FreeBSD® JOURNAL

March/April 2023

Issue topic: **Embedded**

GPU PASSTHROUGH

CHERIBSD PORTS AND PACKAGES

CONFERENCE REPORT: SCALE20X

PRACTICAL PORTS: LET'S CHAT, GPT

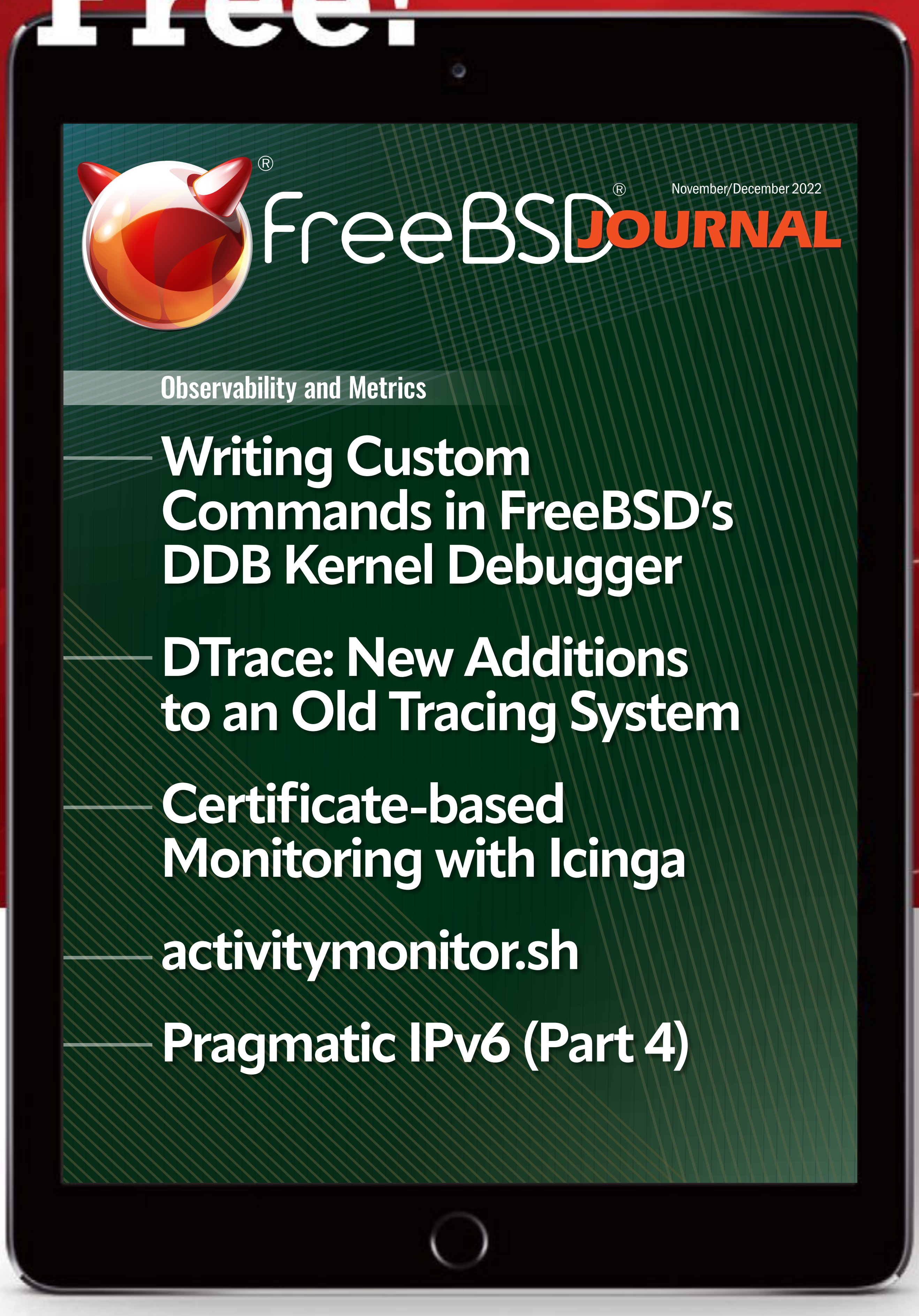
FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!



FreeBSD® JOURNAL

November/December 2022

Observability and Metrics

Writing Custom Commands in FreeBSD's DDB Kernel Debugger

DTrace: New Additions to an Old Tracing System

Certificate-based Monitoring with Icinga

activitymonitor.sh

Pragmatic IPv6 (Part 4)

2023 Editorial Calendar

- Building a FreeBSD Web Server (January-February)
- Embedded (March-April)
- FreeBSD at 30 (May-June)
- Containers and Cloud (Virtualization) (July-August)
- FreeBSD 14 (September-October)
- To be decided (November-December)

Find out more at: freebsdfoundation/journal

FreeBSD® JOURNAL

Editorial Board

John Baldwin • Member of the FreeBSD Core Team and Chair of FreeBSD Journal Editorial Board

Tom Jones • FreeBSD Developer, Internet Engineer and Researcher at the University of Aberdeen

Ed Maste • Senior Director of Technology, FreeBSD Foundation and Member of the FreeBSD Core Team

Benedict Reuschling • FreeBSD Documentation Committer and Member of the FreeBSD Core Team

Mariusz Zaborski • FreeBSD Developer

Advisory Board

Anne Dickison • Marketing Director, FreeBSD Foundation

Justin Gibbs • Founder of the FreeBSD Foundation, President and Treasurer of the FreeBSD Foundation Board

Allan Jude • CTO at Klara Inc., the global FreeBSD Professional Services and Support company

Dru Lavigne • Author of *BSD Hacks* and The Best of FreeBSD Basics

Michael W Lucas • Author of more than 40 books including *Absolute FreeBSD*, the *FreeBSD Mastery* series, and *git commit murder*

Kirk McKusick • Lead author of *The Design and Implementation* book series

George Neville-Neil • Past President of the FreeBSD Foundation Board, and co-author of *The Design and Implementation of the FreeBSD Operating System*

Hiroki Sato • Director of the FreeBSD Foundation Board, Chair of AsiaBSDCon, and Assistant Professor at Tokyo Institute of Technology

Robert N. M. Watson • Director of the FreeBSD Foundation Board, Founder of the TrustedBSD Project, and University Senior Lecturer at the University of Cambridge

---

S&W PUBLISHING LLC
PO BOX 3757 CHAPEL HILL, NC 27515-3757

Editor-at-Large • James Maurer
maurer.jim@gmail.com

Design & Production • Reuter & Associates

FreeBSD Journal (ISBN: 978-0-61 5-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-51 42 • fax: 720/222-2350
email: info@freebsdfoundation.org

Copyright © 2023 by FreeBSD Foundation. All rights reserved.
This magazine may not be reproduced in whole or in part without written permission from the publisher.

LETTER from the Foundation

Dear Readers,

Welcome to the March/April issue of the *FreeBSD Journal*! I've been thinking about how many issues we've published since the first issue came out back in January 2014. I believe this is the 56th issue of helpful and informative content covering many different topics within the world of FreeBSD. The fact that we have had so many issues focused on different areas over these many years, shows how complex, powerful, and innovative FreeBSD is.

This current issue focuses on a subject near and dear to my heart, embedded systems. I started out as an embedded systems engineer writing firmware for storage device companies, and the technology has grown and improved since I began my career. The amount of information that can be stored on one storage device just blows my mind! More and more, things are becoming embedded to create smaller systems, and FreeBSD plays a significant role in these products. From display systems in stadiums to cool video games, FreeBSD is there controlling the systems.

Please, sit back and check out the articles and columns in the following pages! As usual, you'll also find information on upcoming BSD-related events—I hope to see many of you at the upcoming BSDCan in Ottawa in May!

Oh, and I have one ask. If you've enjoyed and learned from some or all of the 56 issues published to date, please consider making a donation to the FreeBSD Foundation, so we can continue funding the *FreeBSD Journal* and our other projects. Thank you!

Deb Goodkin
Executive Director
FreeBSD Foundation

Donate to the FreeBSD Foundation
freebsdfoundation.org/donate/

FreeBSD® JOURNAL

March/April 2023

Issue topic: Embedded

5 **GPU Passthrough**
By Corvin Köhne and Daniel Kerkhoff

12 **CheriBSD Ports and Packages**
By Konrad Witaszczyk

23 **Conference Report: SCaLE20X**
By Drew Gurkowski

25 **Practical Ports: Let's chat, GPT**
By Benedict Reuschling

3 **Foundation Letter**
By Deb Goodkin

35 **We Get Letters**
By Michael W Lucas

38 **Events Calendar**
By Anne Dickison

GPU Passthrough

BY CORVIN KÖHNE AND DANIEL KERKHOFF

Beckhoff has been working on industrial automation technology since it was founded in 1980. In 1986, the concept of PC-based machine control was born. Back then, Beckhoff was moving away from microcontroller-based PLCs and started using standard PC technology as a base for the PLC, while casing it in industrial housing. This boosted performance massively, as PCs were already far more powerful than microcontrollers.

With the rise of Windows, the idea to make a general-purpose operating system part of the machine control system itself was born. The Windows environment was already familiar to machine builders and operators. The Windows environment could also be enriched with additional user applications, which proved beneficial since it added value to the overall control system. Windows itself, however, did not provide the required execution environment for control logic with hard, real-time constraints.

TwinCAT (which stands for The Windows Control Automation Technology) software had been developed and was first released in 1996. TwinCAT was designed as an extension of Windows NT to assure deterministic execution of control tasks on Windows-based control systems from Beckhoff.

Over the years, the hardware components of PCs, such as CPUs, memory, bus systems and I/O devices have become more powerful, as have PC-based control systems. In 2002, Beckhoff invented the embedded PC family, a DIN-rail mountable, industrial PC with a direct connection to the Beckhoff I/O system. As these devices used low power CPUs (AMD SC2200) with compact flash cards, an embedded operating system with a small footprint was needed. At the time, Windows CE was the dominant player on the market and was the obvious choice for this embedded PC family. Today, Beckhoff offers a broad product family of industrial PCs, ranging from small single core ARM CPUs to the latest AMD Ryzen CPUs and Intel Xeon based server class devices. Consequently, control system engineers can choose the right performance class for their applications.

TwinCAT runtime is the common denominator for all Beckhoff IPCs, and naturally, TwinCAT itself has evolved over time to enable control systems to harness the power of modern PC hardware. Windows has become the standard operating system for PC-based control systems at Beckhoff and also for its customers.

In recent years, however, Windows CE has become obsolete and customer demand for a non-Windows based control system has increased, so Beckhoff started to search for an alternative operating system which could become the new TwinCAT host. Ultimately, FreeBSD was chosen as the non-Windows operating system to host TwinCAT. Beckhoff combined FreeBSD and TwinCAT to create a new PC-based control operating system, named TwinCAT/BSD.

TwinCAT/BSD

Currently, FreeBSD 13.2 serves as the foundation for TwinCAT/BSD. Consequently, TwinCAT/BSD benefits from the FreeBSD base system, including FreeBSD user space tools as well as kernel services. In addition, Beckhoff has also imported the majority of TwinCAT features from Windows-based control systems to TwinCAT/BSD. TwinCAT/BSD is made up of both FreeBSD base system and the TwinCAT features. Figure 1 shows the structure of the system.

