WIP/CFT: FreeBSD Boot Performance

BY TOM JONES AND MITCHELL HORNE

For a lot of us, the time a FreeBSD system takes to start up is mostly decided by how long the machine takes to go from power through the system firmware and into loader. On server hardware, the first stages to initialize the system and get to loader can take minutes. In this environment, it is hard to worry much about a couple of seconds in the FreeBSD boot process.

In a cloud environment, where you are charged by the second, time spent in the boot process is time wasted, it is time you are billed for from which you don't get any value back. Faster boot times mean that your platform can be used in dynamically scaled environments more easily. If your system takes minutes to boot versus tens or single digit seconds to boot, you must guess more about future load, rather than being able to spin up and down hosts as demand requires them.

Cloud environments are where the first set of tools to improve the performance of FreeBSD boot were added. In 2018, Colin Percival presented "Profiling the FreeBSD kernel boot: From `hammer_time to start_init`" at AsiaBSDCon. Colin's work added a timestamped event log—called TSLOG—to the kernel, which can be used to track the time the kernel spends in each subsystem during boot [https://papers.freebsd.org/2018/bsdcan/percival-profiling_the_freebsd_kernel_boot/].

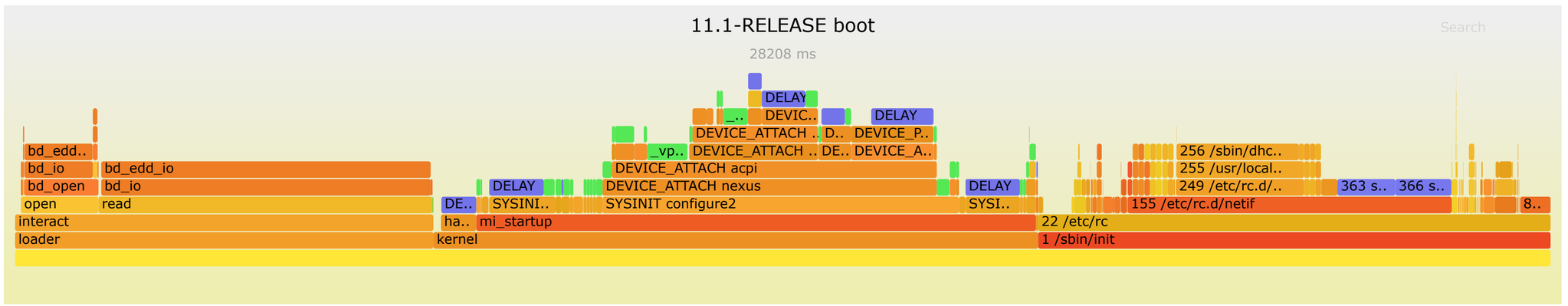
The TSLOG framework traces events that are compiled into kernels with the TSLOG option. Events are implemented with macros which compile to nothing when the option isn't present. TSLOG events go into a buffer and accumulate until the buffer is full and then they are silently dropped, this allows TSLOG to capture early events in preference to later ones in the system.

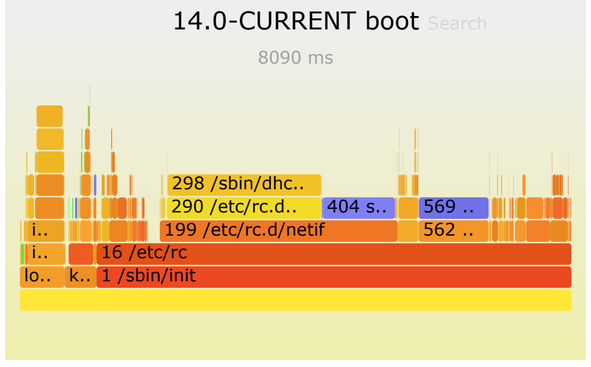
On server hardware, the first stages to initialize the system and get to loader can take minutes.

With TSLOG, the boot time can be tracked and analyzed, to analyze the output log from the system. Colin uses timeshare plots called FlameCharts that work well for this, FlameCharts are like FlameGraphs [https://www.brendangregg.com/flamegraphs.html], but are sorted in chronological order rather than alphabetically.

