IE2050
Operating Systems
2nd Year, 2nd Semester

Assignment

Theseus: An Experiment in Operating System Structure and State Management

Submitted to

Sri Lanka Institute of Information Technology

Submitted by:

Student Name	Registration No.
Q JOYAL	IT22196088
D M T N DISSANAYAKA	IT22169730
D R WICKRAMA ARACHCHI	IT22360496
SHARVAJEN S	IT22231628

04/05/2024

Table of Contents

Workload Matrix

1. A Brief Introduction

Theseus is an experimental operating system (OS) that aims to improve modularity and state management in modern systems software. It is the result of multi-year experimentation and is designed to reduce the states one component holds for another and leverage a safe programming language, Rust, to shift as many OS responsibilities as possible to the compiler. The OS embodies two primary contributions: an OS structure with many tiny components with clearly defined, runtime-persistent bounds that interact without holding states for each other, and an intralingual approach that realizes the OS itself using language-level mechanisms such that the compiler can enforce invariants about OS semantics. These features allow Theseus to achieve live evolution and fault recovery for core OS components in ways beyond those of existing works.

2. Key Findings

2.1. System Hardware Requirements

Regarding system hardware requirements, Theseus has been tested on various real machines, including Intel NUC devices, ThinkPad laptops, and Supermicro servers. The main hardware requirement, and the only known limitation, is the ability to boot via USB or PXE using traditional BIOS, as support for UEFI is still in progress. The paper specifies that Theseus has been successfully tested on different host operating systems like Linux (64-bit Debian-based distributions such as Ubuntu), Windows using the Windows Subsystem for Linux (WSL), MacOS, and Docker container environments. Theseus is designed for the x86_64 architecture and requires a virtual or real x86_64 machine with BIOS.

2.2. Installation Process

2.2.1. Accessing the Theseus Repository

The Theseus repository is hosted on GitHub at https://github.com/theseus-os/Theseus. The top-level README file contains detailed instructions on building and running Theseus. The branch "osdi20ae" contains pre-built Theseus images with specific instructions for reproducing evaluation experiments.

2.2.2. Software Dependencies

Theseus can be built and run in QEMU on host OSes like Linux (Debian-based distributions), Windows (WSL), MacOS, and Docker container environments. Specific package dependencies are listed in the top-level README, with additional packages needed for artefact evaluation specified in the READMEs for each experiment.

2.2.3. Installation

Standard installation procedures are not required for Theseus. Detailed steps to build and run a functional Theseus OS .iso image are provided in the README file of the GitHub repository. Compilation involves using Rust, Make, and working in a no_std, freestanding, bare-metal environment.

2.2.4. Experiment Workflow

All experiments described in the paper are directly implemented within the source code of Theseus and can be run by configuring settings at compile time. Experiments are categorised into groups, each with accompanying scripts and instructions in their respective artefact folders in the repository. Pre-built OS images are available for each experimental setup to simplify the reproduction of results. For more detailed information and specific instructions, refer to the README file in the Theseus repository at https://github.com/theseus-os/Theseus.

2.3. User Interfaces

In Theseus, the Window Manager acts as the traffic controller for what you see and interact with on your screen. It ensures that applications can display their content in windows while managing them effectively. By sharing windows between applications and the manager, it prevents conflicts and ensures smooth operation. The WindowManager structure efficiently handles input events like mouse clicks and keyboard presses, ensuring that your interactions with windows are responsive and intuitive. It is the behind-the-scenes organizer that makes using your computer's interface seamless and enjoyable.

2.4. Process Control

Theseus transforms process management by adopting a task-based model where tasks execute within a shared address space, distinct from the traditional POSIX process model. Task creation is customizable through the TaskBuilder interface, with tasks transitioning through various states like Initializing, Runnable, and Exited. Task management revolves around the Task struct, TaskRefs, and a global task list, enabling efficient multitasking and resource utilization.

Key invariants govern task management in Theseus, ensuring memory safety and resource integrity throughout the task lifecycle. These invariants center around generic type parameters <F, A, R>, representing the entry function's signature, argument type, and return type, respectively:

- **F:** Represents the type of the entry function, including its signature and return type.