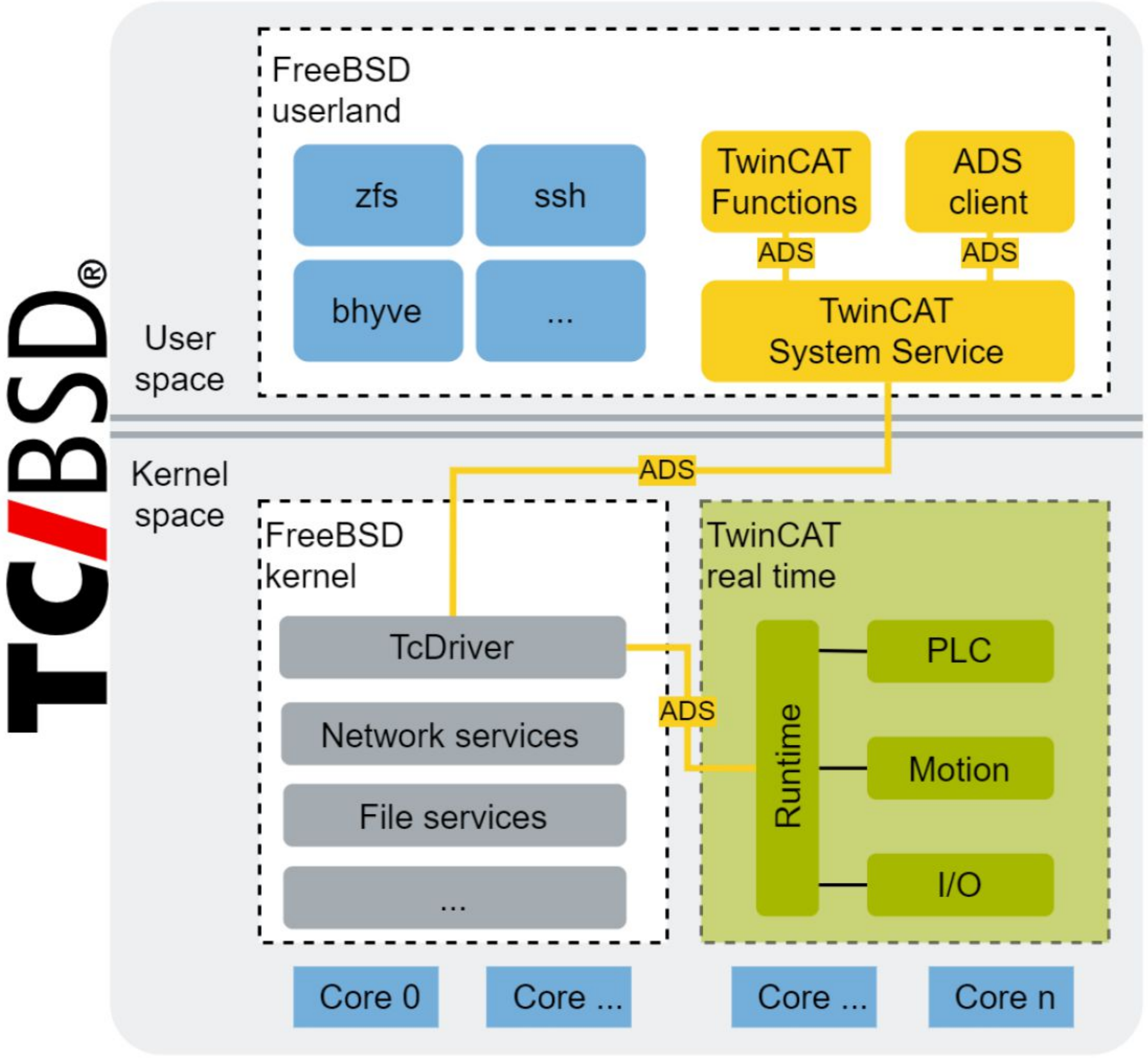


Figure 1: TwinCAT integration on FreeBSD

The core components of a TwinCAT-based control system are the TwinCAT runtime and the TwinCAT System Service including the ADS message router. The TwinCAT runtime is responsible for the execution of real-time control tasks. Real-time control tasks typically involve programmable logic controller (PLC) tasks, motion tasks such as axis positioning or CNC, and the integration of industrial I/O devices connected via fieldbus systems.

To be able to run control tasks with hard real-time requirements, TwinCAT configures different system settings. For example, TwinCAT uses isolated CPU cores for control tasks. Isolated cores are removed from all CPU groups of the FreeBSD scheduler. This allows TwinCAT to fully utilize isolated cores to execute control tasks. The TwinCAT runtime determines the execution of the configured, real-time tasks. Each real-time task has its own cycle time which specifies the frequency at which the task has to be executed.

The TwinCAT runtime provides an interface for the Automation Device Specification protocol (or ADS for short) so that the TwinCAT runtime can send and receive ADS messages. ADS is an application protocol developed by Beckhoff. The whole information exchange in a TwinCAT system is driven by ADS. The TcDriver is a standard FreeBSD kernel module which provides an ADS channel between the TwinCAT runtime and the user space. This channel is used by the TwinCAT System Service to control the TwinCAT runtime. Likewise, the TwinCAT System Service implements an ADS message router to route ADS messages between different applications like the TwinCAT runtime and TwinCAT Functions. Thanks to the ADS message router, the TwinCAT system can easily be extended with other services that implement the ADS protocol. Similarly, ADS client applications only need access to the message router to access ADS services such as the TwinCAT runtime to read and write the PLC task's data, for example. The ADS specification and libraries are publicly available so that ADS-based applications can be implemented easily.

In general, the TwinCAT architecture described for TwinCAT/BSD is similar to the architecture for Windows-based operating systems. However, Beckhoff customers particularly favor TwinCAT/BSD as it is a more compact and robust operating system. Due to its small footprint, it is often used on embedded industrial PCs. These embedded industrial PCs can be found in control applications ranging from building automation to CNC machines to automated guided vehicles. Although TwinCAT/BSD is already used in a variety of control applications, sophisticated control applications always require more flexibility to extend the capabilities of the control systems.

The Need for Virtual Machines

For PC-based control systems, the host operating system serves two purposes. As with any operating system, it is responsible for providing a clean, abstract interface to the subordinate hardware. Equally, the operating system environment can be used to run applications and services which extend the functionality of the control system. The user can thus install or develop their own applications for the control system while interacting with the system as they do with any other PC.

Furthermore, the operating systems serve as hosts for the TwinCAT runtime. FreeBSD, as a host operating system, is responsible for managing the majority of the hardware in an appropriate manner. This allows TwinCAT to focus on scheduling and executing real-time control tasks. However, without a running host operating system, the TwinCAT runtime would not be able to run at all. In terms of the overall control system, the system operator needs to pay a great deal of attention when interacting with the host operating system, since a restart or crash in the host operating system would also result in the real-time control being stopped. This is true for both Windows and TwinCAT/BSD PC-based control systems.

bhyve offers the option of using a virtual machine as user environment, which can be rebooted without affecting the TwinCAT real-time system. In this setup, the TwinCAT/BSD host operating system would still serve as the underlying PC-based control system, but the operator would no longer directly use the host operating system for human-machine interaction; a guest operating system would be used inside a virtual machine environment instead.

> FreeBSD, as a host operating system, is responsible for managing the majority of the hardware in an appropriate manner.

With virtual machines, the system designer can take advantage of a wider range of operating systems to deploy additional services on the control system. A virtualized Windows environment could enable the Windows desktop environment and already developed Windows user interfaces to be used for machine interactions on TwinCAT/BSD systems. Environments such as these are especially useful for customers who are used to Windows-based control systems. A virtualized Linux environment would, in turn, enable containerized Linux applications to be used on the IPC via Docker, Podman or Kubernetes. Again, customers who are already using Linux container deployments to run services or algorithms for analyzing process data could deploy those applications directly on the control system itself.

Virtual machines hosted by bhyve also benefit from OpenZFS, which is the default file system on TwinCAT/BSD. For customers, high availability of both the TwinCAT-based control system itself and user applications is crucial. As a result, any updates on a system which may lead to downtime or inoperability are inherently high-risk. Being able to create ZFS snapshots of datasets which show the functional state of the base system and of virtual machine disks enables a functional system state to be restored, thus cutting the risk of long downtimes if software updates malfunction.

Furthermore, running operating systems in virtual machines can also be harnessed to improve system security via increased availability. The capacity to access process data from a control system is becoming more relevant as a result of digitization, because this data forms the basis for optimized production processes. Acquisition, processing, and interpretation of process data should be as automated, in-process and timely as possible. However, the access to the control system which is required by applications and/or operators also increases the risk that the control system's open interfaces may be exploited or operated incorrectly. If the system is accessed in this way, this can in turn influence the correct operation of the control system and, in the worst case, lead to downtime and thus to uncontrolled processes. A virtual machine can be used as an additional gateway here to restrict both user and network access to the control system. To provide the aforementioned benefits through virtual machine environments, we have started incorporating bhyve into TwinCAT/BSD.

Virtual Machine Configuration

bhyve was integrated into TwinCAT/BSD in order to configure a system setup like the one shown in figure 2.

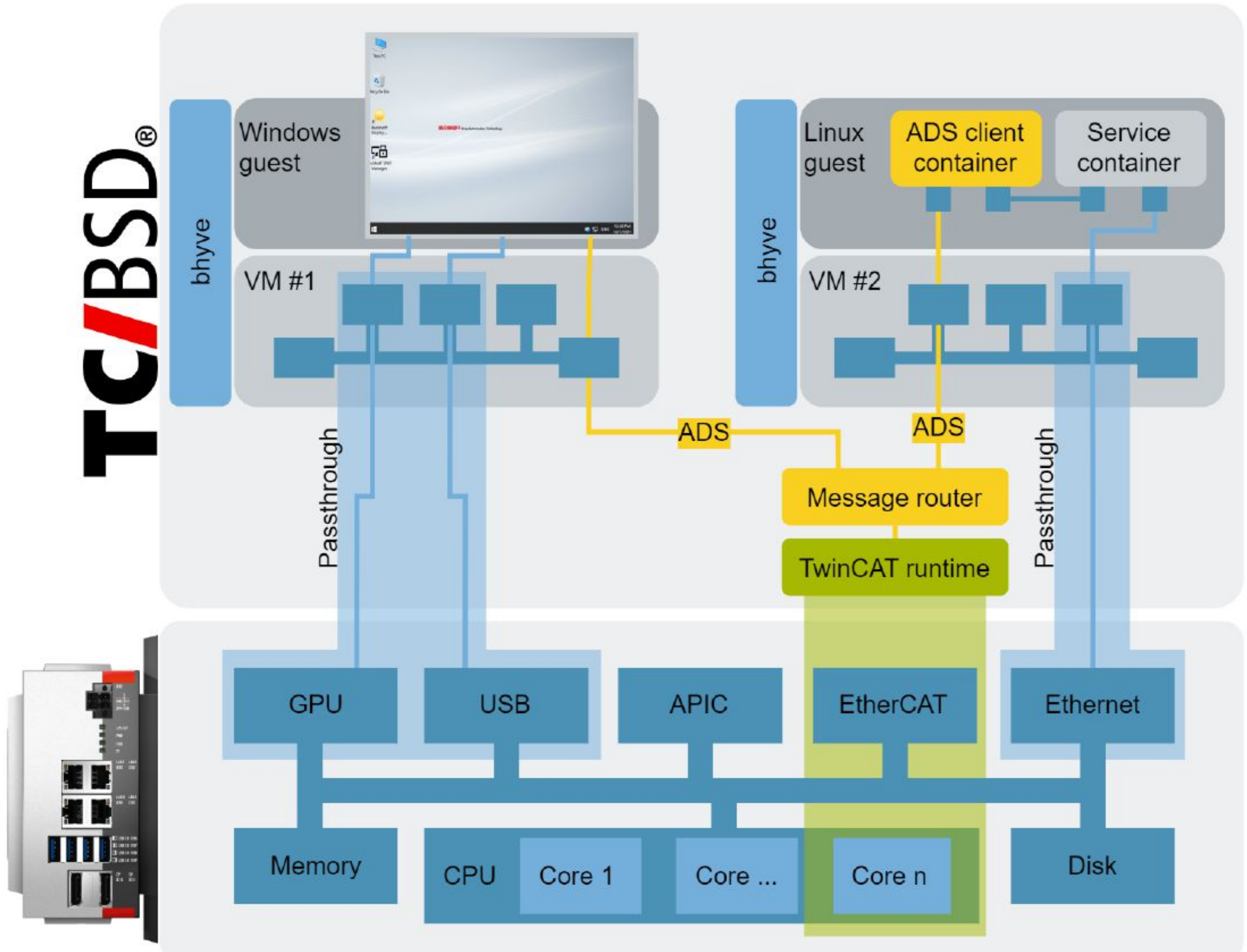


Figure 2: Sample VM configurations on TwinCAT/BSD

In this setup, the execution of virtual machines should be optional as not every TwinCAT/BSD user requires separate execution environments for additional guest systems. If needed, however, the customer would be able to configure a virtual machine with bhyve that could support either the use case of a Windows or Linux guest operating system or both.

Linux guest operating systems are usually needed for headless server applications either installed directly on the Linux guest OS or deployed and managed as Linux containers. In terms of system setup, this means that network interface configuration tasks such as routing and package filtering are handled on host and guest operating sites. In most cases, this type of VM configuration consists of two network interfaces. One of these is used in a host-only network to access the TwinCAT ADS message router on the host operating system.

Process data from the TwinCAT control system can be gathered with access to the ADS message router. These data can then be further analyzed by other service containers inside the guest operating system. The information gained is then usually served or transmitted via a second network interface which is explicitly assigned to the virtual machine.

On industrial PCs that contain modern Intel or AMD CPUs, hardware support is provided for IO virtualization. This is termed VT-d for Intel and AMD-Vi for AMD. Both forms of support are similar and allow a hypervisor to give a guest direct access to a hardware device. The device memory can be mapped onto the guest memory space. Additionally, interrupts can be remapped to be directly delivered to the guest. This allows a guest to access a device without any hypervisor intervention to increase the performance.

bhyve already supports VT-d and AMD-Vi. These are used for PCI passthrough. As a result, a PCI device can be exclusively assigned to a single guest. This device is no longer shared with the host and the guest has direct access to it.

