EECS 4314 - Bit Theory
Architecture Enhancement Report

Amir Mohamad
amohamad@my.yorku.ca

Arian Mohamad Hosaini
mohama23@my.yorku.ca

Dante Laviolette
dantelav@my.yorku.ca

Diego Santosuosso Salerno
nicodemo@my.yorku.ca

Isaiah Linares
isaiah88@my.yorku.ca

Joel Fagen
joefagan@my.yorku.ca

Misato Shimizu
misato1@my.yorku.ca

Muhammad Hassan
furquanh@my.yorku.ca

Yi Qin
aidenqin@my.yorku.ca

Zhilong Lin
lzl1114@my.yorku.ca

York University

April 9, 2023

Abstract

This report proposes the development of a new feature for the Docker system. This new feature is the implementation of Cgroup and Namespace API for FreeBSD. At the moment, Docker's performance depends on the Linux operating system's Cgroup and Namespace features. By creating a FreeBSD equivalent of these capabilities, Docker may be made to function seamlessly on the platform, expanding its user base and increasing its general adaptability.

The suggested improvement will be thoroughly covered, including its potential value and advantages for different stakeholders. We'll look at the architectural adjustments needed to accommodate this functionality, paying particular attention to the high-level parts and interfaces that need to change. The study will examine how this new feature interacts with existing functionality and give a thorough comparison of the system's current and improved states.

Additionally, the study will examine how this architectural change will affect both high-level and low-level system concepts, with the help of an architecture diagram that graphically illustrates the changes. The discussion of flagged files, use cases, and concurrency issues will also be included to ensure that all aspects of the proposed improvement are clear.

The implementation of the Cgroup and Namespace API for FreeBSD will be assessed for potential risks and constraints, identifying any potential negative outcomes. The study will end with a summary of the results and an evaluation of the feasibility and advantages of adding this improvement to the Docker software system overall.
Keywords— cgroups, namespaces, containerization, Docker, compatibility

Contents

1 Introduction

1.1 Problem Definition

FreeBSD currently relies on jails for providing containerization capabilities. However, as containerization needs evolve, the limitations of jails become more evident. Jails work well for simple isolation use cases but lack features critical for more advanced container workflows. They provide a basic level of encapsulation but lack the portability and interoperability of modern container formats. Jails also have limited options for networking, storage, and orchestration between containers. Additional container tools are needed to address these limitations and meet the growing demands of containerized applications and infrastructure. Options like Kubernetes are attracting developers and sysadmins to containerized platforms but FreeBSD remains limited to jails.

Integration with other open source container projects would make FreeBSD far more suitable for the microservices, cloud-native environments, and CI/CD pipelines that are increasingly prevalent. When combined with FreeBSD's security, stability and customizability, new container technologies could significantly expand usage possibilities.

With containerization becoming integral to application development and deployment, FreeBSD requires further enhancements beyond just jails. Options for container formats, networks, storage, and orchestration would provide far more flexibility and capabilities. Developing or leveraging additional tools would position FreeBSD as a robust, configurable operating system able to support containerized workloads at scale.

Advancements in containerization are essential for FreeBSD to meet evolving needs and remain competitive. Jails alone no longer suffice for the kinds of containerized architectures and workflows common today. Additional container tools will be needed to address limitations, unlock new potential, and ensure FreeBSD keeps pace with rapid changes in containerization. Keeping up with container futures will require more possibilities beyond just jails.

1.2 Docker

Docker is a famous software that uses OS-level virtualization to deliver software in packages called containers. It has been widely used in many mainstream operating systems such as Windows, macOS, and Linux. Sadly, Docker can't operate on FreeBSD due to the difference in kernel features. Though the Jails in FreeBSD have similar functions as Docker, most users think that Docker works much better than Jails. In fact, Docker performs better than Jails in almost all categories except startup time. As a result, it's reasonable to try to make Docker work on FreeBSD. To reach our target of allowing features like Docker to be functional in FreeBSD, there are many different approaches. For example, we can run a Linux Virtual Machine within FreeBSD, so that we can containerize without using Jails. However, this will surely be an unnecessary move. Among all those approaches, the one we choose to use to solve the problem is implementing cgroup and namespace 'like' APIs that internally use jails. In Linux, docker requires cgroup and namespace, which are featured on the Linux kernel, to run. What we want is to implement a clone of cgroup and namespace for FreeBSD, so that we can get a result of running Docker on FreeBSD.

1.3 Overview

To clearly introduce our project from multi-angel, the report has been divided into six different chapters:

- Enhancement - This part explains how the project works and explains the benefits it brings.
- Stakeholders - Show the influence that might happen to developers, users, and other stakeholders.
- Architectural Impact - This part shows the concrete architecture changes caused by the enhancements through diagrams.
- Concurrency - Compares concurrency of different versions of cgroup in Linux.
- Risk & Limitation - Find out the risk and limitations that might appear.
- Conclusion - Summarize the whole report and record the lessons learned in the process.

