by Mariusz Zaborski

FreeBSD for Developers

When someone mentions FreeBSD, you probably think about your servers and NAS. Without a doubt, FreeBSD is a great operating system for those purposes. However, there are some myths out there that FreeBSD isn't good for developers and that if you want to do software development on or of FreeBSD, you should get a Mac. The truth is that FreeBSD is a complete operating system and everyone from system administrators to software developers to accountants can use it for daily jobs. In this article let's look into FreeBSD as software developers.

Languages

The most important question for software developers is whether their language is supported by FreeBSD. Well, all the popular languages are supported by FreeBSD. There is no limit, really. Do you want to do your shiny new web in Node.js, PHP, or Python? FreeBSD supports all of them. If you are looking more for low-level programming like Go or Rust, FreeBSD has it. Are you classier than that? Then you can find a new version of clang and gcc in the ports. You can even work with Java through OpenJDK or C# through Mono.

What is great is that in many cases the native FreeBSD package manager pkg(1) permits you to install the required compiler. In the case of PHP, Ruby, or Python, many *packages* are also provided by *pkg*. If you want to install pygame for Python, you can simply type:

```
# pkg install py{27,36}-game
```

This allows you to manage all the packages you have installed in your operating system. It's easy and clean. If, for some reason, you don't like this approach, you can still use pip as always. It is the same with WordPress. If you run the command below you'll be ready to go:

```
# pkg install wordpress
```

The full list of packages with language compilers and interpreters can be found by executing:

```
# pkg search lang/
```

It's also great that all packages are up-to-date. The repology project (https://repology.org/repositories/statistics/newest) is doing an analysis of a huge number of package repositories and other sources comparing package versions across them. The conclusion is that FreeBSD has one of the newest ports. Its packages are far more updated, for example, than the newest version of Ubuntu. If you are looking for an up-to-date language infrastructure, FreeBSD is your choice.

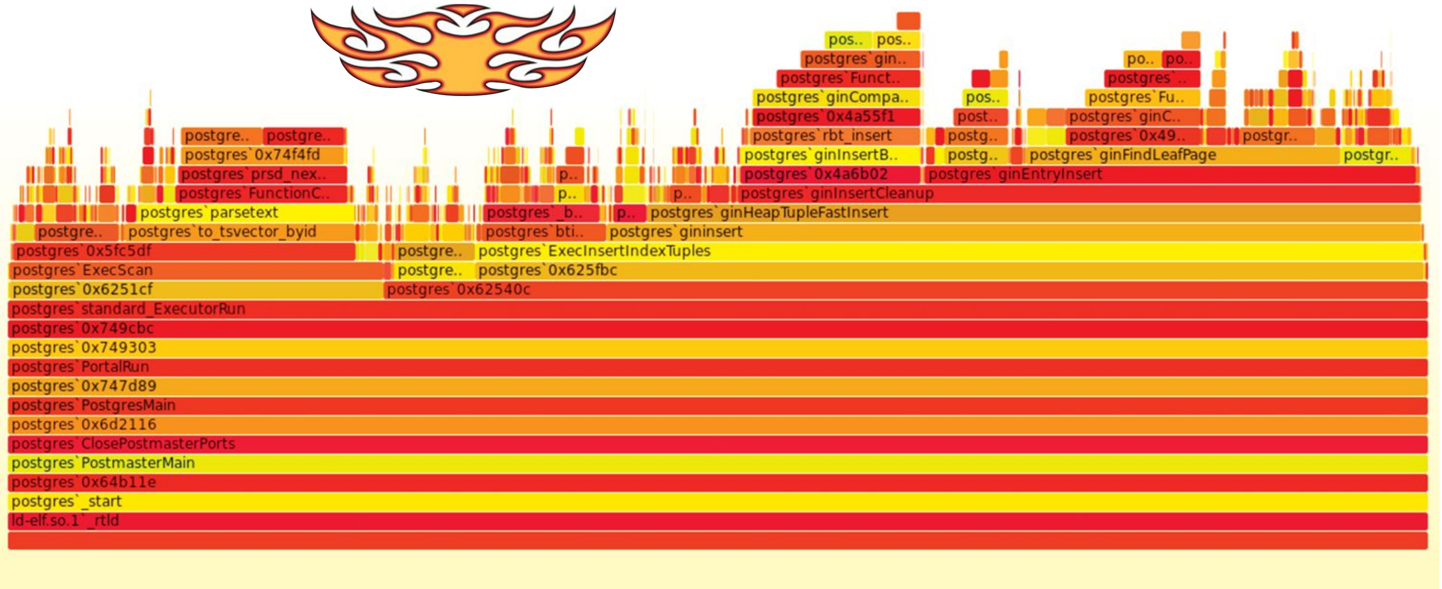
Developer's Environment

Another important question for software developers is whether their favorite IDE will work on FreeBSD. When we talk about a developer environment, you probably associate Unix with vim and emacs. That's true, and you can use them with FreeBSD, although you will also find more modern IDEs for your work like:

- Eclipse
- Sublime Text
- Pycharm (freeware and commercial version)
- IntelliJ (ultimate and community)

If your IDE does not support FreeBSD directly, you can always try to run them using Linuxulator, a Linux compatibility layer in FreeBSD. You also don't need to worry about your code version control system. You will find all popular, open-source VCS like git, mercurial, or svn. You also may use commercial ones like perforce.

There is no better way to maintain your envi-



Example of a flame graph generated from PostgreSQL, generated using output from DTrace one liner:

```
# dtrace -x ustackframes=100 -n ‘profile-5000 { ustack(); }` -p PID -o output
```

ronment and test your software than jails. If you want to have a few versions of your application for different customers or multiple databases and so on, then jails are what you want to use. You can also combine them with ZFS and easily upstream your changes to production. These days, managing jails is easier than ever by using *iocage* or *ezjail*.

