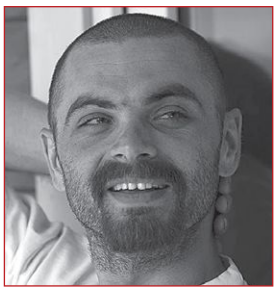
SEE TEXT ONLY

this month In FreeBSD

BY DRU LAVIGNE



An Interview with Gleb Smirnoff

Over the next few issues, we'll be taking a closer look at some of the new features making their way into the 2016 releases and the developers behind those features.

This month we chat with Gleb Smirnoff, a member of the FreeBSD core, release engineering, and security teams. He is also a senior software developer at Netflix Inc.

Q Tell us a bit about yourself. How did you get started with FreeBSD and what is your involvement with the FreeBSD Project?

A I started using FreeBSD back in 2000, when I was 18. At the university dormitory, we were building a LAN and we used FreeBSD for routing, Internet sharing, and web services. Quite quickly I went from documentation to sources. I was fascinated with `netgraph(4)`. In 2004, I received committer access, and from that point all my life is coupled with FreeBSD.

Q You have been working on the next generation of `sendfile(2)`, which is typically used to increase web server performance. Please provide an overview of the original `sendfile(2)` implementation. Does the new `sendfile(2)` address any of its shortcomings and what performance improvement does it provide?

A The original sendfile was quite straightforward. It locked the socket buffer to protect it from any other writes, grabbed memory for the requested data, and filled the memory with data taken from a disk. Once data was read from disk, it put the data into the socket buffer, unlocked the buffer, and returned.

The shortcoming is that it takes time to read data from a disk. As an example, let's assume it takes 5 milliseconds to read the requested data from a disk. This means we can do only 200 requests per second. To do more requests, we need to spawn extra threads or processes for the sendfile job. Actually, we will end up creating extra contexts just for waiting on disk.

A high-performance web server, such as nginx, is written as an event dispatcher which puts all of its sockets and file descriptors into non-blocking mode and then polls them. As a descriptor becomes available for a read or write, the web server sends or reads data and goes to the next one. This works well when system calls are fast. With modern demands, the original sendfile became a bottleneck, particularly in regard to number of connections and throughput.

This was a well-known problem even a decade ago. FreeBSD used a special flag, `SF_NODISKIO`, to tell sendfile to avoid reading from disk if data is not cached, and to return a special error code immediately. The web server then pre-cached the data with a call to `aio_read(4)` and retried the sendfile. Although there is a lot of extra action in place, and there is no guarantee that data will wait in cache for sendfile, this approach worked much better than letting the original sendfile block on disk.

We decided to make it even better. The idea is that sendfile will not wait for disk to read the data and will return immediately. This means that the web server doesn't stall for several milliseconds and can go forward working with the next descriptor. In short, that is the whole idea, but the implementation is not as simple as the description.

Q Were substantial changes required to the FreeBSD kernel or any of its subsystems to implement the new sendfile(2)? Did you come across any unexpected bugs during the implementation?

A Yes, the changes required were substantial. First, we needed to create an asynchronous interface in the kernel to read the data. The original `sendfile(2)` talked to disk similar to a `read(2) syscall`, using the `VOP_READ` filesystem operation. This was already wrong, since the interface is designed to copy out data to userland, which in principle is what sendfile is meant to avoid. The original sendfile pre-wired

pages that correspond to the data read, then ran `VOP_READ` with copy out to nowhere. As a side effect, that made pages paged in, and thus ready to be sent to a socket. Thus, we decided not to delve into the `VOP_READ_ASYNC` interface, but to instead use `VOP_GETPAGES`, which is designed exactly for bringing individual pages into memory. We implemented `VOP_GETPAGES_ASYNC` and built the new sendfile around it.

Second, substantial change affected socket buffers as we put data into a socket, but the data isn't ready yet. Since the pages aren't yet read from a disk, we can't send them. But they must occupy their place in the socket to preserve correct sequencing of data in a socket. When accounting for socket buffer limits, we also must account for that data. This all required bringing a notion of not ready data in socket buffers and functions to write it and to later activate it.

Speaking of bugs: of course, during experiments we found a lot of them. Who doesn't? An interesting one was an arithmetic bug in the `vnode_pager_haspage()` that was inherited from pre-FreeBSD times, when it suggested that it can read beyond the end of a file. We were the first to extensively use this function, which is why we found the bug after 20 years. It was fixed in FreeBSD `commit r282426`.

Q The new sendfile(2) was a joint effort between NGINX Inc. and Netflix. Why was FreeBSD chosen as the reference implementation and are there plans to port this system call to other operating systems such as Linux?

A FreeBSD is used in the heart of Netflix OpenConnect CDN[1], which streams data to Netflix customers all over the world, being the world's largest source of traffic on the Internet. So we were not building a reference implementation of the new sendfile just for fun or as a thought experiment; rather, we were improving the OpenConnect software, so that a single server can serve more data. This resulted in improving FreeBSD itself.

There are no plans for porting this to other operating systems on our side. We are focused on FreeBSD.

Q As a developer at Netflix, you have the unique opportunity to work with a CDN designed to push massive amounts of Internet traffic using FreeBSD. Were other FreeBSD improvements needed to manage the scale required by Netflix, and does Netflix upstream most of its improvements for the benefit of the FreeBSD community?

A Yes, we have made a lot of extra changes to FreeBSD to serve our traffic. And yes, we are trying to upstream all of them. The process isn't easy though. The quality standards for open-source code are actually higher than for production code. In production, you care only about the architecture you are running on, and you don't care about others. You care only about your workload, and you care only about those parts of an operating system that you use. To upstream something, we need to address all other platforms, workloads, and possible users of APIs and standards. At the same time, adopting our code to meet open-source standards must not degrade performance for our case. Sometimes satisfying both internal and open-source demands at the same time appears difficult.

Nevertheless, the process of upstreaming our improvements goes on. Follow commits from emax@, gallatin@, glebius@, imp@, lstewart@, rrs@, and scottl@ to see what is going on for FreeBSD 11 and 12.

Q Are you working on any other interesting projects?

A Right now, I don't have any big projects in my queue, but there is a lot of interesting stuff from my colleagues at Netflix.

For example, there is `SSL_sendfile()`, built on top of the new sendfile. The idea is that once a TLS session is negotiated, the web server gives the session key to the kernel, and then can use sendfile on a TLS socket. The implementation requires two stages of asynchronous data processing: first, disk read, and second, encryption. Only then is the data in a socket activated.

There is also an I/O scheduler and massive improvements to the VM subsystem, to TCP, to storage drivers, and to NIC drivers. These topics actually deserve separate articles, so I won't go into detail and leave that to their respective authors. ●

[1] https://www.nginx.com/blog/why-netflix-chose-nginx-as-the-heart-of-its-cdn/

Dru Lavigne is a Director of the FreeBSD Foundation and Chair of the BSD Certification Group.