FreeBSD and Google Summer of Code 2025

BY JOE MINGRONE

The successful completion of Google Summer of Code (GSoC) 2025 marks FreeBSD's 21st consecutive year participating in the program. Three factors made this year stand out. First, we received 64 applications, which is more than double last year's total and roughly four times the number received in 2023. AI tools likely contributed to the surge, producing some low-quality submissions; however, the overall quality of most applications remained high. Second, we saw a notable increase in interest from South Asia. Given that the FreeBSD Community Survey results indicate that 85% of respondents were from Europe or North America, this interest from Asia and other parts of the world is welcome. Third, the number and quality of accepted projects were encouraging. Out of 1200 total GSoC projects across 185 participating organizations, FreeBSD's 12 accepted projects nearly doubled the average per organization, and unlike last year, all 12 were successfully completed.

Before we discuss individual projects, let's reflect on why we participate in GSoC. The program requires significant effort, from organizing the application process and defining project ideas to mentoring contributors. Is this investment of time and resources, which could otherwise go toward direct development, worthwhile? From a short-term technical perspective, it's debatable; while some projects lead to committed code, many do not. However, considered from a long-term perspective, the answer is clearer. GSoC is playing an important role in attracting and developing new contributors. Since 2022, five new FreeBSD committers have come through GSoC, and one 2017 participant went on to serve on the 12th Core Team.

GSoC 2025 Projects

Sockstat UI Improvements

For those unfamiliar with **`sockstat(1)`**, it is a command to list open Internet or Unix domain sockets. Damin Rido's goal for this project was to enhance the flexibility of the command's output by allowing dynamically sized columns and integrating libxo for structured output support. All three of Damin's pull requests linked below were pushed to the **`src`** tree.

- Add automatic column sizing and remove -w option: freebsd/freebsd-src#1720
- Reintroduce **`-w`** flag to automatically size the columns: freebsd/freebsd-src#1746
- Add libxo support: freebsd/freebsd-src#1770

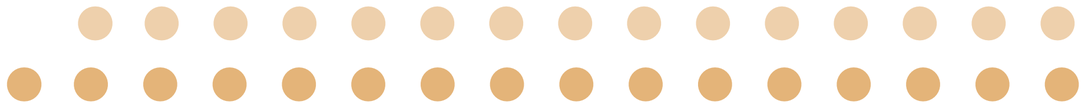
`vmm(4)` Accelerator Support for QEMU

The VMM (Virtual Machine Monitor) module is the kernel component of the **`bhyve(4)`** hypervisor, accessible through **`vmm(4)`**. Other than bhyve, another widely used machine emulator and virtualizer with official support for FreeBSD is QEMU. However, prior to this project, QEMU on FreeBSD could only make use of software emulation (via its Tiny Code Generator) because it lacked support for hardware-accelerated virtualization. In other words,

QEMU on FreeBSD could not take advantage of the host CPU's virtualization extensions to run guest code directly on the hardware. This limitation resulted in significantly higher CPU overhead and slower guest performance compared to hardware-assisted virtualization.

The primary objective of Abhinav Chavali's GSoC 2025 project was to integrate VMM acceleration support into QEMU on FreeBSD. He accomplished this by modifying QEMU's memory management layer to interoperate with VMM's kernel-allocated guest memory. He also adapted VMM to make certain non-critical devices such as the HPET and RTC optional, allowing QEMU to emulate them in user space instead. This enables a hybrid interrupt model (with the virtual LAPIC handled in the kernel and the IOAPIC emulated in user space), which has the potential to deliver performance levels comparable to bhyve under FreeBSD.

The latest from from Abhinav is that he was able to successfully boot FreeBSD 14 under QEMU with the VMM acceleration. You can view his code here.



The latest from Abhinav is that he successfully booted FreeBSD 14 under QEMU with VMM acceleration.

Testing and Development for Rust FreeBSD Device Drivers

Over the past few years, there has been interest in incorporating Rust in both FreeBSD and Linux development. For a sample of some discussions, refer to an RFC for Rust support that was posted to the Linux Kernel mailing and to discussions on the FreeBSD hackers list. In both communities, debates have emerged: some opponents warned of issues such as "doubling build times", whereas proponents argued that Rust would make certain tools easier or even possible to implement. Beyond discussions, tangible progress has been made. In his Master's thesis project, Johannes Lundberg created a Rust KPI and network driver, while David Young created a simple "Hello World" FreeBSD kernel module in Rust and summarized existing community efforts to adopt Rust.