2 Enhancement

2.1 Our Proposed Enhancement

Our proposed enhancement is to extend the current Linux emulator for FreeBSD, Linuxulator, to also handle cgroup and namespace calls. Currently they are not implemented because they are features that act on hardware and can't be directly mapped to a FreeBSD system calls. This causes problems with many Linux binaries that utilize cgroups or namespaces as well as Container runtimes which are used widely in Linux development. After this interface is added almost all binaries will now run easily using the Linuxulator, which is important because Linux is heavily supported and has a lot of developers making useful binaries. Another benefit is that OCI compliant container run-times like runc, Containerd, or Docker would work natively instead of using ports. The port for running OCI compliant runtimes, runj, utilizes jails to make Containers and is currently under development. This works great natively for FreeBSD but the goal is to integrate FreeBSD into the broader Docker ecosystem and using our interface would allow widely used OCI runtimes to work seamlessly and remove complication.

2.2 Alternatives

For many Linux binaries running through Linuxulator there aren't any great alternatives to this interface, as cgroup and namespaces are very integral to some programs. It is possible to edit Linux binaries through FreeBSD but this can cause the Linux binary to use a lot of efficiency or functionality. A more viable solution is to use a Linux Virtual machine inside FreeBSD, which allows you to run any Linux binary and use all types of OCI runtimes like Docker, Containerized, runc, etc… Although a lot of developers will not like this method as it brings a lot of overhead which will slow down Linux binaries. This is something Linux developers will not tolerate because the main reason people use Linux operating systems is because it removes overhead and allows them to work more efficiently!. This is why our solution is favorable to many developers.

3 Stakeholders

With this new implementation of Namespace and Cgroups for FreeBSD several stakeholders who would be involved.

3.1 Developers

The first stakeholder is the developers who are working to implement this new improvement as they will be the ones responsible for creating code that will enable docker to run on the FreeBSD platform.Their work will require expertise in kernel architecture, API design, and programming languages like C and Rust, among others.

3.2 Users

Secondly, users of Docker who want to use the platform FreeBSD are another stakeholder group. These users, who want to leverage Docker's capabilities on the FreeBSD platform, could be freelance developers, small businesses, or bigger corporations. They might be interested in how Cgroup and Namespace API implementation will affect performance, stability, or security.

3.3 Docker

Thirdly the Docker company would also be another key stakeholder. By supporting the FreeBSD platform, they will get the chance to increase their user base. The strategic market expansion or rivalry with competing containerization platforms may be the driving forces behind Docker's involvement in this project.

3.4 FreeBSD Community

The FreeBSD community is also a stakeholder group. The Cgroup and Namespace API would improve FreeBSD's functionality and make it a more desirable platform for users and developers. The project might also spark more interest in the FreeBSD operating system, which might result in additional contributions to the open-source initiative or a rise in the use of FreeBSD in commercial settings.

4 Architecture

4.1 Current State of The System

In the current state of the system, the Linuxulator subsystem provides kernel interfaces identical to Linux system calls to allow for emulation [2]. It does this by mapping Linux system calls to their FreeBSD equivalents [8], while also emulating parts of the Linux filesystem using `linsysfs` [6] Due to this, many Linux binaries can run natively on FreeBSD if the `linux_enable` flag is enabled in `/etc/rc.conf`.

Although, Linuxulator currently doesn't support `cgroups` or `namespaces`. Due to this, any binaries that rely on these Linux features cannot run natively on FreeBSD without modifications.

With that being said, it's important to note that FreeBSD has `jails`, which cover most of the use-cases of `cgroups` and `namespaces`. A `jail` provides containerization, while `namespaces` and `cgroups` can be used together to enable containerization.

4.2 Enhanced State of The System

Our proposal is to extend Linuxulator to provide interfaces for `namespaces` and `cgroups`, while using `jails` internally.

More specifically, `namespaces` will be supported by supporting the `set_tid` argument in the `linux_clone3` system call (`sys/compat/linux/linux_fork.c`) which is currently ignored. Then when the `CLONE_NEWPID` flag is passed in `set_tid`, a new jail will be created, which will enable namespaces [3]. Internally, some logic will handle mapping the jail ID to the returned PID, and simulate the cloned process seeing itself as PID 0.

The `cgroups` virtual file system will then be supported in the Linuxulator file system (`linsysfs`), which will act as an interface for modifying the `jails` resources.

With that being said, this will cause the Linuxulator to become dependent on jails. Although, it will also greatly improve compatibility with linux binaries, allowing applications such as Docker to run natively, while still taking advantage of `jails`.