> All network traffic is directly passed to the virtual machine environment, meaning that no additional filter or routing rules need be applied on the TwinCAT/BSD host.

With PCI passthrough, one Ethernet controller can be isolated from the TwinCAT/BSD host and explicitly assigned to the virtual machine. As a result, all network traffic is directly passed to the virtual machine environment, meaning that no additional filter or routing rules need be applied on the TwinCAT/BSD host. Likewise, network traffic can be processed much faster as it is directly available inside the guest.

As mentioned above, virtual machines with Windows guest operating systems should serve as an isolated environment for human-machine interaction. Consequently, the physical I/O interfaces of the industrial PC must be passed to the virtual machine. On Beckhoff IPCs, these interfaces are the USB controller for user input via keyboard, mouse or touch panel and, of course, the graphics controller for video output on any connected display.

As usual, PCI passthrough is used to explicitly assign the USB controller to the Windows guest. PCI passthrough works for almost all PCI devices. However, graphics cards are one class of PCI devices in which PCI passthrough isn't supported by bhyve. As this is a common issue in hypervisors, it's often referred to as GPU passthrough.

GPU Passthrough

Most graphics cards have some special requirements which aren't handled by bhyve yet. These special requirements are dependent on the graphics card vendor and the graphics driver. Different operating system graphics drivers require different features within the graphics card. Some graphics drivers need a VBIOS while others need access to special memory regions. All of these elements require extra handling, which has to be implemented for bhyve. Beckhoff has developed patches for bhyve to enable GPU passthrough on AMD and integrated Intel graphics cards. We're also working on upstreaming all of these patches.

GPU passthrough of AMD graphics cards requires a PCI ROM emulation. In the first step, the VBIOS of the graphics card has to be extracted from the host system. It is then used by the guest BIOS to initialize the graphics card. Additionally, some guest graphics drivers require the VBIOS.

The VBIOS can be extracted using a variety of different methods. There's no common method for extracting the VBIOS on FreeBSD yet. Therefore, a different operating system such as Linux must be booted to extract the VBIOS. This ensures that the right VBIOS version is used for GPU passthrough. If a different version is used, there could be incompatibilities which may damage your device in the worst-case scenario.

Once the VBIOS has been extracted, it has to be passed to the guest. Therefore, we added a PCI ROM emulation to bhyve so that the guest can read the PCI ROM to get the VBIOS. The patches required to support the PCI ROM emulation have already been upstreamed. The PCI ROM emulation will be included in FreeBSD versions 13.2, 14.0 and subsequent versions.

Sadly, supporting the PCI ROM emulation isn't enough to support a VBIOS. It has to be executed and shadowed into main memory. This should be done by the guest BIOS. However, bhyve's current edk2 port is not capable of this. The patches that would make this possible have been placed in the upstream but have not yet been accepted. Therefore, a modified guest BIOS must be used to fully support GPU passthrough for AMD devices.

The VBIOS can be extracted using a variety of different methods. There's no common method for extracting the VBIOS on FreeBSD yet.

Adding a VBIOS to graphics cards also provides another benefit. A VBIOS includes an UEFI graphics driver. When the guest executes the VBIOS, the VBIOS installs an UEFI graphics card driver. This will be picked up by the guest BIOS to produce graphic output. It makes the graphics card available in an early boot phase. For that reason, graphic output can be produced in the UEFI and bootloader phase when a VBIOS is passed to the guest. Without a VBIOS, the first graphic output is produced after the OS driver is loaded.

In contrast to AMD graphics cards, GPU passthrough for integrated Intel graphics cards has not yet been upstreamed. Intel graphics cards have two special memory regions which need special attention. Intel calls these memory regions OpRegion and Graphics Stolen Memory. They have to be reserved by the guest BIOS and reported in a PCI register. Therefore, bhyve needs to detect and report those regions to the guest. We have patched bhyve to provide an E820 table to the guest. It includes all memory regions and reports where the OpRegion and Graphics Stolen Memory reside in the memory. Our guest BIOS is modified to parse this E820 table and reserve all memory regions as reported by the table. bhyve's PCI emulation reports the addresses of these memory regions by PCI register.

Our current implementation works for Intel processors from the 3rd generation, called Ivy Bridge, up to 9th generation Intel processors, called Coffee Lake Refresh. Newer Intel processors require slight modifications for emulation. Support will be added in the future. We have not yet tested GPU passthrough for AMD graphics cards on a variety of different graphics cards. According to some feedback from bhyve users who have tested the patches, it is supported on a wide range of different AMD graphics cards.

As previously stated, GPU passthrough is the same as PCI passthrough. Due to this, it does not differ from PCI passthrough in terms of usage. bhyve's -s option can be used for GPU passthrough as shown in the following code snippet.

```
bhyve \
       -s 0,hostbridge \
       -s 2,passthru,0/2/0 \
       -s 31,lpc \
       -l bootrom,/usr/local/share/uefi-firmware/BHYVE_UEFI.fd \ my-vm
```

If you wish to pass a VBIOS to the guest, we've added a ROM option to bhyve's -s option.

```
       -s 2,passthru,0/2/0,rom=/home/user/vbios.rom
```

Conclusion

FreeBSD, in combination with Beckhoff TwinCAT automation software, forms the foundation of the TwinCAT/BSD operating system. TwinCAT/BSD is Beckhoff's Unix alternative to Windows-based control systems and is already used in a variety of control systems.

Now, by integrating bhyve into TwinCAT/BSD, PC-based control systems benefit from virtual machine environments. With the added GPU passthrough feature, virtual machine graphic output can now be displayed on connected screens. This is particularly useful for applications which provide a human-machine interface. Since these applications often originate from former Windows-based control systems, they can now be used on TwinCAT/BSD.

Virtual machines, in combination with the PCI passthrough feature, also provide an advanced system setup in which user interfaces and network interfaces can be isolated from the host operating system. Consequently, virtual machine environments can provide an additional layer to improve the security of the control system.

CORVIN KÖHNE is a software developer at Beckhoff Automation GmbH & Co. KG. He's a maintainer and developer of Beckhoff's FreeBSD fork, which is called TwinCAT/BSD. He focuses on x86-based systems and hypervisor technology. Due to his contributions to the bhyve project, he became a FreeBSD committer in 2022.

DANIEL KERKHOFF started his career as an application engineer for industrial automation solutions at Beckhoff Automation GmbH & Co. KG. After graduating with a master's degree in information technologies, he joined the TwinCAT product management team. In his role, he is responsible for queries from customer who use bhyve and TwinCAT/BSD as part of Beckhoff's PC-based control systems.

CheriBSD Ports and Packages

Pure-capability third-party software for Arm Morello and CHERI-RISC-V CheriBSD

BY KONRAD WITASZCZYK

CHERI is a hardware/software/semantics co-design project that aims to improve the security of existing and future hardware-software stack implementations. Recent studies from Google and Microsoft show that around 70% of vulnerabilities in their products relate to memory-safety issues. CHERI not only allows us to prevent most such vulnerabilities from being exploited but also to compartmentalise software and thus limit the impact of successfully exploited vulnerabilities currently unknown to software maintainers (e.g., backdoors in third-party software dependencies).

Until 2022, the CHERI project was mostly developed by the University of Cambridge, SRI International, and their partners, including Microsoft, Google, and Arm. The CHERI ecosystem rapidly expanded with the release of the Arm Morello platform, the first public hardware implementation of CHERI. In January 2022, Arm started shipping the first (out of roughly a thousand) Morello boards to companies, academic and government institutions. To provide a user-friendly work environment for Morello users, *CheriBSD* — a FreeBSD-based operating system adapted for Arm Morello and CHERI-RISC-V — needed an infrastructure to build and ship CHERI-adapted third-party software before Morello was released. Today, dozens of universities, government research labs, and companies are using CheriBSD in their work on Morello, and rely on this infrastructure daily.

> The CHERI ecosystem rapidly expanded with the release of the Arm Morello platform.

This article describes our journey of building third-party software packages for CheriBSD without CHERI-enabled hardware, using QEMU user mode, FreeBSD ports, and Poudriere. While discussing implementation details of the package building infrastructure, the article summarises what decisions and changes we needed to make to finally achieve ~24,000 AArch64 packages and ~9,000 CHERI-enabled packages.

CHERI Hardware-software Stack

In order to fully understand the infrastructure for CheriBSD package building, we should first describe the SDK that a developer can use to build software for CHERI. The CHERI hardware-software stack (see Table 1) consists of hardware, emulators, compilers, debuggers, operating systems and applications for CHERI-enabled operating systems. Each component of this stack needed to be adapted for CHERI and must implement support for CHERI capabilities[22].

Third-party software	~9,000 CHERI packages (Morello) ~24,000 non-CHERI packages (Morello)
Operating systems	CheriBSD (Morello, CHERI-RISC-V) FreeRTOS (CHERI-RISC-V) CHERIoT RTOS (CHERI-RISC-V) Linux (Morello) Android (Morello)
Toolchains	CHERI LLVM for CHERI C/C++ (Morello, CHERI-RISC-V) Morello GCC for CHERI C/C++ (Morello) GDB-CHERI (Morello, CHERI-RISC-V)
CPUs	Arm Morello SoC CHERI-RISC-V on FPGA QEMU-CHERI (Morello, CHERI-RISC-V) Microsoft CHERIoT (CHERI-RISC-V)

Table 1: Current CHERI hardware-software stack

Before the Arm Morello platform[20] was released, CheriBSD and third-party software had been developed and ported using QEMU emulators for Morello and CHERI-RISC-V[9]. This environment is still useful today to work on multiple CheriBSD branches or to attach the GDB debugger to QEMU and step through the CheriBSD kernel. Anyone interested in our research can try the CHERI exercises[25] to explore under QEMU how CHERI prevents memory-safety issues. A Morello QEMU-based VM can be created on FreeBSD, Linux and macOS with the *cheribuild* utility[1] using one simple command that fetches and compiles required software, and runs the VM:

```
$ ./cheribuild.py --include-dependencies run-morello-purecap
```

Available toolchains include LLVM compilers[14,17] and GDB debuggers[15]. LLVM can cross-compile code or compile it natively on hardware or under QEMU. GDB-CHERI, currently based on GDB 12, can disassemble capability-aware instructions and print information on register and in-memory capabilities. While this article focuses on CheriBSD[3], Arm also develops Linux and Android operating systems[18] with CHERI LLVM and GCC compilers for Morello[19]. In February 2023, Microsoft also published the CHERIoT project[16] that implements a full hardware-software stack with the CHERIoT RTOS for embedded RISC-V devices.

Having the above SDK, we decided to fork FreeBSD ports and extend them with bug fixes and changes necessary for CHERI and CheriBSD. We call this ports collection *CheriBSD ports*.

The process of porting software to CHERI is similar to porting code developed for 32-bit architectures to 64-bit architectures. A pure-capability program can only use CHERI capabilities and CHERI-aware CPU instructions to access memory. A pointer in such a program has its size increased to 128 bits to hold a CHERI capability. In order to compile a C/C++ program, the code must be adapted to the CHERI C/C++ semantics[24] that require the use of appropriate data types to store pointers (e.g., uintptr_t instead of long), and increase the alignment of pointers to 16 bytes. CHERI LLVM can identify many incompatibilities between C/C++ and CHERI/C++, and display detailed warnings suggesting what changes should be applied in code to make it compatible with CHERI. In many cases, a developer can successfully compile and run their software after fixing all issues found by CHERI LLVM. However, extensive testing is recommended to make sure that ported software does not include any run-time bugs (e.g., misaligned allocations in a custom memory allocator).

While a lot of open-source projects have been ported to CHERI[23], many crucial applications still cannot be compiled to use CHERI capabilities. For example, web browsers are extremely complicated pieces of software requiring lots of dependencies. In order to provide a fully functional development platform, CheriBSD allows to run both CHERI-adapted applications and applications compiled for a baseline architecture of a CHERI-extended CPU (e.g., Armv8-A for Morello). Compatibility with existing software has been essential to the CHERI project to allow to incrementally adapt software for CHERI rather than require to reimplement an application from scratch. FreeBSD, as a baseline operating system for CheriBSD, enabled the implementation of run-time environments for legacy and CHERI-aware software. However, when it comes to providing third-party software for multiple run-time environments, there are still some challenges that CheriBSD inherited from FreeBSD.

Compatibility with existing software has been essential to the CHERI project to allow to incrementally adapt software for CHERI.