The *Testing and Development for Rust FreeBSD Device Drivers* project builds on past efforts to incorporate Rust into FreeBSD development. One of its primary goals was to create a testing and continuous-integration framework for Rust kernel modules. Aaron gives an overview of his Rust echo driver in this video and summarizes the project in this write-up. His code is available here:

- https://github.com/Acesp25/rustdrv
- https://github.com/Acesp25/freebsd-kernel-module-rust
- https://github.com/Acesp25/RustKLD

Full-Disk Administration Tool for FreeBSD

Prior to this GSoC 2025 project from Braulio Rivas, FreeBSD lacked a user-friendly tool for full-disk administration, i.e., a utility comparable to Linux's GParted for partitioning, resizing, moving, and managing file systems. The goal of this project was to fill that gap by creating a new partitioning tool called geomman. Upon completion of the project, geomman supports the following operations:

- copy and paste partitions on the same disk or across disks
- grow UFS, NTFS, ext2, ext3 and ext4 filesystems
- shrink NTFS, ext2, ext3 and ext4 filesystems

- visually select free space to place a partition
- create exFAT, NTFS, ext2, ext3 and ext4 filesystems
- check filesystems: fsck_ufs (UFS), fsck_msdos (FAT), fsck.exfat (exFAT), ntfsfix (NTFS), and e2fsck (ext)
- create and label a partition
- create and encrypt a partition

Remaining work:

- ZFS support
- resolve issues when moving a partition
- test cases

The upstream repository can be found at: https://gitlab.com/brauliorivas/geomman.

Adding QCOW2 Compressed Image Support to mkimg

QCOW2 (QEMU Copy-On-Write version 2) is a widely used disk image format for virtualization, recognized for features such as thin provisioning and built-in compression. FreeBSD's `mkimg(1)` tool can create disk images in a variety of formats, including QCOW2. Until now, however, mkimg's QCOW2 support was limited and did not allow for the creation of compressed QCOW2 images.

This summer, Christos Komis enhanced mkimg by completing these milestones:

- add support for QCOW2 v2 compressed images
- add support for QCOW2 v3 compressed and uncompressed images
- update the user interface to expose the new features
- extend the test suite to verify correctness
- update the man pages with the new functionality
- perform code refactoring to improve readability and maintainability.

The implementation has been thoroughly tested and is ready for commit. Users can now generate compressed QCOW2 images directly with mkimg, simplifying workflows for virtual machine image generation and reducing reliance on external conversion tools. Check out Christos's code at https://github.com/ckkomis/freebsd-src/commits/mkimg/qcow2-compression/.

ACPI Initialization in Loader With Lua Bindings

Kayla Powell's project extends the ACPICA library's initialization into the FreeBSD loader, ensuring the full ACPI namespace is available before the kernel is loaded. The work replaces a somewhat ad hoc bootloader approach to ACPI by invoking standard ACPICA routines (e.g., AcpiInitializeSubsystem, AcpiLoadTables, AcpiWalkNamespace, AcpiEvaluateObject) within the loader. This gives consistent discovery and interrogation of ACPI objects early in the boot process. To maintain the loader's lightweight design, only the necessary ACPICA components were ported, omitting many functions unnecessary for initialization or scripting.

On top of the foundational layer, the project introduces Lua bindings that expose the ACPI namespace and object-evaluation facilities to scripts running under the loader. In other words, we can now write Lua loader code to walk the ACPI tree, examine device-table entries, and attach or read data from ACPI nodes, before the kernel loads. Along with the implementation, unit and regression tests were included (e.g., comparing namespace dumps between C and Lua and building the loader across architectures).

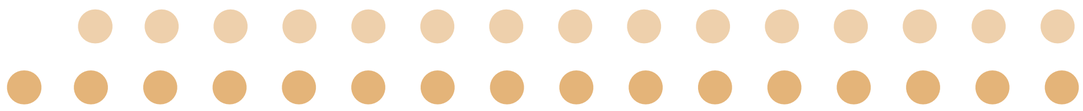
Refer to Kayla's summary of the project on her blog at https://kmpow.com/content/gsoc-writeup and her pull requests: 1818, 1819, and 1843.

`mac_do(4)` and `mdo(1)` Improvements

Kushagra Srivastava's project aims to enhance FreeBSD's credential transition infrastructure by improving both the kernel-side MAC module, **`mac_do(4)`**, and its companion userland tool, **`mdo(1)`**. Rather than relying on traditional setuid executables (which carry inherent risks), the goal is to enable controlled, fine-grained credential transitions under the umbrella of FreeBSD's MAC framework. On the kernel side, **`mac_do(4)`** was extended to support per-jail configuration of authorized executables (so that an admin can specify exactly which binaries in a given jail are allowed to request credential transitions, instead of being limited to a hardcoded path). Also, it now intercepts standard credential-changing syscalls such as **`setuid(2)`**, **`setgid(2)`**, **`setgroups(2)`**, and treats them as full transition requests that are subject to the **`mac_do(4)`** policy module.