4.3 Interactions with Other Features

In this part, we will be discussing about the impact of interaction and design patterns after adding these features. We know that Cgroups allows the system administrator to allocate system resources and place limits on resource usage. First, it will interact with the virtual memory subsystem to create a more secured system environment. It can be used to limit the amount of memory being used and preventing it from consuming excessive system resources. The second interacting part can be security subsystem. Cgroups can be used in conjunction with other security mechanisms. For example, with the Mandatory Access Control, it can prevent unauthorized access to system resource. There will be two design patterns can be found in this feature. It will use command pattern to implement the functionalities and improve the system performance. Another pattern that scan be found is the builder pattern. It is good for customize Cgroups for different applications and use cases. It can also make it easier to create and manage complex configurations. Namespace is a feature allows for the creations of isolated environments that have their own independent view of certain system resources including process IDs, network interfaces, file systems, user IDs, and other system resources. It will be interacted with the process management system. Namespaces can be used to create isolated process trees so the process can be limited into smaller parts. Second interacting part can be file system. Namespaces can be used to isolate files systems between different processes. It can prevent the applications from accessing or modifying files outside their own namespaces. The design pattern might be found is proxy pattern. Since it will create an independent view of certain system resource, using proxy pattern can allow the system to create a virtualized view of the system that is isolated from the underlying system. It will make it easier and safer to customize the behavior of the virtualized system.

4.4 Effect of the Enhancement on the Concrete Architecture

The Linulator subsystem resides in the newly discovered subsystem called *Compat*, or the Compatibility subsystem. Enhancing FreeBSD for native Docker support by extending the linuxulator subsystem but keeping jails largely unchanged would not drastically alter the concrete architecture of the FreeBSD operating system itself. The core architecture and design of FreeBSD would remain intact, with Docker capability added through strategic, focused changes rather than a complete overhaul.

While linuxulator would gain strengthened Jail integration and dependence on it to support containers, the existing jail subsystem would persist and continue functioning as before. Jails have been a fundamental part of FreeBSD's containerization strategy and do not require replacement with Docker. Integrating Docker would add useful complementary functionality rather than replacing central components. In fact, it will add a dependency to the existing central component. Administrators and users could utilize either jails or Docker containers, or both, based on their particular needs.