Multi-ABI support

FreeBSD includes a feature called compatibility layers that provides system call implementations for programs compiled for different ABIs than the native ABI. For example, an amd64 FreeBSD kernel with a compiled-in 32-bit compatibility layer (also known as *freebsd32*) can run a program compiled for i386. CheriBSD benefits from this feature and supports two ABIs relevant to CHERI: CheriABI also known as the pure-capability ABI (`MACHINE_ARCH` aarch64c and riscv64c) for programs that can only use CHERI capabilities to access memory, and the hybrid ABI (`MACHINE_ARCH` aarch64 and riscv64) for programs that can but do not have to use CHERI capabilities. The latter ABI is implemented by the pure-capability CheriBSD kernel with the *freebsd64* compatibility layer, similar to freebsd32.

Missing cross-ABI support

While the FreeBSD and CheriBSD kernels implement support for multiple ABIs, multi-ABI environments are not supported by FreeBSD ports and Poudriere. This is a major issue in the context of CHERI. Many ports require dependencies that have not been adapted for CHERI yet. For example, Meson and Ninja are commonly used build systems depending on Python. Since we do not have CheriABI Python at the moment, we cannot build these utilities for CheriABI to compile other ports. If FreeBSD ports and Poudriere supported compile-time cross-ABI dependencies, we could use hybrid ABI Meson and Ninja to build CheriABI packages that do not require them at run time. The *CheriBSD ports* section briefly explains how we managed to partially resolve this issue.

Package manager(s)

The `pkg(8)` package manager can only manage packages built for one ABI — by default the ABI of the base system (based on `uname(1)`). For example, i386 packages cannot be installed on an amd64 host alongside amd64 packages and be registered in a single package database (`pkg-register(8)`). Of course, it is possible to create a package with binaries and shared libraries compiled for i386 that is marked as created for amd64, just like FreeBSD does for Linux packages, but that would require creating two packages for the same port (for amd64 and i386), and does not reflect the actual ABI of packaged files at the package manager level. There are two important issues that would have to be resolved to better support such multi-ABI environment:

1. Two packages with the same pre-compiled port but for different ABIs must use distinct paths not to conflict with each other.
   For example, Git compiled for two different ABIs with the same local base path (e.g., `/usr/local`) would conflict on files installed within that path (e.g., `/usr/local/bin/git`).
2. A package for one ABI should be able to depend on a package for a different ABI.
   For example, Git depends on Perl because it includes multiple Perl scripts used by its subcommands (e.g., `git add -i`). Instead of using an interpreter compiled for the same ABI, it could use Perl available for another supported ABI.

As of today, we solved the first issue and decided to ignore the second one for CheriBSD.

Not to create conflicts between packages in CheriBSD, we place CheriABI and hybrid ABI packages in two separate locations. We build CheriBSD ports for CheriABI with `LOCALBASE` set to `/usr/local` and hybrid ABI packages with `LOCALBASE` set to `/usr/local64`. While the FreeBSD ports build system provides the localbase feature (in `Mk/Uses/localbase`), we discovered and fixed many ports that break this functionality, e.g. by hardcoding paths in their code; or not using the localbase feature at all.

Built packages are registered in two separate package repositories that can be managed with separate package managers: `pkg64c` for CheriABI packages, and `pkg64` for hybrid ABI packages. `pkg64c` and `pkg64` are programs compiled for the same ABI as packages they manage, they use separate package repository configuration directories, databases and caches. In short, the package managers are completely unaware of each other.

CheriBSD ABI version

By default, the `pkg(8)` package manager decides which package repository to use based on the `NT_FREEBSD_ABI_TAG` ELF note of `uname(1)`. The value of that note is used to construct a value of the ABI pkg variable that can be embedded in a package repository URL (see `pkg.conf(5)` and `/etc/pkg/FreeBSD.conf`). For example, the URL:

```
pkg+http://pkg.FreeBSD.org/${ABI}/latest
```

is expanded to:

```
pkg+http://pkg.FreeBSD.org/FreeBSD:14:amd64/latest
```

on an amd64 host running FreeBSD 14-CURRENT.

In contrast to FreeBSD, CheriBSD does not have any assumptions regarding ABI stability across its releases and branches. Instead, CheriBSD maintains the ABI counter `__CheriBSD_version` (set to the current date as it is bumped), similar to `__FreeBSD_version` and also in `sys/param.h`, that describes the current ABI version used by a CheriBSD branch. In result, two CheriBSD releases can use the same ABI version and two different CheriBSD branch revisions can use two distinct ABI versions.

This approach allows us to flexibly make changes in the CheriBSD development branch and provide package repositories to users using different revisions of this branch. As for releases, we do not make any changes that would break the ABI within a single release as it would heavily disrupt users' work environments and require recompiling all user code.

We extended the *csu* code in CheriBSD to include the `__CheriBSD_version` counter in the additional `NT_CHERIBSD_ABI_TAG` ELF note of each program compiled for a given branch. Instead of using `NT_FREEBSD_ABI_TAG`, `pkg64` and `pkg64c` use `NT_CHERIBSD_ABI_TAG` when building an URL to a package repository. For example, the URL:

```
pkg+http://pkg.CheriBSD.org/${ABI}
```

is expanded by `pkg64c` to:

```
pkg+http://pkg.CheriBSD.org/CheriBSD:20220828:aarch64c
```

and by `pkg64` to:

```
pkg+http://pkg.CheriBSD.org/CheriBSD:20220828:aarch64
```

on a Morello host running CheriBSD 22.12.

Package building

The CheriBSD/Morello package building infrastructure consists of a local machine starting a build, a FreeBSD/amd64 host building CheriABI packages using the QEMU user mode and a FreeBSD/arm64 host building hybrid ABI packages natively. The builders use the following software stack:

- QEMU BSD user mode for CheriABI programs[10];
- CheriBSD base system;
- CHERI LLVM toolchain;
- CheriBSD ports[6];
- Poudriere extended for CheriBSD[5, 7];
- Poudriere configuration files and helper scripts (e.g., `poudriere-remote.sh`)[8].

Figure 1 presents an overview of the above components. Upon a command from `poudriere-remote.sh`, the FreeBSD/amd64 and FreeBSD/arm64 hosts create Poudriere jails, ports trees, and build the ports trees in CheriBSD/aarch64c and CheriBSD/aarch64 jails, respectively. The CheriBSD/aarch64c jails execute programs compiled for CheriABI using the QEMU user mode while toolchain utilities compiled for the amd64 architecture are executed natively. Similarly, the CheriBSD/aarch64 jails execute all programs natively as they are compiled for arm64. There are currently no ports' hybrid ABI compile-time dependencies that partially use CHERI capabilities and must be executed during the building process; thus, no QEMU user mode is needed for the hybrid ABI packages. The following sections describe the building infrastructure components in more detail.

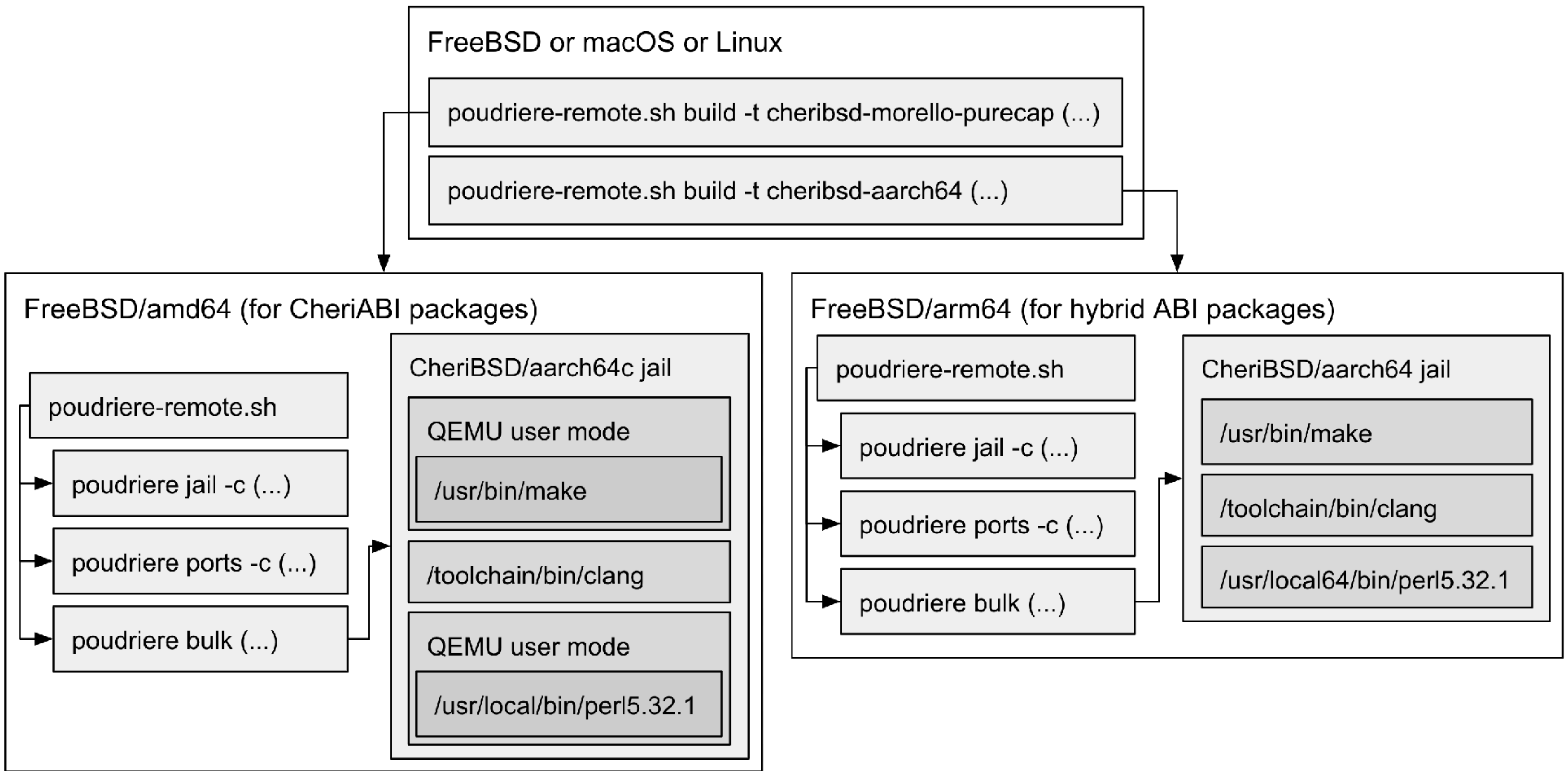


Figure 1: Package building process for CheriBSD

QEMU BSD user mode

Before the package building project started, QEMU-CHERI implemented only the QEMU system mode for CHERI-RISC-V and Arm Morello architectures. While the system mode allows developers to experiment with CheriBSD and cross-compiled third-party software, its performance is not sufficient to build large code scope projects because it emulates a full operating system with devices. Thankfully, Poudriere implements support for the QEMU user mode, on top of `binmiscctl(8)`. The user mode emulates user program instructions and executes system calls by translating them from their emulated user versions to native user versions, executing the translated system calls and translating results back to their emulated versions. Using this mode, we can run processes without unnecessary overhead related to system emulation. Most system calls in CheriBSD are compatible with FreeBSD and QEMU can handle any incompatibilities when translating them (e.g., when handling an `mmap(2)` call, an allocation might need to be padded with guard pages to make a returned CHERI capability representable[24]). However, we must make sure that a FreeBSD host running the user mode is not older than a baseline FreeBSD version used by the base system of an emulated CheriBSD branch.

Poudriere makes use of the user mode through the *imgact_binmisc* kernel module. FreeBSD allows defining binary image activators with `binmiscctl(8)` that execute binaries matching an ELF header pattern using a specific interpreter. For example, a system administrator can define an activator that runs an aarch64 binary using a QEMU user mode emulator on an amd64 host. In practice, a program executed within a FreeBSD/aarch64 jail on a FreeBSD/amd64 host is wrapped with the user mode, e.g.:

```
$ sh
```

is executed within the jail as the command:

```
$ /usr/local/bin/qemu-aarch64-static sh
```

where the `/usr/local/bin/qemu-aarch64-static` binary is compiled for the native ABI of the host and hence is natively executed rather than wrapped by an image activator again.

Our work on the QEMU user mode[10] started with support for CHERI-RISC-V. Unfortunately, the upstream QEMU repository included an outdated BSD user mode implementation — initially developed for FreeBSD/mips64 in 2015[11, 12], also in collaboration with the University of Cambridge and SRI International. Thanks to the *qemu-bsd-user project*[13] improving the BSD user mode support in QEMU, we had a baseline CHERI-RISC-V user mode. We rebased these changes onto QEMU-CHERI, and extended the implementation. Main modifications included:

1. Improved system call interface implementation.
   a. System call arguments and results used integer types rather than data types corresponding to `syscallarg_t` from FreeBSD. We modified the system call interface implementation to use appropriate machine-independent data types, allowing us to handle CHERI capabilities.
   b. We changed QEMU to match closer the CheriBSD/FreeBSD system call interface implementation and generated a system call table for QEMU using the `makesyscalls.lua` script from CheriBSD derived from FreeBSD.
2. New data types `abi_uintptr_t` and `abi_uintcap_t`.
   We used the data types in places where integer data types were incorrectly used, not matching the actual machine-dependent data types storing pointers and capabilities.
3. CheriABI support, including ELF loading code, stack, `mmap(2)` implementations, and adapting existing system calls for CHERI capabilities.
4. Machine-dependent changes for CHERI-RISC-V and Arm Morello, including CHERI capability permission bits and capability register access routines.