Sometimes jails are not enough and we have to work with different operating systems. FreeBSD also has this covered. If you are looking for lightweight virtualization of any modern operating system like Windows, Linux, OpenBSD, NetBSD or FreeBSD *bhyve* is the way to go.

Contain Directory Madness

Where do you install your software in Linux: */usr/bin, /bin, /usr/local/bin*? Or maybe *bin* is a simple symlink to */usr/bin* and everything is a total mess? The administrative binaries always land in the *sbin*; the normal binaries land in *bin*. Also, the hierarchy of directories is clean and understandable:

- */bin* and */sbin* are the FreeBSD-specific binaries shipped with the operating system that are required for minimal work with the system,
- */usr/bin* and */usr/sbin* are the FreeBSD-specific binaries shipped with operating systems, and
- */usr/local/bin* and */usr/local/sbin* are directories where all your third-party software will land.

Additionally, you can read about the whole filesystem hierarchy on the manpage **here(7).** Again, FreeBSD wins with its simple and clean solution.

Look into Your Program

One of the best tools for developers, which is lacking in other operating systems, is DTrace. We don't have space to describe all functionalities of DTrace but once I started using DTrace, I don't know how I coped without it. DTrace is a dynamic tracing framework and was originally developed by Sun Microsystems. Using a few lines of the D script, we are able to see a current stack of the program, analyze the performance, trace the input of the functions, and much, much more.

The real power of DTrace is in the few scripts written by Brendan Gregg (https://github.com/brendangregg/FlameGraph) that allow you to generate flame graphs. Flame graphs are a visualization of traced software allowing the most frequent code paths to be identified. You can use these to trace many things, starting with your software, FreeBSD kernel, and ending with third parties. Everything that you need is a binary with some symbols. To use it, you do not need to stop your program or recompile it. We recently used DTrace on the PostgreSQL database to trace why the insert was taking so much time. Thanks to DTrace, we easily found the problem with table settings and were able to fix it. This is a big advantage over other operating systems for software developers, and DTrace should be in your toolbox.

Other tools that may be useful for you when developing low-level applications, and which are FreeBSD-specific, are:

- ***procstat*** – a powerful tool that allows you to get a lot of process information
- ***ktrace/kdump*** – an alternative for *strace(1)*, which allows you to print a list of called syscalls
- ***pmccontrol*** – control hardware performance monitoring counters
- ***pmcstat*** – performance measurement with performance monitoring hardware

In the case of low-level debugging by default, you will not find a gdb in the base system any more, but you can use a great alternative lldb from the LLVM project, which we encourage you to give a try.

Some Great APIs That the World Needs to Catch Up With

FreeBSD is a pioneer in many technologies. It was the first operating system to catch up with ZFS—which took the Linux world ages. FreeBSD first saw the potential in containers a long time before docker—developing *chroot* and *jails*. FreeBSD is POSIX compatible, but it also introduces or integrates many great APIs that are not available in many popular operating systems.

All operating systems implement poll and select, which have problems with scalability. We also see epool, which, instead of solving problems, introduces some. FreeBSD took a different direction and introduced *kqueue(1)*. *kqueue* provides efficient input and output event pipelines between the kernel and userland. *kqueue* is much more efficient, especially when polling for events on a large number of file descriptors.

In talking about descriptors, we should also mention process descriptors—a new way to add a handle (descriptor) to the process. Process identifiers (PID) used by many operating systems are not reliable. Between checking the status of process and sending signals, many things can occur in the operating system, and, in theory, the PID may be used by another process. Process descriptors solve those problems by giving you a reliable handle to add to the process. If the process is terminated, you still will have a handle to inform you about it.

And there is the FreeBSD sandboxing technique called Capsicum. Capsicum is a lightweight OS capability and sandbox framework. Capability-based security means that processes can only perform actions that have no global impact. Processes cannot open files by their absolute path or cannot open network connections. The idea is to protect your application with the process privilege separation in mind.

To secure your applications, you may also consider using CloudABI (https://github.com/NuxiNL/cloudabi). CloudABI takes a POSIX function and adds capability-based security and removes everything that's incompatible with that. This forces software developers to use very specific sets of functions in their applications but increases the security of the application.

FreeBSD is working on many interesting technologies that in the future may be standard in other operating systems. If you want to be up-to-date you should look into this OS.

Documentation

FreeBSD is known to have great documentation. In the case of software development, this is no different. How many times have you googled for an ASCII table? In FreeBSD, by default, you have a manpage for it: **ascii(7)**. The same goes for architecture/language-specific things like:

- **arch(7)** – architecture-specific details like size of the pointer, numbers, or pages.
- **operator(7)** – C and C++ operator precedence and order of evaluation.
- **zstyle(9)** – the best C style you will find – FreeBSD has used it for decades.
- **hier(7)** – to understand Unix filesystems layout.

The Sky's the Limit

FreeBSD has a lot of interesting features like jails, bhyve, ZFS, and DTrace that make life easier for software developers. You also will find very useful documentation in the system. This is not to mention the large amount of third party software that is just waiting to be used. One of the great ways of learning new languages is to create some games in it—and, of course, you can do this on FreeBSD! So, don't wait a moment longer—configure your new developer box on FreeBSD! ●

MARIUSZ ZABORSKI is a QA&Dev manager at Fudo Security. He has been the proud owner of the FreeBSD commit bit since 2015. Mariusz's main areas of interest are OS security and low-level programming. At Fudo Security, Mariusz leads a team that is developing the most advanced solution for monitoring, recording, and controlling traffic in IT infrastructure. In 2018, Mariusz organized the Polish BSD User Group. In his free time, he enjoys blogging at https://oshogbo.vexillium.org.