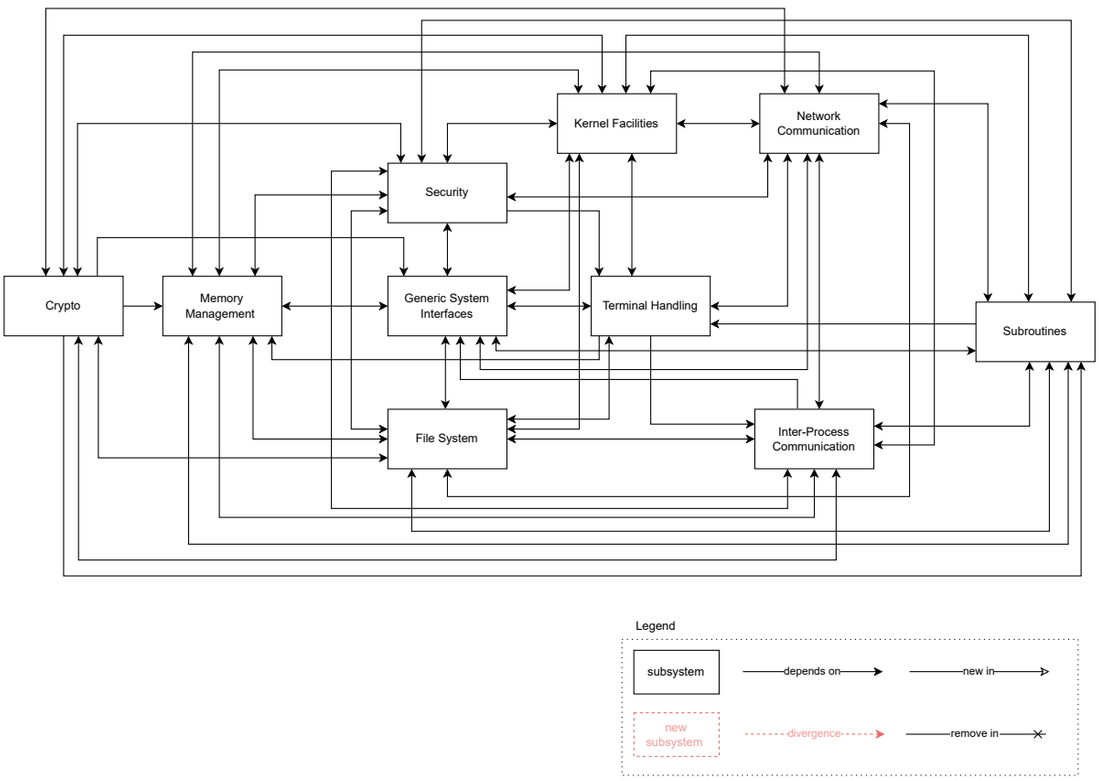

Figure 1: Current Concrete Architecture

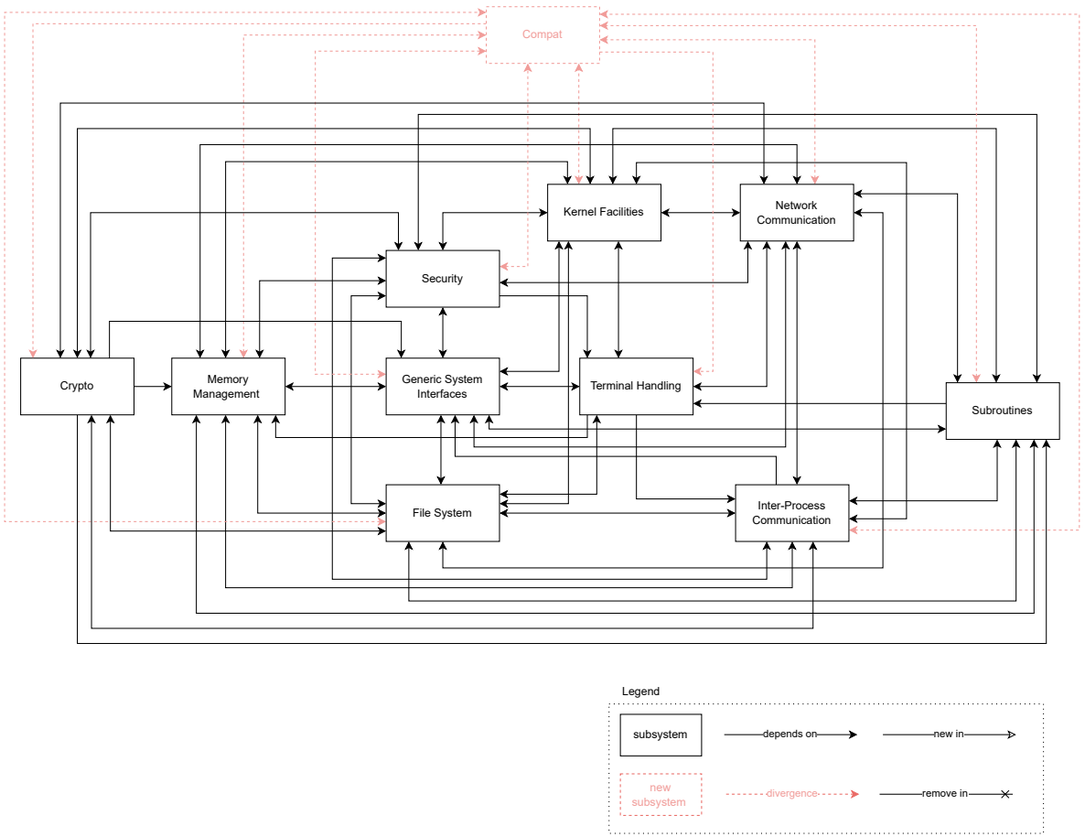

Figure 2: Enhanced Concrete Architecture

4.5 Flagged Files

The proposed enhancement to the Linuxulator subsystem in FreeBSD will significantly impact the system's directory structure. Firstly, the Linuxulator subsystem, which provides a Linux kernel interface for emulation, will be extended to include support for jails. This will introduce a new way for users to leverage FreeBSD's native containerization technology while running Linux applications like Docker. As a result, the impacted directories include:

- `sys/compat/linux`
- `sys/compat/linsysfs`

`sys/compat/linux` provides binary compatibility files for Linux, while the `sys/compat/linsysfs` provides file system emulation for `cgroups`. The proposed enhancement will also require changes to specific files, including `sys/compat/linux/linux_fork.c` and its `linux_clone3()` method, to provide support for namespaces by enabling the `set_tid` arg. Once these changes are made, the Linuxulator subsystem will have improved Linux binary compatibility, thanks to the newly added support for cgroups and namespaces. This will enable FreeBSD to run any Linux binary that requires the use of containerization, making it an attractive platform for developers and businesses alike.

5 Concurrency

There are two different versions for how threads are handled in version 1 and 2. In version 1, threads are made distinct from processes. This way, one process can consist of multiple threads at a time. It is also possible to change the cgroups memberships of the threads within the process. However, that ability was removed in version 2, because it caused issues in some cases. For instance, all threads in a process shared a single address space, which caused issues for the memory controller, because it makes no sense to split threads across multiple memory cgroups. That ability was brought back in version 2's thread mode.

In version 2, it has two restrictions imposed on it. One of them is no thread-granularity control, meaning that all threads of a process must belong to the same cgroup. Another restriction is no internal processes, meaning that a cgroup cannot have member processes while controlling child cgroups at the same time. In some cases, it makes sense to have thread-granularity control, such as the CPU controller, so that ability was brought back in the form of thread mode.