This part of the project took us the longest time. Currently, the user mode itself can easily be used with the cheribuild *qemu-cheri-bsd-user* branch[2]. For example, you can run a CheriABI shell from a CheriBSD/riscv64c base system on a FreeBSD/amd64 host using:

```
$ ./cheribuild.py run-user-shell-riscv64-purecap
```

CheriBSD ports

Besides CHERI/CheriBSD-specific patches for software included in the FreeBSD ports collection, we introduced additional `make(1)` variables to allow modifying port build configurations depending on an ABI they are built for and allow building CheriABI packages with hybrid ABI compile-time dependencies:

- `USE_PACKAGE_DEPENDS_REMOTE`;
  When `USE_PACKAGE_DEPENDS{, _ONLY}` is enabled, try to install a package from a remote repository instead of building a port from scratch, if a local package does not exist.
- `USE_PACKAGE_64_DEPENDS_ONLY`;
  Install dependencies marked with `USE_PKG64` using their replacement hybrid ABI packages with `pkg64` instead of building them from scratch.
- `USE_PKG64`;
  When `USE_PACKAGE_64_DEPENDS_ONLY` is set, use a hybrid ABI package for a port that cannot be built for CheriABI and is required by another port that is being built for CheriABI.
- `OPTIONS_{DEFINE,DEFAULT,EXCLUDE}_${ABI}`;
  Lists of options specific to `${ABI}`.
- `BROKEN_${ABI}`.
  When set, a port is believed to be broken for `${ABI}`.

We also modified autoreconf, cmake, meson, ninja and python support to allow us to specify custom commands for hybrid ABI build utilities with `<UTILITY>_CMD make(1)` variables, e.g. `CMAKE_CMD`.

Poudriere

Our Poudriere fork[7] supports package building on both FreeBSD and CheriBSD hosts. By default, it uses base system tarballs for an operating system it is executed on but a user can specify the OS with a new flag `-o` for **`poudriere-jail(8)`**. Since CheriBSD does not include a toolchain in its base system, Poudriere installs it using **`pkg`** or **`pkg64`** within a Poudriere jail, outside a local base directory not to conflict with a toolchain built from CheriBSD ports. There are two set configurations shipped with Poudriere: **`cheriabi`** and **`hybridabi`**. Both use the same toolchain but define different `LOCALBASE` values, and the **`cheriabi`** one enables hybrid ABI build utilities that are unavailable for CheriABI.

Building CheriABI packages for the development branch on a CheriBSD/Morello host requires executing three simple commands:

```
$ poudriere jail -c -j aarch64c-dev -a arm64.aarch64c -v dev
$ poudriere ports -c -p main
$ poudriere bulk -j aarch64c-dev -p main -z cheriabi -a
```

When porting software to CHERI, CheriBSD users also can benefit from Poudriere to easily bootstrap a build environment. This is especially useful for hybrid ABI software that sometimes requires setting custom shared library search paths not to use by mistake CheriABI libraries from default search paths. With a Poudriere hybrid ABI jail, a developer does not have to worry about possible linking with CheriABI libraries as such jail only includes hybrid ABI programs and libraries.

Poudriere configuration and scripts

The last piece of the infrastructure is the *poudriere-infrastructure* repository[8] including Poudriere configuration files and shell scripts to bootstrap a build environment on a remote host, sign a package repository and deploy it at *pkg.CheriBSD.org*. In particular, **`poudriere-remote.sh`** builds the CheriBSD base system, the SDK, the QEMU user mode (if needed) and starts package building with Poudriere. Poudriere logs of CheriBSD package builds are publicly available at *poudriere.CheriBSD.org*.

Results

As of March 2023, CheriBSD provides 9104 CheriABI packages and 24494 hybrid ABI packages[4,5]. There are only 37 CheriBSD ports with patches. Most of the changes specific to CHERI have been successfully upstreamed to third-party software repositories. Some of the patches include changes specific to CHERI restrictions (e.g., the stronger pointer alignment to 16 bytes) which shows that open-source communities consider CHERI and Arm Morello as a promising platform.

CheriBSD users can easily set up a Morello host using a memstick installer obtained from CheriBSD.org (see Figure 2). **`bsdinstall(8)`** in CheriBSD includes an installer step[21] where a user can decide if they want to install packages to run a CheriABI graphical environment (using KDE Plasma and Wayland; see Figure 3) and additional hybrid ABI programs (at the moment Firefox and Chromium). These packages can easily be installed with meta-packages:

```
$ pkg64c install cheri-desktop
$ pkg64 install cheri-desktop-hybrid-extras
```

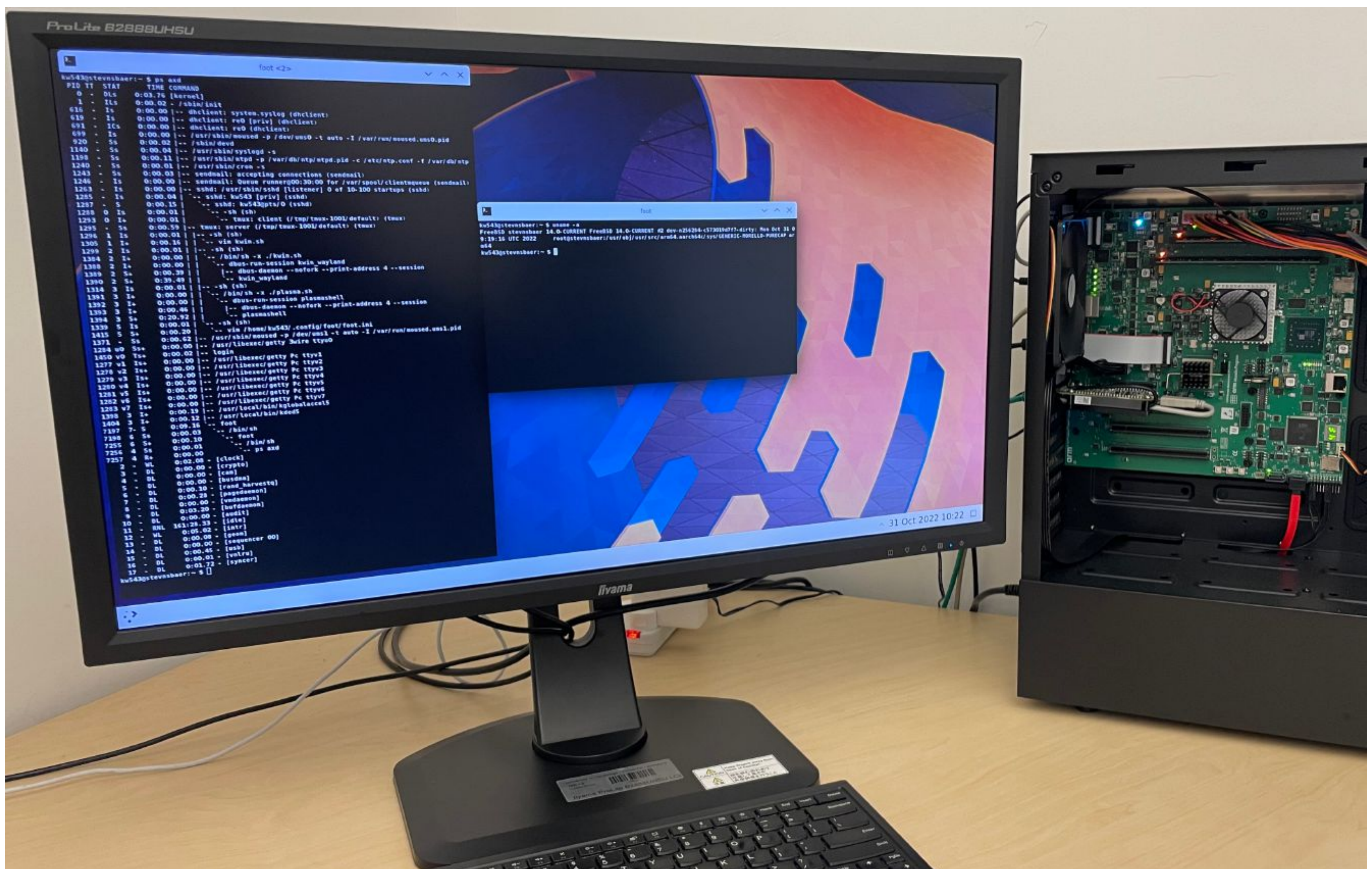
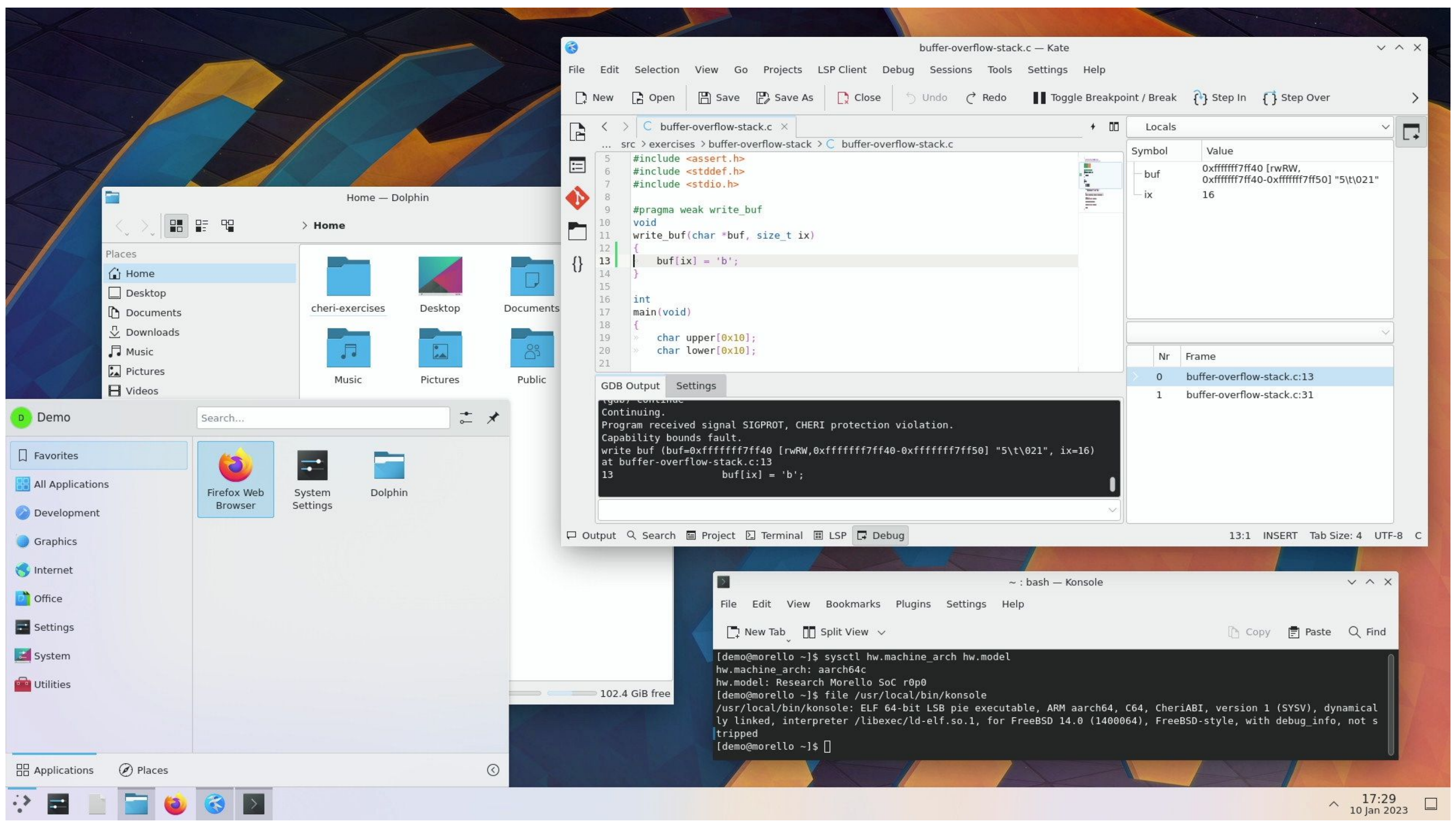