For userland work, the **`mdo(1)`** tool was improved to provide fine-grained credential transition requests. It now supports explicit overrides of user/group IDs, as well as supplementary groups, via flags such as **`-g`**, **`-G`**, and **`-s`**. It also includes a **`--print-rule`** option to display the matching **`mac_do(4)`** rule for a requested transition, which helps administrators with rule creation and debugging.

Together, these enhancements make credential transitioning more flexible, secure, and integrated with FreeBSD's jail and MAC frameworks. This reduces the need for risky setuid binaries and brings improved auditability and control.



These enhancements make credential transitioning more flexible, secure, and integrated with FreeBSD's jail and MAC frameworks.

Refer to Kushagra's project write-up at https://thesynthax.hashnode.dev/my-google-summer-of-code-journey-part-3 for details.

Speed up the FreeBSD Boot Process

Lahiru Gunathilake's project is a continuation of past projects to speed up FreeBSD's boot time by profiling the boot sequence, identifying bottlenecks, and implementing optimizations. Using the built-in TSLOG tracing framework, Lahiru generated flame charts of the boot path to understand where time was being spent and where unnecessary delays could be eliminated.

Once the profiling revealed hotspots such as device attach phases, initialization of large filesystem subsystems (especially ZFS), and sleeps in vfs_mountroot (root filesystem mount), the work progressed to the implementation phase. This included:

- reducing a benchmarking buffer size from 16MB to 256KB, cutting startup from 989 ms to 67 ms
- reducing long wait loops in keyboard and mouse initialization, and introducing the tunable hw.atkbd.short_delay
- eliminated unnecessary waits for USB devices.

Overall, Lahiru reported reductions of 8.2 s in kernel initialization, 3.5 s after the ZFS and input device optimizations, and 1.9 s when skipping the USB boot wait.

WiFi Management UI

Muhammad Saheed took on a project to develop cohesive CLI (wutil) and TUI (wutui) utilities for managing WiFi networks on FreeBSD. The aim was to cover "station mode op-

erations, such as scanning, connecting/disconnecting from wireless networks," and to wrap these into a clearer, more consistent user interface. Other completed work includes:

- updating related man pages
- creating a port for wutil
- adding libwpa_client build option to security/wpa_supplicant port
- creating a port for libifconfig
- extracting required ifconfig helpers into libifconfig (D52130, D52131)

Refer to Muhammad's blog for more information about his work.

Journaling for FreeBSD ExtFS

This project by Pau Sum set out to bring Linux-compatible journaling to FreeBSD's ext2fs filesystem implementation. With FreeBSD's existing ext2fs driver, FreeBSD users could already mount and use ext2/3/4 filesystems, but the driver lacked journaling support, meaning unclean shutdowns required lengthy recovery via fsck. Pau's work introduced on-disk journal awareness and transaction logging to improve crash recovery and filesystem integrity, allowing FreeBSD to mount and replay journals on ext3/4 volumes and interoperate more closely with Linux systems.

Rather than replicating Linux's full journaling framework, the design implements a traditional metadata-only journal using the same on-disk structures for compatibility. The new code defines key data structures, including ext2fs_journal (representing the active journal), ext2fs_journal_transaction (grouping atomic metadata updates), and ext2fs_journal_buf (tracking per-block state). Core filesystem operations like ext2_link, ext2_mkdir, and ext2_write were extended with journal hooks to begin, dirty, and end transactions. Committing a transaction writes descriptor, metadata, revoke, and commit blocks, followed by checkpointing to flush updates to disk. Recovery proceeds in three passes: validating transaction ranges, collecting revoked blocks, and replaying non-revoked metadata.

By the end of the project, 11 of 12 journal hooks were complete, with work in progress on truncation and extent-based operations. Planned extensions include journaling support for extents and truncation, checksum validation for journal integrity, more extensive crash simulation, and documentation cleanup. The implementation was tested using fsx and dirconc. Pau's code is available from his fork of FreeBSD's src repository.

Power Profiling Tool

The goal of Kasyap Jannabhatla's project was to provide granular, process-level insights into power usage on FreeBSD. This addressed the limitations of ACPI's whole-system power statistics. Inspired by Performance Co-Pilot (PCP) and RAPL (Running Average Power Limit) support, the project implemented a FreeBSD-native framework rather than porting Linux PowerTop. The solution consisted of a kernel-level component to collect power-related metrics and a user-space daemon with a command-line interface that provides CPU usage and energy consumption per process tracking. By combining RAPL readings with per-process CPU utilization derived from kvm_getprocs, the tool can estimate energy usage for individual processes and threads, providing a foundation for fine-grained power profiling and future enhancements in FreeBSD's power management ecosystem.