Thread mode has several abilities. One is the ability to create threaded subtrees where the threads of a given process are spread across multiple cgroups inside the tree. Another is a new feature called threaded controllers, which can distribute resources across the cgroups in a subtree. Also, the restriction for no internal processes is relaxed. Now, a cgroup can have its own threads and manage child cgroups at the same time.

6 Use Cases

6.1 Docker

An important use case of our enhancement is the use of an industry standard tool, Docker. Docker does not work without `namespace` and `cgroup` functionality and our compatibility enhancement would allow these types of applications to run on FreeBSD. Docker and similar containerization engines are widely used in distributed systems within the industry and the diagram below covers the process of emulating `namespace` and `cgroup` functionality in FreeBSD.

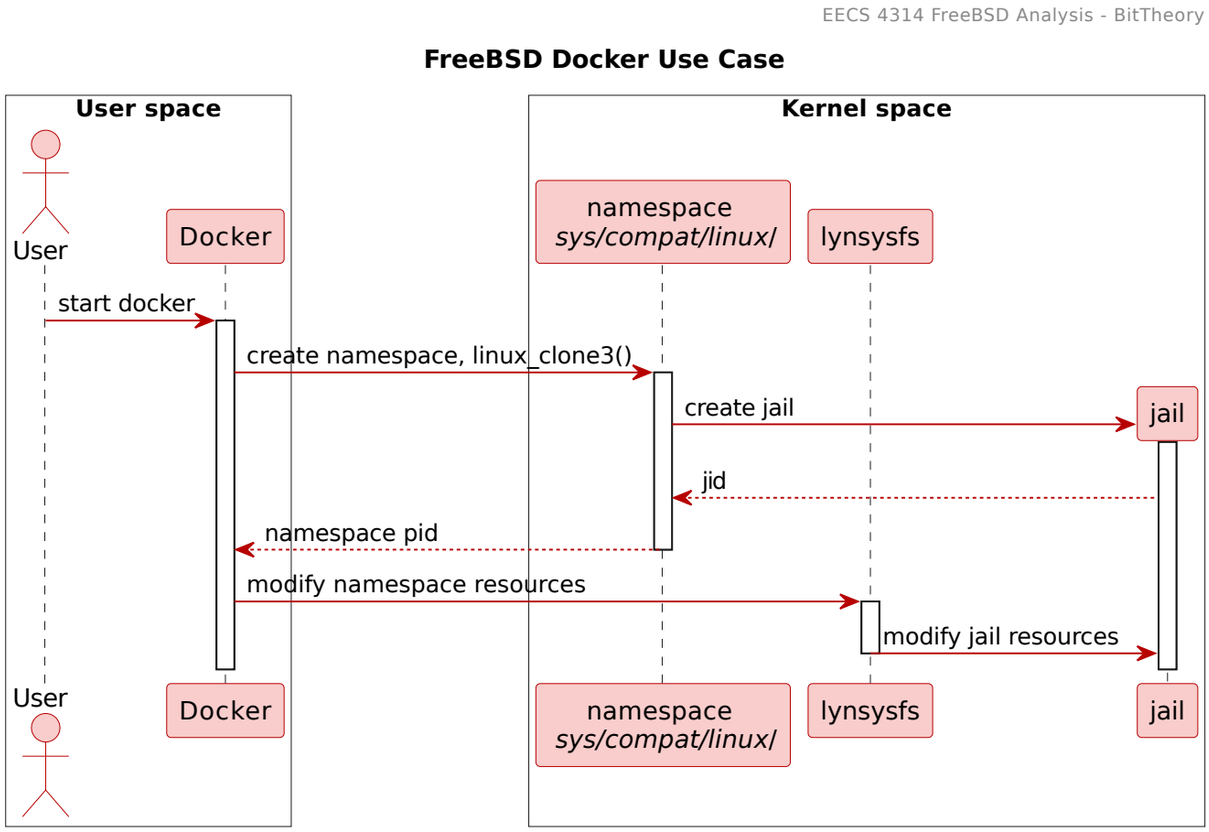

Figure 3: Docker use case using enhanced state.

7 Risks & Limitations

Although the proposed enhancement will lead to improving the overall operating system with better compatibility with Linux, it is still significant to examine how it would potentially affect other factors. For the section, we will observe the risks and limitations from three perspectives: Security, Maintainability, and Performance.