Figure 2: Arm Morello board running CheriBSD



buffer-overflow-stack.c — Kate
File Edit Selection View Go Projects LSP Client Debug Sessions Tools Settings Help
New Open Save Save As Close Undo Redo Toggle Breakpoint / Break Step In Step Over
buffer-overflow-stack.c
... src > exercises > buffer-overflow-stack > buffer-overflow-stack.c
Locals
GDB Output Settings
Output Search Project Terminal LSP Debug
13:1 INSERT Tab Size: 4 UTF-8 C
Home — Dolphin
Places
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
cheri-exercises
Desktop
Documents
Music
Pictures
Public
102.4 GiB free
Demo
Search...
Favorites
All Applications
Development
Graphics
Internet
Office
Settings
System
Utilities
Firefox Web Browser
System Settings
Dolphin
Applications
Places
~ : bash — Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab Split View Copy Paste Find
17:29
10 Jan 2023

```c
#include <assert.h>
#include <stddef.h>
#include <stdio.h>

#pragma weak write_buf
void
write_buf(char *buf, size_t ix)
{
	buf[ix] = 'b';
}

int
main(void)
{
	char upper[0x10];
	char lower[0x10];

```

Symbol	Value
buf	0xffffff7ff40 [rwRW, 0xffffff7ff40-0xffffff7ff50] "5\t\021"
ix	16

Nr	Frame
0	buffer-overflow-stack.c:13
1	buffer-overflow-stack.c:31

```
Continuing.
Program received signal SIGPROT, CHERI protection violation.
Capability bounds fault.
write_buf (buf=0xfffffff7ff40 [rwRW,0xfffffff7ff40-0xfffffff7ff50] "5\t\021", ix=16)
at buffer-overflow-stack.c:13
13                      buf[ix] = 'b';
```

```
[demo@morello ~]$ sysctl hw.machine_arch hw.model
hw.machine_arch: aarch64c
hw.model: Research Morello SoC r0p0
[demo@morello ~]$ file /usr/local/bin/konsole
/usr/local/bin/konsole: ELF 64-bit LSB pie executable, ARM aarch64, C64, CheriABI, version 1 (SYSV), dynamical
ly linked, interpreter /libexec/ld-elf.so.1, for FreeBSD 14.0 (1400064), FreeBSD-style, with debug_info, not s
tripped
[demo@morello ~]$ 
```

Figure 3: Memory-safe Morello desktop environment (CheriBSD, KDE Plasma, Wayland)[3]

CheriBSD releases and packages have been used by Technology Access Programme (TAP) participants. The Digital Security by Design (DSbD) initiative run by UK Research and Innovation organises TAP to allow UK-based companies experiment with the Arm Morello platform and prototype memory-safe projects. Currently, we collaborate with around 30 such companies. Thanks to the CheriBSD installer and pre-compiled third-party packages, TAP participants could easily deploy a work environment without the need to adapt for CheriABI, or even cross-compile, their software dependencies. However, many of them still needed to redesign their projects due to some missing third-party software, or port that software themselves.

Future work

Based on CheriBSD 22.05 and 22.12 releases, TAP, and CheriBSD users' experiences, we are planning next steps to increase the number of CHERI memory-safe packages. Currently, we are considering:

- CheriABI Python;

  As noted in the *Missing cross-ABI support* section, multiple build systems make use of

Python. Having Python adapted to CheriABI, we could not only build additional packages for CheriABI but also enable interesting research space in compartmentalising Python-based applications.

- Cross-ABI support in Poudriere;
  We would like to make use of hybrid ABI packages to build CheriABI packages with Poudriere. Currently, we build CheriABI packages using hybrid ABI build utilities by executing make package in port directories and transferring resulting packages to a package repository created with Poudriere. This feature would allow us to easily rebuild and deploy package repositories.
- Upstreaming patches;
  We would like to minimise the number of patches that must be maintained in CheriBSD ports, and instead commit them to upstream repositories. This includes changes in ports to better support custom local base paths in FreeBSD ports.
- CHERI-RISC-V packages.
  Currently, CheriBSD ships packages only for Arm Morello. Given that most of the applied patches are not specific to Morello, we should be able to build a large number of packages for CHERI-RISC-V as well. This would allow researchers to also evaluate the CHERI-RISC-V architecture against a large code corpus.

Conclusion

CheriBSD is a mature research operating system that can be used to prototype projects making use of new security primitives provided by CHERI, and as a development platform to test software against security vulnerabilities. While a lot of third-party software has been adapted for CHERI, there are still many crucial applications missing that would enable developing projects in new areas. We are excited to see the growing CHERI ecosystem, with at least 70 organisations from the Technology Access Programme[26], the CHERI within Defence and Security competition[27] and Digital Security by Design[28]. In the following months, we expect to continue our work to increase the number of available pure-capability packages.

References

1. cheribuild.py. https://github.com/CTSRD-CHERI/cheribuild
2. cheribuild.py. the qemu-cheri-bsd-user branch. https://github.com/CTSRD-CHERI/cheribuild/tree/qemu-cheri-bsd-user
3. CheriBSD. https://www.cheribsd.org/
4. CheriBSD packages. https://pkg.cheribsd.org/
5. CheriBSD Poudriere logs. https://poudriere.cheribsd.org/
6. CheriBSD ports. https://github.com/CTSRD-CHERI/cheribsd-ports
7. Poudriere extended for CheriBSD. https://github.com/CTSRD-CHERI/poudriere
8. Poudriere infrastructure for CheriBSD packages. https://github.com/CTSRD-CHERI/poudriere-infrastructure
9. QEMU with support for CHERI. https://github.com/CTSRD-CHERI/qemu
10. QEMU with support for CHERI. the qemu-cheri-bsd-user branch. https://github.com/CTSRD-CHERI/qemu/tree/qemu-cheri-bsd-user
11. BSDCan 2015: Embedded FreeBSD Development and Package Building via QEMU. https://www.bsdcan.org/2015/schedule/events/532.en.html
12. BSDCan 2015: Stacey Son. https://www.bsdcan.org/2015/schedule/speakers/267.en.html
13. The qemu-bsd-user project. https://github.com/qemu-bsd-user/qemu-bsd-user

14. The CHERI LLVM Compiler Infrastructure. https://github.com/CTSRD-CHERI/llvm-project
15. The GNU debugger extended to support CHERI. https://github.com/CTSRD-CHERI/gdb
16. Microsoft. CHERIoT: Rethinking security for low-cost embedded systems. https://www.microsoft.com/en-us/research/publication/cheriot-rethinking-security-for-low-cost-embedded-systems/
17. Arm. The CHERI LLVM Compiler Infrastructure. https://git.morello-project.org/morello/llvm-project
18. Morello Platform Software Repositories. https://git.morello-project.org/morello/docs
19. Arm. Morello Development Tools. https://developer.arm.com/Tools%20and%20Software/Morello%20Development%20Tools
20. Arm. Morello Program. https://www.arm.com/architecture/cpu/morello
21. Robert N. M. Watson, et al. Getting Started with CheriBSD. Installing on a Morello Board. https://ctsrd-cheri.github.io/cheribsd-getting-started/morello-install/
22. Robert N.M. Watson, et al. An Introduction to CHERI. Technical Report UCAM-CL- TR-941, University of Cambridge, Computer Laboratory, 2019.
23. Robert N. M. Watson, et al. Assessing the Viability of an Open-Source CHERI Desktop Software Ecosystem, Technical Report, Capabilities Limited, 17 September 2021.
24. Brooks Davis, et al. CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment. In Proceedings of 2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS'19). Providence, RI, USA, April 13-17, 2019.
25. Robert N. M. Watson, et al. Adversarial CHERI Exercises and Missions. https://ctsrd-cheri.github.io/cheri-exercises/
26. Digital Security by Design. Technology Access Programme Participants. https://www.dsbd.tech/whos-involved/technology-access-programme-participants/
27. Defence and Security Accelerator. Competition: CHERI within Defence and Security. https://www.gov.uk/government/publications/competition-cheri-within-defence-and-security
28. Digital Security by Design. Funded Projects. https://www.dsbd.tech/whos-involved/funded-projects/

KONRAD WITASZCZYK is a Research Associate and a PhD Student at the University of Cambridge working on the CHERI project. He graduated with a BSc degree in Theoretical Computer Science from the Jagiellonian University, an MSc degree in Computer Science from the University of Copenhagen and has been working with FreeBSD and its security-related mechanisms since 2013, including at Fudo Security. As part of his PhD, he is studying and working on compartmentalisation strategies for the CheriBSD kernel, and thus the FreeBSD kernel as well.

SCaLE20X
Conference Report

BY DREW GURKOWSKI

A month ago, I had the opportunity to head down to Pasadena, California to join members of the open source community at SCaLE20X. This was my second SCaLE, and the first time I had been to one held in the original Pasadena location.

On the first day of the conference, I assisted Roller Angel with running a full-day FreeBSD workshop. The goal of the day was to help people install FreeSBD on either a virtual machine or cloud device, install and run a desktop environment, set up a basic jail, create a local ports repository, and more. While it mostly went off without a hitch, we did run into a small issue when Vultr (the cloud provider we were using) was unable to process card payments, sending us scrambling to find a different provider for those attendees. The turnout was great, and we got a wide range of participants in the workshop, from FreeBSD newbies, to experienced users wanting to put FreeBSD on their new machines. If you are interested in going through the workshop in your own time, Roller Angel has posted it online as a text file which can be found here.

I also attended the SCaLE expo hall and staffed the FreeBSD table there for most of the event. Talking to members of the open-source community is always my favorite part of these events, and we bring plenty of FreeBSD swag to hand out while there. As I have recently been working on an expansive FreeBSD timeline, I was particularly interested in many of the attendees who had been working with Unix-like operating systems since their inception. A memorable conversation at the booth was with a FreeBSD user who told us a story of when, as a child, he had taken apart his parents' computer and installed FreeBSD. The parents, horrified when they saw the state of the computer and were greeted by the unfamiliar "daemon" when it booted, grounded him, and forbade him from using FreeBSD. Fortunately, he still uses FreeBSD to this day (Though I didn't ask what his parents thought about it now).

I believe that SCALE was, yet again, another successful advocacy effort to encourage more people to use and contribute to FreeBSD. The workshop continues to be a highlight, and each year we see more attendees. I'm looking forward to SCaLE21X and hopefully meeting even more of the FreeBSD community!

DREW GURKOWSKI is the Marketing Coordinator at the FreeBSD Foundation.

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

Let's chat, GPT

BY BENEDICT REUSCHLING

You've probably heard about it by now: there is a new, controversial star in the field of artificial intelligence (AI) called ChatGPT. A lot of things have been written about it, both how it will improve people's lives by doing certain tasks that are considered chores or how it will doom us all by letting the robots win. The spectrum is wide, and the jury is still out on whether we should regulate AI or not. Since this is practical ports, I'm going to take a practical look at what the chatlike tool can do.

I remember my own university classes about artificial intelligence and natural language processing. The former was an unexciting lecture with little value, since, at the time, there were no big breakthroughs happening in the AI field and so the lecture mostly revolved around past efforts in the field—it was a history lesson.

The natural language course in my masters was more engaging. One of the tasks in the class was to program a simple version of Eliza. Back in the day, when the computerized therapist first came out, it was lauded as if it would make psychoanalysts file for unemployment the very next day. But looking at it now, that certainly has not happened. Since we students had to implement this ourselves, we got a better understanding of what the program is actually doing. It was exciting to see what it could do and with a bit of added randomness, it did not get too predictable or boring too soon. The illusion of the almighty, all-knowing program was certainly shattered, having had a look at the inside of Frankenstein's software monster.

> The illusion of the almighty, all-knowing program was certainly shattered.

ChatGPT is obviously more complex and had a lot more knowledge fed into it than Eliza did. The software is impressive but should certainly be taken with a grain of salt as the following experiment demonstrates.

At work, I install cluster machines the old-fashioned way: Boot the FreeBSD ISO image, drop to the shell and set up basic networking (`ifconfig ixl0 up && dhclient ixl0`). Then, I use netcat to transfer a shell script to the host and run that. While there are certainly better and automatic ways (e.g., PXE boot, pre-built images, etc.), this one has served me for a while now. The script is not too complex but has gone through a number of evolutions because of slight variations in the underlying hardware: some nodes have only HDDs, newer ones have NVMe drives, and some have a mixture of HDDs and SSDs. While the script runs fine, I thought I'd let the AI have a look to see if it found any obvious errors or had suggestions for improvements.