- **A:** Denotes the type of the single argument passed into the entry function F.
- **R:** Indicates the return type of the entry function F.

The invariants include:

1. Spawning tasks must not compromise memory safety.
2. All task states must be properly released in all execution paths.
3. Memory reachable from a task's entry function must outlive the task, maintaining memory integrity and preventing leaks.

These invariants uphold the reliability and security of task execution, enhancing system stability and performance.

2.5. Memory Management

Theseus's memory management system is optimized for efficiency, safety, and flexibility within its Single Address Space (SAS) architecture, where all kernel entities coexist in a unified environment. Leveraging Rust's memory safety features, it ensures robust isolation without hardware-based protections. Clear differentiation between virtual and physical memory addresses is maintained through dedicated types like VirtualAddress and PhysicalAddress, minimizing confusion and errors. Pages and frames serve as fundamental units, managed effectively with range types like PageRange and FrameRange.

The page and frame allocators dynamically manage virtual and physical memory, supporting various allocation requests and reserved memory regions during boot. Advanced memory types such as AllocatedPages and MappedPages enforce ownership and exclusivity guarantees, enhancing security and data integrity. MappedPages empower the compiler to enforce key invariants:

1. **Bijective Mapping:** The mapping from virtual pages to physical frames must be one-to-one or bijective.
2. **Bounds Checking:** Memory must not be accessible beyond the mapped region’s bounds.
3. **Unmapping:** A memory region must be unmapped exactly once, only after there remain no outstanding references to it.

4. **Permissions:** A memory region must only be mutable or executable if mapped as such.

These invariants ensure precise memory mapping and access control, enhancing system-wide memory security and reliability.

2.6. Deadlock Management

Deadlock management in Theseus relies on resource cleanup via unwinding, implemented within drop handlers to ensure safe resource release triggered by the compiler. Tasks own resource objects directly, with ownership tracked by the Rust compiler to prevent leakage and simplify deadlock prevention. Lock guards are automatically released during unwinding, enhancing efficiency. The custom-built unwinding process in Rust is independent and triggered only during exceptions or task termination, minimizing performance impact while handling stack frames effectively, even in nonstandard contexts. Theseus enables intralingual resource revocation, either forcibly or cooperatively, ensuring resources are freed exactly once, reducing deadlock risks, and enhancing fault isolation.

2.7. Secondary Disc Scheduling Management

This block-based storage device caching layer aims to optimize disk operations by reducing costly calls to the storage medium, enhancing system efficiency with increased memory usage. However, it is currently limited by hardcoded references to a specific storage device type and lacks support for efficient handling of larger block reads/writes. Cached blocks are stored inefficiently on the heap, and the cache may produce inconsistent results if other system crates write to the device directly. These limitations highlight the need for a more flexible and robust caching solution for secondary disk scheduling management.

2.8. Standard Support

Theseus OS is open sourced where technical assistance is provided through discussion forum in GitHub. The users can go through the issues tab, read any closed issue, or open a new issue to troubleshoot. The Operating-System blog also provides documentation and contact details.

2.9. State Management

Theseus prioritizes minimizing state spill within its cells, ensuring that interactions across cell boundaries do not result in unnecessary state changes. Opaque exportation in client-server interactions allows clients to own progress states independently, reducing OS overhead and eliminating handle-based abstractions. Special states, including soft states and unavoidable hardware-related states, are managed within state_db to ensure system-wide persistence. Examples like MappedPages ownership and Task struct design showcase spill prevention and efficient resource cleanup, promoting separation of concerns and system evolution.