Throughout the development, the project focused on building a lightweight daemon-based architecture, implementing a library (librapl) for structured RAPL data access, and integrating it with process accounting to calculate per-thread energy consumption. Testing involved stress benchmarks such as OpenSSL Speed and careful handling of multi-

core runtime accounting using thread IDs. By the end of the project, the framework could reliably report per-process energy usage over time, and all deliverables, including the daemon, library, and documentation, were completed. The implementation is available here.

Port FreeBSD to QEMU microvm

QEMU microvm is a minimalist virtual machine inspired by Firecracker. While FreeBSD was ported to Firecracker, that platform is Linux-specific, limiting portability. For this project, Wyatt Geckle aimed to develop a QEMU microvm version of the FreeBSD kernel inspired by the Firecracker port. To do this, Wyatt replicated prior porting attempts and analyzed kernel initialization issues, particularly timer configuration, which caused long boot times. By studying NetBSD's working microvm implementation and Intel documentation, the project identified differences in FreeBSD's timecounter and APIC initialization.

The current FreeBSD microvm kernel boots under QEMU microvm but does not call tc_init, which limits the available timers. The Firecracker port remains broken due to MPTables issues, requiring further investigation. Despite these limitations, the project broadened Wyatt's understanding of FreeBSD and NetBSD kernel internals, virtualization, and microvm platforms, and produced extensive documentation for building, running, and debugging FreeBSD in microvms and Firecracker. The work also provides a foundation for future contributions to FreeBSD, QEMU microvm, and Firecracker support, as well as reproducible debugging workflows for other microvm projects. Those interested in this work can find more information on Wyatt's blog. The code can be found at Wyatt's fork of the src tree.

It's gratifying to review the success of our GSoC 2025 program, but the time before GSoC 2026 starts will come quickly.

Mentor Summit

Robert Clausecker, a FreeBSD GSoC co-administrator and mentor, represented FreeBSD at this year's Mentor Summit that was held from October 23 to 25 in Munich, Germany. Topics discussed included AI-generated and spam applications. While no definitive solutions have emerged, one approach under consideration is to require applicants to meet with potential mentors before applying. This is something FreeBSD has already encouraged in previous years to help ensure good matches between mentors, contributors, and projects. Robert also met with representatives from the Linux Foundation to brainstorm potential collaboration between the Linux and FreeBSD Foundations, such as attracting more students to operating systems development.

Other summit sessions covered funding in open source projects. Robert spoke with a developer working on RTEMS, a real-time operating system used in various devices and learned that they incorporate FreeBSD's network stack in their system. He also met GSoC organizer Stephanie Taylor and shared the positive impact GSoC has had on FreeBSD. Of course, he returned with some swag, including a T-shirt and a pair of GSoC(k)s.

Final Thoughts

It's gratifying to review the success of our GSoC 2025 program, but the time before GSoC 2026 starts will come quickly. As usual, our most significant challenges to repeating this year's success will be developing suitable projects, finding dedicated mentors, and matching applicants to mentors. Fortunately, recent changes to the Google Summer of Code program should help.

- Flexible Timelines and Scope: Project timelines are more flexible. Contributors and mentors can choose from small (90-hour), medium (175-hour), or large (350-hour) projects, and the total time for the projects can be extended from the standard 8 weeks (small) or 12 weeks (medium and large).
- Expanded Contributor Pool: The pool of applicants has grown. Contributors do not have to be university students, so everyone new to open source is eligible to participate.

But for now, let's bask in our collective success and acknowledge the considerable effort that went into this year's program.

Thank you

Thank you to everyone who contributed project ideas to https://wiki.freebsd.org/SummerOfCodeIdeas and especially to our 2025 mentors:

- John Baldwin
- Olivier Certner
- Robert Clausecker
- Pedro Giffuni
- Tom Jones
- Warner Losh
- Ed Maste
- Getz Mikalsen
- Joe Mingrone
- Mehdi Mokhtari
- George Neville-Neil
- Colin Percival
- Alfonso Sabato Siciliano
- Alan Somers
- Toomas Soome
- Fedor Uporov
- Aymeric Wibo

JOE MINGRONE is a FreeBSD ports developer and works for the FreeBSD Foundation. He lives with his wife and two cats in Dartmouth, Nova Scotia, Canada.