The disk setup for the boot, swap, and freebsd partitions for the four disks in the node in combination with ZFS were always a bit tricky. My goal there is to create a RAID-10 zpool (2 mirrors striped together) with the disks having the disk serial numbers as identifiers like this:

```
pool: mypool
 state: ONLINE
config:

NAME                    STATE     READ WRITE CKSUM
mypool                 ONLINE       0     0     0
 mirror-0              ONLINE       0     0     0
   gpt/ABC543210       ONLINE       0     0     0
   gpt/DEF123450       ONLINE       0     0     0
 mirror-1              ONLINE       0     0     0
   gpt/IJK098761       ONLINE       0     0     0
   gpt/LMN678901       ONLINE       0     0     0
```

(See, the gpt is everywhere, but this gpt refers to the GUID partition table). Since the serial numbers are different on each system, I have a little routine that figures out the serial number of each disk and creates a pool, making sure that the mirror keyword is added to the list of serial numbers, as ZFS requires this to create the RAID10.

When I first heard the term GPT, my initial thought was: "why would someone talk to the GUID Partition Table?" I must have worked too much with partitions and disk setups in my day job, so I thought, let's give it a try. Since the GPT in ChatGPT stands for generative pre-trained transformer, the distinction was clear. After creating an account at https://chat.openai.com/chat, a rather unimpressive chat window was presented to me. So, I typed the following:

BCR: How would you create a function for this piece of code:

```
for S in $SSD; do
        gpart add -t freebsd-swap -l swap${COUNTER} -s ${SWAP} -a 4k "${S}"
        echo "Setting ${SWAP} of swap space on disk ${S}."
        SERIAL=$(camcontrol identify "${S}"|awk '/serial number/ { print $3 }')
            gpart add -t freebsd-zfs -l "{$SERIAL}" -a 4k "${S}"

        if [ $((COUNTER%2)) -eq 0 ]; then
                SERIALS="${SERIALS} mirror"
        fi
        SERIALS="${SERIALS} /dev/gpt/${SERIAL}"
        COUNTER=$((COUNTER+1))
        NUM_SSDS=${COUNTER}
done
```

Now, this is neither beautiful code nor elegant, but it works. This is what ChatGPT replied:

ChatGPT: Based on the code snippet you provided, I believe you are trying to create a function that sets up swap and ZFS partitions on multiple disks. Here's a sample function that could accomplish that:

```
setup_partitions() {
    local SSD=$1
    local SWAP_SIZE=$2
    local COUNTER=1
    local SERIALS=""

    for S in $SSD; do
        # Create swap partition
        gpart add -t freebsd-swap -l swap${COUNTER} -s ${SWAP_SIZE} -a 4k "${S}"
        echo "Setting ${SWAP_SIZE} of swap space on disk ${S}."

        # Create ZFS partition
        SERIAL=$(camcontrol identify "${S}"|awk '/serial number/ { print $3 }')
        gpart add -t freebsd-zfs -l "${SERIAL}" -a 4k "${S}"

        # Add to serials and counter
        if [ $((COUNTER%2)) -eq 0 ]; then
            SERIALS="${SERIALS} mirror"
        fi
        SERIALS="${SERIALS} /dev/gpt/${SERIAL}"
        COUNTER=$((COUNTER+1))
    done

    NUM_SSDS=${COUNTER}
}
```

ChatGPT: You can call this function by passing two arguments: an array of SSDs and the desired swap size. For example, if you have two SSDs (`/dev/sda` and `/dev/sdb`) and you want each of them to have 4GB of swap space, you can call the function like this:

```
SSD=("/dev/sda" "/dev/sdb")
SWAP_SIZE="4g"
setup_partitions "$SSD" "$SWAP_SIZE"
```

OK, that's certainly impressive for a start. It could figure out what I was trying to do and offered a solution, along with a fitting explanation. On closer inspection, it needed a couple of modifications: First, the code was written in bash (as the code snippet window was labeled). Since we're on the installer and FreeBSD does not ship with bash by default, bash-specific constructs would not work. Second, it assumed I was using Linux as the disk device nodes indicate (`/dev/sda`). The rest of the code was fine though, and it did not have to modify most of my own code, simply wrapping it into a function.

Now, my prompt (my input into the chat) was not too specific about the system I was running this on, so the AI had to make a best guess about both the shell I was running and the device names. I should have provided this up-front to avoid the AI going in the wrong direction.

What impressed me was how specific the answer was. It felt like I was chatting with a colleague and discussing this code via IRC or other ways of textual communication. This specificity is not to be underestimated. Back when I was studying (I feel old now), the internet was just becoming mainstream, and we still had to go to the library to access certain information. Had I been tasked with a problem like this, I would have had to find a book about shell scripting or ask a fellow student. But in a book, I would get general concepts about programming, explanations, and example code. I would still need to apply what I'd read to the problematic code I had to come up with a fitting solution.

Nowadays, people would google their problem and find similar questions on sites like stackoverflow or forums dealing with coding. I could even post the code there and ask for help if I couldn't find the solution myself. But that requires waiting time for an answer (if anyone actually bothers to reply) and maybe going through a couple of iterations to discuss various approaches.

With tools like ChatGPT, it's like having a personal coding assistant available to discuss your code without a long wait for an answer. Just like Chief Engineer LaForge had "cozy little chats" with the Enterprise's computer, a second pair of eyes can give new ideas and insights that a single person would not have thought of. I could even re-generate the answer that ChatGPT was giving me or ask for more details. Since the AI seems to understand the problem domain (better than a book might), it can make specific recommendations when it is given enough information. Since my colleagues are not available to me 24/7 to discuss a piece of code, I can see that an AI is useful in situations when I need a good answer quickly.

Encouraged, I continued giving the AI a bigger example:

BCR: How would you unify this piece of code into one generic function for both?

```
for HD in $HDD; do
        gpart add -t freebsd-swap -l swap${COUNTER} -s ${SWAP} -a 4k "${HD}"
        echo "Setting ${SWAP} of swap space on disk ${HD}."
        SERIAL=$(camcontrol inquiry "${HD}" -S|sed -e 's/^[[:space:]]*//')

        if [ $((COUNTER%2)) -eq 0 ]; then
                SERIALS="${SERIALS} mirror"
        fi
        SERIALS="$SERIALS /dev/gpt/${SERIAL}"

        gpart add -t freebsd-zfs -l "${SERIAL}" -a 4k "${HD}"

        COUNTER=$((COUNTER+1))
done

COUNTER=0
# we boot from one of the SSDs
for S in $SSD; do
        gpart add -t freebsd-swap -l swap${COUNTER} -s ${SWAP} -a 4k "${S}"
        echo "Setting ${SWAP} of swap space on disk ${S}."
        SERIAL=$(camcontrol identify "${S}"|awk '/serial number/ { print $3 }')
     gpart add -t freebsd-zfs -l "${SERIAL}" -a 4k "${S}"

        if [ $((COUNTER%2)) -eq 0 ]; then
                SERIALS="${SERIALS} mirror"
        fi
        SERIALS="$SERIALS /dev/gpt/${SERIAL}"
        COUNTER=$((COUNTER+1))
        NUM_SSDS=${COUNTER}
done
```

ChatGPT: To unify the two pieces of code into one generic function, you can create a function that takes three arguments: an array of disks, the desired swap size, and a flag indicating whether the disks are HDDs or SSDs. Here's an example function:

```
setup_partitions() {
    local disks=("$@")
    local swap_size=$2
    local is_hdd=$3
    local counter=1
    local serials=""

    for disk in "${disks[@]}"; do
        # Create swap partition
        gpart add -t freebsd-swap -l swap$counter -s $swap_size -a 4k "$disk"
        echo "Setting $swap_size of swap space on disk ${disk}."

        # Determine disk serial number
        if [[ $is_hdd == "true" ]]; then
            serial=$(camcontrol inquiry "$disk" -S | sed -e 's/^[[:space:]]*//')
        else
            serial=$(camcontrol identify "$disk" | awk '/serial number/ { print $3 }')
        fi

        # Create ZFS partition
        gpart add -t freebsd-zfs -l "$serial" -a 4k "$disk"

        # Add to serials and counter
        if [ $((counter%2)) -eq 0 ]; then
            serials="$serials mirror"
        fi
        serials="$serials /dev/gpt/$serial"
        counter=$((counter+1))
    done

    NUM_DISKS=$counter
}
```

ChatGPT: You can call this function with an array of disks, the desired swap size, and a flag indicating whether the disks are HDDs or SSDs. For example, to set up partitions on two HDDs (`/dev/ada0` and `/dev/ada1`) with 2GB of swap space and two SSDs (/dev/ada2 and /dev/ada3) with 4GB of swap space, you can call the function like this:

```
DD=("/dev/ada0" "/dev/ada1")
SSD=("/dev/ada2" "/dev/ada3")
SWAP_SIZE_HDD="2g"
SWAP_SIZE_SSD="4g"

setup_partitions "${HDD[@]}" "$SWAP_SIZE_HDD" "true"
setup_partitions "${SSD[@]}" "$SWAP_SIZE_SSD" "false"
```

ChatGPT: Note that I added a flag to indicate whether the disks are HDDs or SSDs, since the serial number retrieval method is different for each type of disk.

Another impressive response for sure. Especially the verbosity of the reply and explanations beyond handing me some random code snippets. It did figure out the similarities of both functions and could distinguish that the swap size is different for SSDs and HDDs in my example. It was clever enough to figure out the proper disk device nodes (`/dev/ada0` instead of **`/dev/sda`**). Also, some of my coding style like having uppercase variable names were transformed to lower case. This may or may not be what you like and requires some hand-editing afterwards. The code was still given in bash with some more language specific parts like arrays that are not available in the Bourne shell.

But I rectified my repeated mistake with the following prompt:

BCR: Can you provide the same code in **`/bin/sh`** only (no bash)?

ChatGPT: Sure! Here's the same code in **`/bin/sh`** syntax:

```
#!/bin/sh

setup_partitions() {
    disks="$1"
    swap_size="$2"
    is_hdd="$3"
    counter=1
    serials=""

    for disk in $disks; do
        # Create swap partition
        gpart add -t freebsd-swap -l swap$counter -s $swap_size -a 4k "$disk"
        echo "Setting $swap_size of swap space on disk ${disk}."

        # Determine disk serial number
        if [ "$is_hdd" = "true" ]; then
            serial=$(camcontrol inquiry "$disk" -S | sed -e 's/^[[:space:]]*//')
        else
            serial=$(camcontrol identify "$disk" | awk '/serial number/ { print $3 }')
        fi

        # Create ZFS partition
        gpart add -t freebsd-zfs -l "$serial" -a 4k "$disk"

        # Add to serials and counter
        if [ $((counter%2)) -eq 0 ]; then
            serials="$serials mirror"
        fi
        serials="$serials /dev/gpt/$serial"
        counter=$((counter+1))
    done

    NUM_DISKS=$counter
}

HDD="/dev/ada0 /dev/ada1"
SSD="/dev/ada2 /dev/ada3"
SWAP_SIZE_HDD="2g"
SWAP_SIZE_SSD="4g"

setup_partitions "$HDD" "$SWAP_SIZE_HDD" "true"
setup_partitions "$SSD" "$SWAP_SIZE_SSD" "false"
```

ChatGPT: Note that there are a few syntax differences between `/bin/sh` and `/bin/bash`, such as the use of **=** instead of **==** for string comparison, and the lack of array variables. To work around these differences, I passed the disk lists as space-separated strings, and used **$1**, **$2**, and **$3** to access the function arguments.

So, it was able to transform the code into a working Bourne shell syntax (I tested the results each time).

Now that I had a look at the unified function, I had another idea (which again, could have come from discussing this with a colleague looking over the code):

BCR: What would the code look like when we use the same 4g as the swap size for both HDD and SSD?

ChatGPT: If we want to use the same swap size of 4g for both HDD and SSD, we can modify the `setup_partitions` function to take a single disk argument and remove the `is_hdd` argument and the conditional logic for determining the disk serial number. Here's what the updated code would look like in `/bin/sh` syntax:

```
#!/bin/sh

setup_partitions() {
    disks="$1"
    swap_size="$2"
    counter=1
    serials=""

    for disk in $disks; do
        # Create swap partition
        gpart add -t freebsd-swap -l swap$counter -s $swap_size -a 4k "$disk"
        echo "Setting $swap_size of swap space on disk ${disk}."

        # Determine disk serial number
        serial=$(camcontrol identify "$disk" | awk '/serial number/ { print $3 }')

        # Create ZFS partition
        gpart add -t freebsd-zfs -l "$serial" -a 4k "$disk"

        # Add to serials and counter
        if [ $((counter%2)) -eq 0 ]; then
            serials="$serials mirror"
        fi
        serials="$serials /dev/gpt/$serial"
        counter=$((counter+1))
    done

    NUM_DISKS=$counter
}

HDD="/dev/ada0 /dev/ada1"
SSD="/dev/ada2 /dev/ada3"
SWAP_SIZE="4g"

setup_partitions "$HDD $SSD" "$SWAP_SIZE"
```

ChatGPT: Here, we pass both the HDD and SSD disk lists as a single space-separated string to the `setup_partitions` function and use the same swap size of 4g for both types of disks.