7.1 Security

One of the major concerns people would consider when they use containers is leaks of files and resources between the objects. Containers without complete separations would lead to poor efficiency for developments. However, it would not be a concern for this enhancement at this time because this proposed enhancement is based on existing functions and features. For example, to make `namespace`'s functions work correctly on FreeBSD, we utilize the existed `clone()` system call, which Linuxulator supports, instead of introducing a completely new idea into the system. Moreover, the mechanism of containers, which will be necessary for the namespace functionality, are from Jails. Jails is a virtual-machine-like containerization tool, completely separating users and processes from the space outside of a Jail [4]. Therefore, utilizing it as part of the enhancement's functionalities would keep the system secure.

7.2 Maintainability

It might be an issue to maintain our proposed enhancement simply because of the relatively small population of contributors who actively support FreeBSD overall. As proof, only several-thousands of developers involve in FreeBSD development [5]. On the other hand, over 10,000 developers contribute in Linux more actively and frequently [9]. It is possible that the failure of implementation on native Docker development in the past is also due to this factor. Moreover, as there are already existing developed or developing implementations that can support Linux compatibility on FreeBSD, there might not be enough developers interested in or encouraged to maintain this new enhancement in the future. One of the existing implementations is runj, which follows the basics of runc for Linux [7].

7.3 Performance

This aspect would be challenging to estimate as there is no existing case of internal usage of Jails. As explained earlier in this report, our idea of the enhancement is to create the same functionalities as unsupported Linux features, particularly `cgroups` and `namespaces`. Although the enhancement involves modifying the existing functions to add a new parameter feature and introducing fully built container system for isolations, it is still possible that the system will not work as it does by itself.

8 Conclusion

In conclusion, the proposed enhancement to extend the linuxulator subsystem in FreeBSD has the potential to significantly improve Linux binary compatibility on FreeBSD. Through a detailed analysis of the benefits, architectural impact, and use cases/sequence diagrams, the proposal demonstrates promising prospects for the FreeBSD operating system and its users.

The enhancement allows for improved compatibility with Linux binaries, particularly for containerization platforms like Docker. This will provide FreeBSD users with access to a wider range of applications and a more seamless deployment and management experience.

The benefits of this enhancement are numerous. As mentioned throughout this report, it would improve compatibility with Linux binaries, enabling FreeBSD to support a wider range of applications and services. This increased compatibility would make FreeBSD more attractive to stakeholders, developers and users who rely on Linux-specific applications, potentially expanding the user base and fostering further development.

In addition, implementing cgroups and namespaces support would not only enhance resource management and process isolation within the operating system (which would consequently optimise performance, allocate resources more efficiently, and ensure better isolation between processes), but would also facilitate its integration with Docker. This would further broaden FreeBSD's appeal and applicability in various use cases, from personal computing to large-scale data centres.

While there are some limitations and risks associated with this enhancement, such as the potential for leaks of files and resources between containers, with proper attention to security and maintainability, these risks can be mitigated. Additionally, the increased interest in the FreeBSD operating system that this enhancement may generate could result in additional contributions to the open-source community and further improvements to the operating system in the future.

In summary, extending the linuxulator subsystem in FreeBSD to provide interfaces for cgroups and namespaces using jails internally offers numerous benefits, including improved Linux binary compatibility, enhanced resource management and process isolation, and better integration with containerization technologies. However, it is essential to acknowledge the limitations and challenges associated with the proposed enhancement that will increase the overall system's complexity. By carefully considering these factors and mitigating the identified challenges, the proposed enhancement has the potential to bring significant value to the FreeBSD operating system and its users.

9 Lessons Learned

Importance of compatibility between operating systems: Enhancements that improve compatibility with other operating systems can significantly expand the range of applications and tools available to users, making an operating system more versatile and attractive. By ensuring compatibility, an operating system can also be used more easily in a variety of environments and integrated more easily with other systems.

Different methods of containerization in Linux and FreeBSD: As we analysed the FreeBSD Operating System to look for potential enhancement proposals, we learned about different methods of containerization technologies in both Linux and FreeBSD. Containerization is a popular technology used for deploying and managing applications. It was learnt that the methods used for containerization differ between Linux and FreeBSD, with jails being the preferred method in FreeBSD. Understanding these differences is important for making informed decisions about how to add support for features such as cgroups and namespaces.

Security, maintainability, and performance are potential drawbacks of our proposal: Enhancements that introduce new functionality or change the architecture of an operating system can also introduce potential risks and challenges by increasing it's overall complexity. It is important to carefully consider the potential drawbacks of any proposed enhancement and develop strategies to mitigate risks related to security, maintainability, and performance.

Architectural analysis is crucial the implementation of such enhancement feature: Enhancements that involve changes to an operating system's architecture require careful analysis to ensure that they are technically feasible, maintainable, and efficient. By performing a thorough analysis of the architecture and potential impacts of a proposed enhancement, stakeholders can make informed decisions about the potential implementation of features that could potentially add value to the overall system.