3. Comparative Analysis of Theseus Operating System

Aspect	Theseus	Traditional OS
User Interface	• Redesign needed for improved efficiency • Focus on system-level interactions	• Established GUI frameworks and libraries • Extensive support for application GUIs
Process Management	• Tasks as threads within same address space • Lifecycle management of tasks • Preemptive and cooperative multitasking	• Traditional POSIX process model • Separate process management mechanisms • Context switching for task management
Memory Management	• Single Address Space (SAS) architecture • Utilizes Rust's memory safety features • Dedicated memory types for clarity	• Hardware-based memory protection • Reliance on hardware mechanisms • Less precise terminology
Deadlock Management	• Resource cleanup via unwinding • Drop handlers for timely resource release	• Manual deadlock detection and resolution • Lock-based deadlock prevention
Secondary Disk Scheduling	• Implementation of caching layer for block-based storage devices • Reduction of disk access calls for improved efficiency • Limitations include hardcoded references and inefficiencies	• Utilization of traditional disk scheduling algorithms • Reliance on disk scheduling policies for disk access optimization • Established disk scheduling algorithms with optimizations

State Management	• Minimization of state spill in cells • Opaque exportation for client-server interactions • Management of soft states for convenience and performance	• Standardized state management models • Emphasis on encapsulation and state preservation • Focus on critical state preservation

4. Limitations and Extensions to the Case Study

This report has several limitations as recommended by report guidelines. First, this research is limited to summarizing the key points of the original research namely system hardware requirements, installation process, user interfaces, process control, memory management, deadlock management, secondary disk scheduling management and standard support. Additionally, statement management was included. The document was confined to a word count of around two thousand.

Appendices

Appendix A: Theseus Repository README

- Point by point directions on building and running Theseus
- Pre-assembled Theseus pictures and trial arrangements
- Available at: Theseus Repository README

Appendix B: Theseus Operating System Blog

- Extra documentation and assets
- Contact subtleties and local area support
- Available at: Theseus OS Blog

Appendix C: Theseus Crates Documentation

- Definite documentation of Theseus containers and modules
- Available at: Theseus Crates Documentation

Appendix D: Introduction to Theseus OS

- Exhaustive outline of Theseus operating system highlights and plan
- Available at: Theseus OS Book

Appendix E: Presentation at Usenix Conference

- Show slides or materials from the Usenix gathering
- Extra bits of knowledge into Theseus operating system
- Available at: Usenix Conference Presentation

Appendix F: Memory Management Documentation

- Definite clarification of Theseus memory the executive framework
- Available at: Memory Management Documentation

Appendix G: Task Management Documentation

- Detailed explanation of Theseus task management system
- Available at: Task Management Documentation

Appendix H: Display and Window Management Documentation

- Detailed explanation of Theseus display and window management
- Available at: Display and Window Management Documentation

Appendix I: Block Cache Documentation

- Documentation of Theseus block-based storage device caching layer
- Available at: Block Cache Documentation

References

[1]

K. Boos, N. Liyanage, R. Ijaz, and L. Zhong, “Theseus: an Experiment in Operating System Structure and State Management,” in Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation, USENIX. Accessed: Apr. 01, 2024. [Online]. Available: https://www.usenix.org/system/files/osdi20-boos.pdf

[2]

“___Theseus_Crates___ - Rust,” www.theseus-os.com. https://www.theseus-os.com/Theseus/doc/___Theseus_Crates___/index.html (accessed Apr. 05, 2024).

[3]

“Memory Management in Theseus,” in The Theseus OS Book, Theseus OS. Accessed: Apr. 07, 2024. [Online]. Available: https://www.theseus-os.com/Theseus/book/subsystems/memory.html

[4]

“Tasking Subsystem in Theseus,” in The Theseus OS Book, Accessed: Apr. 07, 2024. [Online]. Available: https://www.theseus-os.com/Theseus/book/subsystems/task.html

[5]

“Display Subsystem,” in The Theseus OS Book, Accessed: Apr. 08, 2024. [Online]. Available: https://www.theseus-os.com/Theseus/book/subsystems/display/display.html

[6]

“block_cache - Rust,” www.theseus-os.com. https://www.theseus-os.com/Theseus/doc/block_cache/index.html (accessed Apr. 08, 2024).

[7]

Kevin Boos and Lin Zhong, “The Theseus OS Book,” GitHub, 2019. https://github.com/theseus-os/Theseus/blob/osdi20ae/book/book.toml (accessed Apr. 22, 2024).

[8]

K. Boos, N. Liyanage, R. Ijaz, and L. Zhong, “Theseus: an Experiment in Operating System Structure and State Management,” www.usenix.org, 2020. https://www.usenix.org/conference/osdi20/presentation/boos (accessed Apr. 24, 2024).