Alright, this is certainly nice as a refactoring step. This time, it did learn that I wanted `/bin/sh` code from my earlier inputs, so it defaulted to it in this next reply.

Bear in mind that results may vary and may or may not work as well as shown here. The AI is constantly evolving, and I surely trained it with my prompts a tiny bit to get better. Each reply had the option to be rated with the familiar thumb-down-or-up buttons.

Next up, I wanted to see what the AI thought about any potential for optimizations:

BCR: Can you recommend other optimizations to the code?

ChatGPT: Sure, here are a few potential optimizations:

1. Use a more efficient way to increment the counter variable: Instead of using `counter=$((counter+1))` inside the loop, you can use the shorthand `counter=$((counter++))` or even just `((counter++))`. This avoids creating a subshell to perform the arithmetic operation, which can be slower.
2. Use the `-n` option for echo to avoid printing a newline character: When printing the message about setting swap space, you can use the `-n` option for echo to avoid printing a newline character, since you're immediately following it with another message. This will make the output look cleaner.
3. Use parameter expansion to remove whitespace: Instead of using `tr -d '[:space:]'` to remove whitespace from the serial number, you can use parameter expansion to achieve the same result: `${variable##*[[:space:]]}`. This avoids the need to spawn an additional process and may be faster.

Here's the code with these optimizations applied:

```
#!/bin/sh

setup_partitions() {
    disks="$1"
    swap_size="$2"
    counter=1
    serials=""

    for disk in $disks; do
        # Create swap partition
        gpart add -t freebsd-swap -l swap$counter -s $swap_size -a 4k "$disk"
        echo -n "Setting $swap_size of swap space on disk ${disk}. "

        # Determine disk serial number
        serial=$(camcontrol identify "$disk" | grep "serial number" | cut -d':' -f2 |
${SED-sed} 's/^[[:space:]]*//')

        # Create ZFS partition
        gpart add -t freebsd-zfs -l "$serial" -a 4k "$disk"

        # Add to serials and counter
        if [ $((counter%2)) -eq 0 ]; then
            serials="$serials mirror"
        fi
        serials="$serials /dev/gpt/$serial"
        ((counter++))
    done

    NUM_DISKS=$counter
}

HDD="/dev/ada0 /dev/ada1"
SSD="/dev/ada2 /dev/ada3"
SWAP_SIZE="4g"

setup_partitions "$HDD $SSD" "$SWAP_SIZE"
```

ChatGPT: These optimizations should make the code slightly more efficient, but keep in mind that the impact on performance may be negligible depending on the size of the disks and the number of partitions being created.

These are certainly some optimizations I had not considered. When running your code through an AI like this, I suggest you approach it with an open mind and leave your programmer ego at the login prompt. Nobody's perfect and the AI may suggest stuff that is either theoretically incorrect or making you fear losing your job because word of the horrible code you've written thus far may reach your boss's ear. But, since this is just between you and the AI, what is stopping you from implementing some of the suggestions to see if they actually improve things?

This is where I find these tools to be a double-edged, blunt sword: if students use it to generate solutions to tasks given to them, no one is stopping them. It's a tool like any other, be it your shell, your compiler, or a handy keyboard shortcut. But, if they can't explain it or just take the code without reflecting on what it does, it becomes dangerous. Cheats don't prosper, especially when they must one day present a solution without any helpers on a whiteboard in front of an examining committee. Understanding is required as well as critical reflection on the solution that is presented. It may have flaws, be inefficient or outright stolen from someone else who provided this exact solution to a forum or stackoverflow and it got fed to the AI's training data. That is a controversy the experts are still discussing.

It's a tool like any other, be it your shell, your compiler, or a handy keyboard shortcut

Try seeing it as a helper tool—a second pair of artificial eyes looking over your code providing valuable insights. Also, be aware that examiners know about this and may use these tools themselves—either to create assignments or exams that are uniquely tailored to each student.

My own discussion with the AI continued a bit more about parts of this code. I also had it give me suggestions in a separate chat about standing desks of certain dimensions (my workspace is a bit limited) and what optimizations I could put into hadoop configuration files to speed up writes to HDFS.

My suggestion is for people to try it out, take the output with a grain of salt, and use it when it makes sense. I think these tools can improve certain areas like making code more robust or sparking new ideas which then must be implemented by you. I don't anticipate losing my own job because of the existence of these tools any time soon, as there are certain skills that we humans have that are not yet available in ChatGPT and friends.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Benedict is one of the hosts of the weekly bsdnow.tv podcast.

FreeBSD

The FreeBSD Project is looking for

- Programmers
- Testers
- Researchers
- Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website
freebsd.org/projects/newbies.html

Downloading the Software
freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at **freebsdfoundation.org**

Help Create the Future. Join the FreeBSD Project!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by

FreeBSD
FOUNDATION

WeGet letters

by Michael W Lucas

Dear We Get Letters,

Why do all these open-source projects have foundations anyway? Aren't we just contributing code to the world? Why bother with these silly legalities?

—Just Hack The Code

JHTC,

The simplest way to understand open-source foundations is to start a really tiny open-source project, preferably as a hobby. Nurture that project until it becomes key infrastructure for a considerable portion of the world's technology stack. You will discover that all your time, energy, health, relationships, and hope have been exchanged for technological success and an impressive colony of keratinophilic onychomycosis. Going barefoot in the sunlight would help the latter, if your homeland experiences occasionally thaws, which mine does not. You can exchange technological success for precious Social Media Clout, a virtual currency accepted precisely nowhere. Still, my efforts to breed new varieties of *Trichophyton rubrum* proceed nicely.

As usual, the problem is people.

You write a bit of code to solve a problem. That's fine. That's what people in our business do. The problem starts when you invitingly share that code. Yes, I occasionally include my code in books, but only in the most hostile manner possible. You won't find my code on GitHedge or ForgedSource or any of those dubious repositories of unhealthy knowledge. If you want my code, you must retype it from the print book. The act of passing it through your eyes into your brain and down into your fingertips gives your nervous system ample opportunity to recoil in revulsion and fling the tome into the nearest incinerator. If you snag my code from an ebook, you'll discover that an electronic document can contain characters that are invisible in place, but when copied and pasted become obvious. Code in my ebooks all include the black sigil *Odegra*, which translates to "Hail the Great Beast, Devourer of Worlds," so copying is self-correcting or at least self-immolating. Being the world's foremost proponent of fault-oblivious computing imposes heavy responsibilities, but I fulfill them as completely as my microplastics-infested meatsuit permits.

But you? You share your code and invite others to use it. To evaluate it. To send bug reports. To *deploy it in production*. Some other person finds your code and it doesn't quite meet her needs, so she sends a patch to add a feature, which makes the code more inviting so other people adopt it, and pretty soon you have dozens of users. Hundreds. Thousands. Perhaps hundreds of millions, and you're hunched over a keyboard evaluating patches and settling disagreements 25 by 8 by 366, living your worst life and wishing you had time to scratch your feet.

Every one of those users and contributors has a different idea of what your software should be. You've foolishly retained a sense of community so you feel obliged to tell them all the exact nature of their errors, but writing lengthy emails would crank your days up to thirty hours and at this point you're incapable of recording coherent videos. It would be so much easier if you could berate them into submission face-to-face.

Again, the problem is people.

You can't stick a meatsuit in a cardboard box and ship it by sea mail. They are heavy and need air and feeding and watering and the occasional bout of vice. They need tickets for trains or planes or dolphins or however they get from their current spot to the meeting. And you need a spot for a meeting. You could invite everyone to your hovel, but you're better off remaining ignorant of exactly which of your contributors are indifferent or full-on hostile to personal hygiene. This means purchasing, conquering, or renting a meeting space.

If your software is widely used and you have collected enough Social Media Clout, you can probably interest big companies in giving your project money. Monopolists believe that donations to the little folk absolve them of guilt. Your problem is, they don't want to give you the money. They want to give your project the money. Does your project have a bank account? No, because you need identification to get a bank account and you didn't even think to get it a legal birth certificate before your first public posting, you selfish short-sighted doofus. Let alone a motor vehicle operator's license. If you want outsiders to give you—uh, your project, your *project*—money, it needs a legal entity.

You write a bit of code to solve a problem. That's fine. That's what people in our business do. The problem starts when you invitingly share that code.

You could start a company, but then outsiders would expect you to provide a service or product in exchange for their cash. Not only is that work you don't have time for, the aforementioned outsiders would care what that service or product is and how reasonable the price looks. But a foundation? A foundation is charity. People give money to charities to do their charity thing and don't care if it's reasonable or not. If the Internet depends on your software, that's a legit charity. Plus, charities can employ people. You could collect donations, pay yourself a salary, ditch the day job, and fall back to working on your project a paltry twelve hours a day!

It's not that easy, because—again—people. Every country has voluminous rules on who can form foundations and how they must be licensed and which reports must be filed with which agencies. If your foundation collects enough funds, it will need to hire a person who knows what they're doing, creating another set of headaches, except you're outsourcing them to the person you hire so that's okay.

A well-run foundation supports its project. By "support," I mean it can buy tickets and meals and meeting spaces and even pay people to write particularly vexing code or to sojourn to distant lands and slap particularly obnoxious would-be contributors with a white glove and challenge them to pistols at dawn. A particularly intransigent foundation can even hire *lawyers*, or at least keep them on retainer. Most of the funding comes from big companies who know perfectly well how you're spending the money but get tax benefits so they don't care. Foundations also need a large number of small contributors, to show the tax authorities that individuals care about the charity and that the foundation is not about tax evasion, or at least not *only*. Your five dollars is not about the five dollars: it's about adding a name to the list of people who care.

So, if a foundation does all this, why don't more projects have one?

The exact same problem: people.

The FreeBSD Foundation that suckered me into answering your insipid^Wirritating^Winevitable letters (while still not delivering on the gelato I was promised thirty columns ago) relies on community to do the work. Someone has to figure out what meetings need to happen and which tickets need purchasing and who merits administration of a white glove across the cheekbone at high velocity. So long as the foundation remains involved in the community, and the community with the foundation, all is well. The people are the foundation.

And by "people," by the way, I mean you. The person reading this column.

So, there you have it. A foundation is a method of converting big company money into junkets, meals, and gelato. Except there's no gelato.

Now pardon me while I do something about my feet and contemplate solving once and for all the actual root cause of everything.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS has written more than fifty books, including *Absolute FreeBSD*, *$ git commit murder*, and *Networking for System Administrators*. He seriously expected this Journal to give him the boot years ago and intends to incrementally increase the vitriol of his column until they do so. Learn more at https://mwl.io.

Books that will help you. Or not.

"While we appreciate Mr Lucas' unique contributions to the Journal, we do feel his specific talents are not being fully utilized. Please buy his books, his hours, autographed photos, whatever, so that he is otherwise engaged."

— John Baldwin
FreeBSD Journal Editorial Board Chair

https://mwl.io
FreeBSD Mastery Jails
SSH Mastery
FreeBSD Mastery: Storage Essentials
Letters to ed(1)
DNSSEC Mastery
Ed Mastery
Tarsnap Mastery: Online Backups for the Truly Paranoid
FreeBSD Mastery: Specialty Filesystems
Michael W Lucas
ONLY FOOTNOTES
PAM Mastery
Sudo Mastery
SNMP Mastery
FreeBSD Mastery Advanced ZFS
TLS Mastery

2023 Events Calendar

BSD Events taking place through July 2023

BY ANNE DICKISON

Please send details of any FreeBSD related events or events that are of interest for FreeBSD users which are not listed here to freebsd-doc@FreeBSD.org.

May 2023 FreeBSD Developer Summit

May 17-18, 2023
Ottawa, Canada
https://wiki.freebsd.org/DevSummit/202305

Join us for the May 2023 FreeBSD Developer Summit, co-located with BSDCan 2023, which will take place in Ottawa, Canada. The two-day event takes place May 17-18, 2023 and will consist of developer discussion sessions, vendor talks and working groups. The fee to attend is $75 and you must register through the BSDCan 2023 Registration System.

BSDCan 2023
The Technical BSD Conference
High value. Low cost. Something for everyone.

BSDCan 2023

May 17-20, 2023
Ottawa, Canada
https://www.bsdcan.org/2023/

BSDCan is a technical conference for people working on and with BSD operating systems and related projects. It is a developers conference with a strong focus on emerging technologies, research projects, and works in progress. It also features Userland infrastructure projects and invites contributions from both free software developers and those from commercial vendors.

FOSSY 2023

July 13-16, 2023
Portland, OR
https://2023.fossy.us/

A new event, the hope is that FOSSY will become a community focused conference that brings together the local open source community as well as the wider global community focused on Free and Open Software. The FreeBSD Foundation is pleased to be holding an Introduction to FreeBSD workshop.