Collaboration with stakeholders is crucial for successful enhancements: Enhancements that involve significant changes to an operating system require collaboration with a range of stakeholders, including developers, users, and companies that may be impacted by the changes, such as Docker in this case. By engaging with stakeholders and soliciting feedback throughout the development process, it is possible to build enhancements that meet the needs of a wide range of users and maximise the benefits of the enhancement for all stakeholders involved.

10 Diagrams

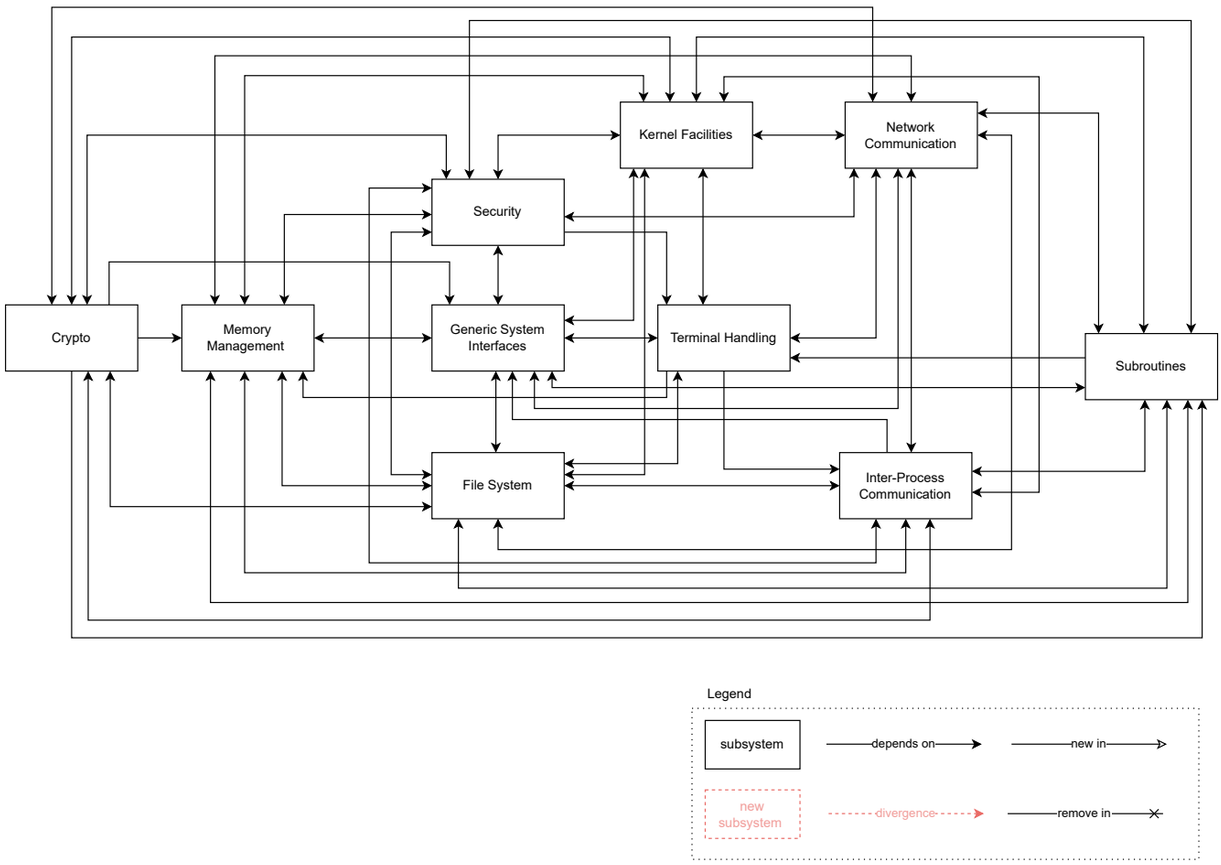

Figure 4: Current state of the concrete architecture diagram of FreeBSD.