In each flame graph, the horizontal time represents how much of the boot process is spent in each area, broken down vertically by sub system.

WIP/CFT: FreeBSD Boot Performance





Figures 1 and 2 show how Colin and others have used TSLOG to find slow periods in the boot process. Fgure 1 shows approximately where we started in 2017 and Figure 2 shows the time they have managed to trim out of the boot process in last 5 years, with most of the of the hard work happening recently.

BootTrace

TSLOG is a great way to discover where the kernel was spending time during start up. In early 2022, Mitchell Horne began upstreaming a framework from NetApp called BootTrace. BootTrace allows us to perform the same sort of tracing as TSLOG, but with a couple of major enhancements that give us even more coverage. BootTrace provides the enhanced ability to trace userspace processes, allowing us to cover the rc subsystem, and we can trace the shutdown process.

Shutdown process is difficult to trace with TSLOG because when it is done, the host is gone. In some ways, shutdown (or reboot) is just as important as startup, time spent putting the system to bed helps with system consistency, but the host really isn't doing work during shutdown. BootTrace works around this limitation and gives us a great total view into how the system starts and stops.

How to Contribute

We now have a wealth of tooling to look at the performance of the FreeBSD boot and shutdown process, but developers can only look at the systems and application workloads to which they have access.

Colin is focused on running FreeBSD in Amazon Web Services. There are many, many more cloud providers around and each is likely to have areas that can be tuned to get the fast boot and shutdown times out of a FreeBSD system.

The best way to contribute to improving FreeBSD performance in these areas is to run the tooling we have and share the resulting FlameCharts. Colin is eager to have boot FlameCharts for more and non-cloud systems. You can use the FlameCharts to figure out where the 'hot spots' (or maybe they are 'cold spots') are in the boot and shutdown processes. Once these have been identified to the FreeBSD project, developers can figure out how to improve performance in each case.

TSLOG lead to large improvements in the FreeBSD boot process. FreeBSD boot has gone from ~30 seconds in 2017 on 11.1-RELEASE when Colin started and is down to ~9 seconds in

WIP/CFT: FreeBSD Boot Performance

March 2022 on 14-CURRENT. Some of these gains are in the form of second-sized improvements, but plenty of others came as 100ms reductions in the time subsystems spent initializing.

There is still a way to go to get FreeBSD down to clear Linux, which can boot in ~1 second on EC2. The way to get there is by testing with the boot profiling tools we now have and highlighting areas for improvement to developers in the project.

Tracing Boot Time with TSLOG

TSLOG is reasonably straightforward to run if you have experience building and using your own FreeBSD kernels. You need to build a custom kernel with an 'options TSLOG' and then run the script Colin has provided in the freebsd-boot-profiling repository [https://docs.freebsd.org/en/books/handbook/kernelconfig/].

The latest version of these steps should be on the Boot Time FreeBSD wiki page [https://wiki.freebsd.org/BootTime]. With the FlameChart from Colin's script in hand, you can now follow the much more difficult process of identifying areas in the boot process that take longer than they should.

As of writing, `rtsold` takes up a large portion of the userspace boot time on my systems. This is an important daemon for doing IPv6 autoconfiguration, but it might also be a good starting point for improving your systems boot performance if you use DHCP6 for your network configuration.

Tracing Boot and Shutdown Time with BootTrace

With FreeBSD 14-CURRENT the new BootTrace framework from NetApp is present in the FreeBSD kernel. BootTrace is built into the kernel but is disabled by default.

BootTrace allows the tracing of boot, run time and shutdown time events which it stores in three separate logs. On a FreeBSD 14-CURRENT system from March 2022 you can enable tracing by setting the `kern.boottrace.enabled sysctl` to 1 in `/boot/loader.conf`.

