linux tracing tools that can use those probes!

lttng-ust
userspace

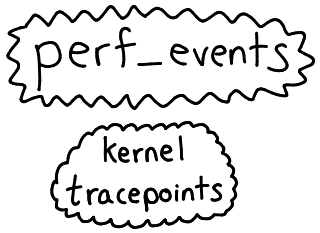
lttng-ust is a tracing format (works with LTTng) that works entirely in userspace.

Ways to get (delicious delicious) tracing data

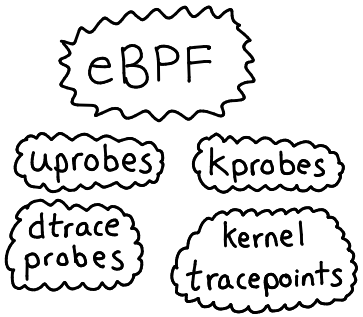
There are a bunch of ways to collect tracing data. These 3 are the ones that are built into the Linux kernel.



magical filesystem at /sys/kernel/debug/tracing. Super powerful, you interact with it by reading from / writing to files.



① call the perf_event_open syscall
② the kernel writes data to a ring buffer ("perf buffer")

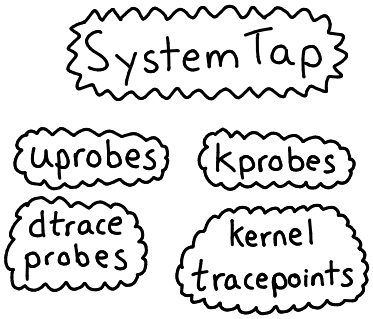


The newest and most powerful

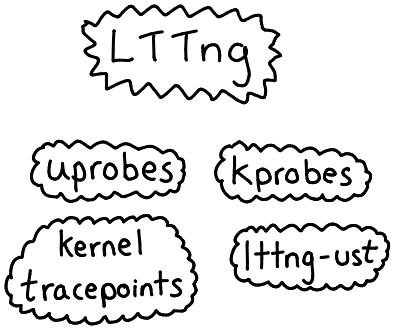
① Write a small eBPF program
② Ask Linux to attach it to a kprobe / uprobe / tracepoint
③ The eBPF program sends data to userspace with ftrace / perf / BPF maps

more ways

These are all developed outside the kernel (though they all ultimately insert kernel modules)



① Write some C code
② Compile it into a custom kernel module
③ Insert that module into the kernel



① Insert the LTTng kernel module
② Use the LTTng tools to get it to collect data for you

just traces system calls I think