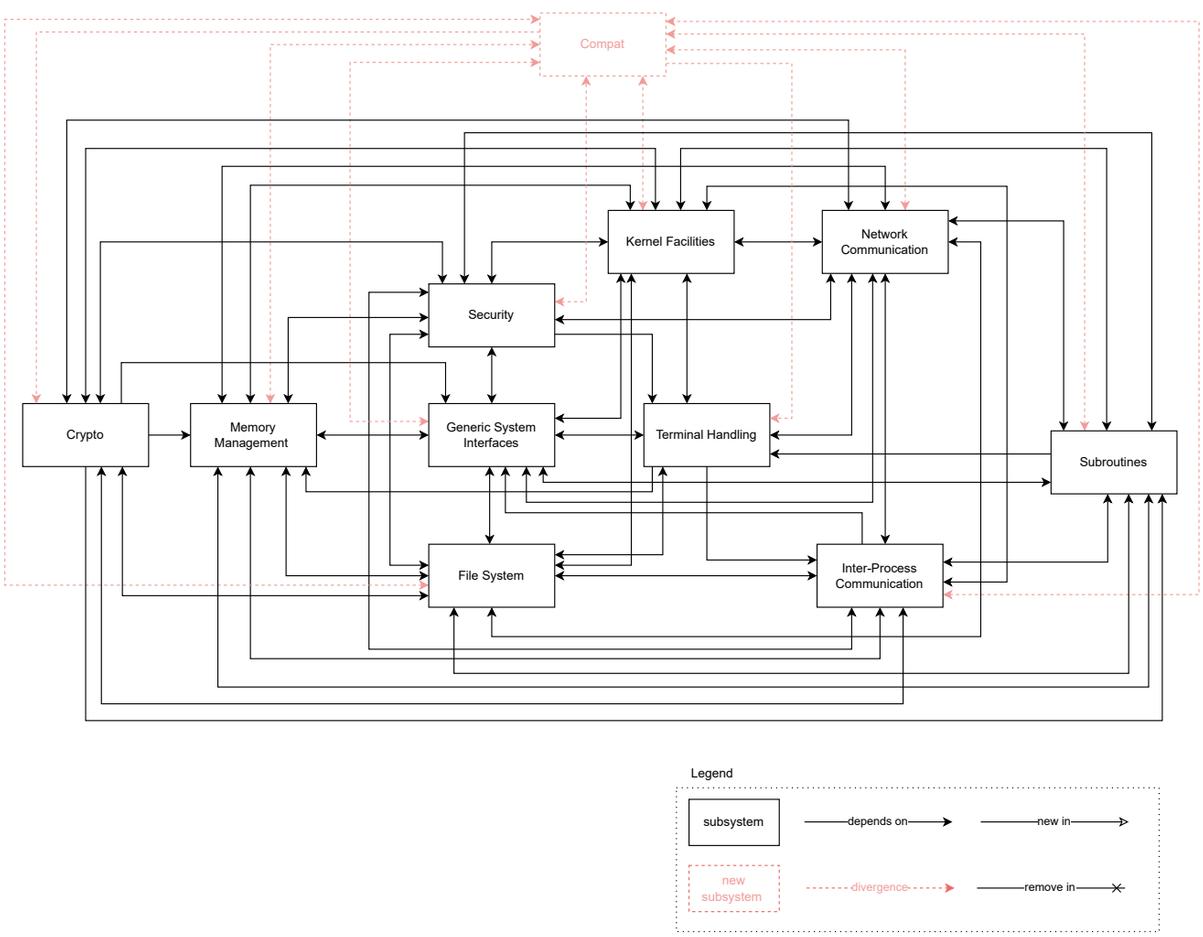

Figure 5: Enhanced state of the concrete architecture diagram of FreeBSD.

11 Data Dictionary

- Linuxulator (Linux Emulation): A technique to Linux binaries on FreeBSD without any modifications [8]
- cgroups: One of the features in Linux, which sets aside resources within namespaces for other usages
- namespaces: One of the features in Linux, which creates several numbers of completely separated spaces within the operating system
- Docker: A technique based on both hypervisor-based and containerized virtualizations
- runj: A Docker-like tool which supports Jails based on the OCI runtime specifications
- runc: A Linux tool which manages container systems based on the OCI specifications
- Jails: A FreeBSD tool which follows the functionalities of the containerized virtualization

12 Naming Conventions

- cgroup: Control group
- OCI: Open Container Initiative

References

[1] Cgroups(7) - Linux Manual Page, https://man7.org/linux/man-pages/man7/cgroups.7.html.

[2] “Chapter 11. Linux Binary Compatibility.” FreeBSD Documentation Portal, https://-docs.freebsd.org/en/books/handbook/linuxemu/.

[3] Clone(2) - Linux Manual Page, https://man7.org/linux/man-pages/man2/clone.2.html.

[4] Delgado, Sergio Carlavilla. “Chapter 16. Jails.” FreeBSD Documentation Portal, 4 Mar. 2023, https://-docs.freebsd.org/en/books/handbook/jails/.

[5] Freebsd. “FreeBSD/FreeBSD-Src : The Freebsd Src Tree.” GitHub, 1993, https://github.com/freebsd/freebsd-src.

[6] “FreeBSD Manual Pages.” Linsysfs, https://man.freebsd.org/cgi/man.cgi?linsysfs.

[7] Karp, Samuel. “Samuelkarp/Runj : Runj.” GitHub, 2020, https://github.com/samuelkarp/runj.

[8] “Linuxulator.” Edited by Graham Perrin, Linuxulator - FreeBSD Wiki, 29 Jan. 2023, https://wiki.freebsd.org/Linuxulator.

[9] Torvalds, Linus. “Torvalds/Linux : Linux Kernel Source Tree.” GitHub, 2002, https://github.com/-torvalds/linux.