Once enabled, the system will output the boot and run time logs via the `kern.bootrace.log sysctl`.

```
 CPU     msecs      delta process          event          PID     CPUtime IBlks OBlks
   0    177873          0 kernel       sysinit 0x2100001   0        0.00     0     0
   0    177873          0 kernel       sysinit 0x2110000   0        0.00     0     0
   0    177873          0 kernel       sysinit 0x2140000   0        0.00     0     0
   0    177873          0 kernel       sysinit 0x2160000   0        0.00     0     0
  15    182874          0 kernel       sysinit 0xf100000   0        0.00     0     0
...
  15    182874          0 kernel       sysinit 0xfffffff   0        0.00     0     0
  15    182875          1 swapper      mi_startup done     0        0.00     0     0
   9    182880          5 init         init(8) starting... 1        0.00     0     0
   9    182880          0 init         /etc/rc starting... 1        0.00     0     0
  14    202622      19742 init         /etc/rc finished    1        0.61   909    23
Total measured time: 24749 msecs

CPU      msecs      delta process          event          PID     CPUtime IBlks OBlks
  14    202622          0 init         multi-user start    1        0.61   909    23
Total measured time: 0 msecs
```

WIP/CFT: FreeBSD Boot Performance

Shutdown tracing is possible by setting the `kern.bootrace.shutdown_trace sysctl` to 1 and shutting down the system. Shutdown tracing is quite a difficult problem. Until boot performance tracing at the end of the shutdown process, you don't have a system from which to pull the information. To work around this, BootTrace logs the shutdown log to the systems console. To recover the log, you will need to have a console that records message sent to it (such as serial).

The log from my test system looks like this:

```
CPU    msecs     delta process         event                                PID  CPUtime IBlks OBlks
 11  8089055         0 init    single-user from multi-user                    1     0.57   914    45
 11  8128611     39556 init    halt & poweroff from multi-user                1     1.38  1274   142
 15  8129849      1238 init    kernel shutdown (clean) started                1     1.69  1274   248
  0  8129849         0 init    system halting...                              1     1.69  1274   248
  0  8129849         0 init    system powering off...                         1     1.69  1274   248
  0  8132523      2674 init    shutdown pre sync complete                     1     1.69  1274   248
  0  8132523         0 init    bufshutdown begin                              1     1.69  1274   248
  0  8132524         1 init    shutdown sync complete                         1     1.69  1274   248
  0  8132615        91 init    shutdown unmounted all filesystems             1     1.69  1274   248
  0  8132715       100 init    bufshutdown end                                1     1.69  1274   248
  0  8132715         0 init    shutdown post sync complete                    1     1.69  1274   248
  0  8132715         0 init    shutdown final begin                           1     1.69  1274   248
```

BootTrace offers three write only `sysctls`, `boottrace`, `runtrace` and `shuttrace`. When written to an event will be logged as '`${procname}: name`'. These `sysctls` make it easy to add BootTrace logging to your own applications.

Dealing with the Results You Get

Once you have identified an area in the boot process in your environment, you then need to determine if it can be improved and, if possible, suggest how. From TSLOG results, you should look for items in the FlameChart that take up a substantial portion of the boot time first and dig down into those in the FlameChart svg.

More subsystems and userspace components could benefit from BootTrace events. Adding these into your workloads' start up and shutdown scripts can provide insight into where issues are and you might uncover issues that impact FreeBSD users generally.

Some subsystems build in long delays to allow other network hosts to synchronize. These sorts of delays might be prime candidates for trimming down the boot process.

FreeBSD developers welcome good bug reports via the bug tracker, if you can provide hints or patches that will fix boot performance issues, then all the better.

The FreeBSD boot process is never going to be finished. Over time, as services and hardware change, it is going to vary. A lot of the most significant boot performance benefits came from reducing interactions with emulated legacy hardware. On systems with many cores, loads of time was spent writing characters to the emulated VGA console. Over time, hardware will age out of use and faster or slower equipment will follow. With your help we can keep FreeBSD competitive in the cloud and stop wasting time starting up machines.

TOM JONES wants FreeBSD-based projects to get the attention they deserve. He lives in the North East of Scotland and offers FreeBSD consulting.