# Content

# Theseus: An Experiment in Operating system Structure and State Management

# Work matrix

Quintus Joyal  IT22196088

- Process Control
- Memory Management
- Deadlock Management

DR Wikrama Arachi   IT221360496

- Brief Introduction
- System Hardware Requirements
- Installation Process

Sharvajen S     IT22231628

- User Interfaces
- Secondary Disc Sheduling Management
- State Management

DM Thimira Niromin Dissanayaka    IT22169730

- Standard Support
- Comparative Analysis of Theseus Operating System
- Limitations

# Table of Contents

# A Brief Introduction

DR Wikrama Arachi IT221360496

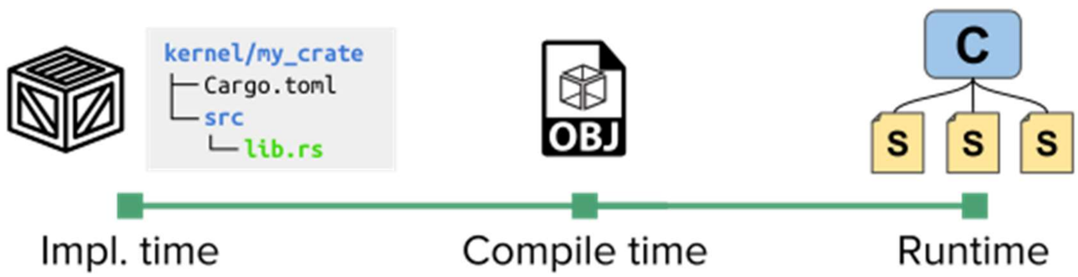
• Experimental operating system for modularity and state management in modern systems software.

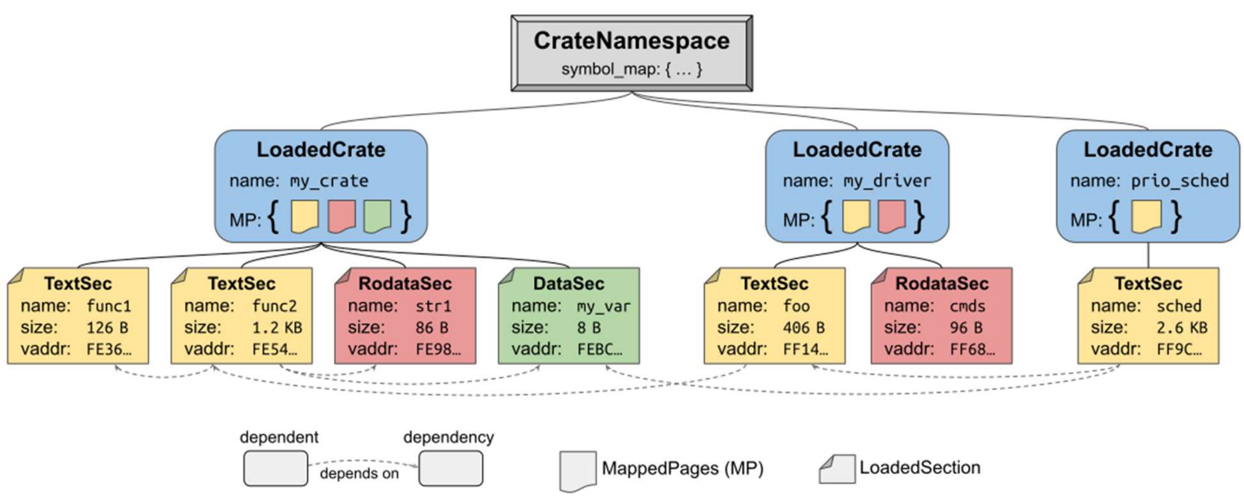
• A safe-language OS running in a single address space and privilege level.

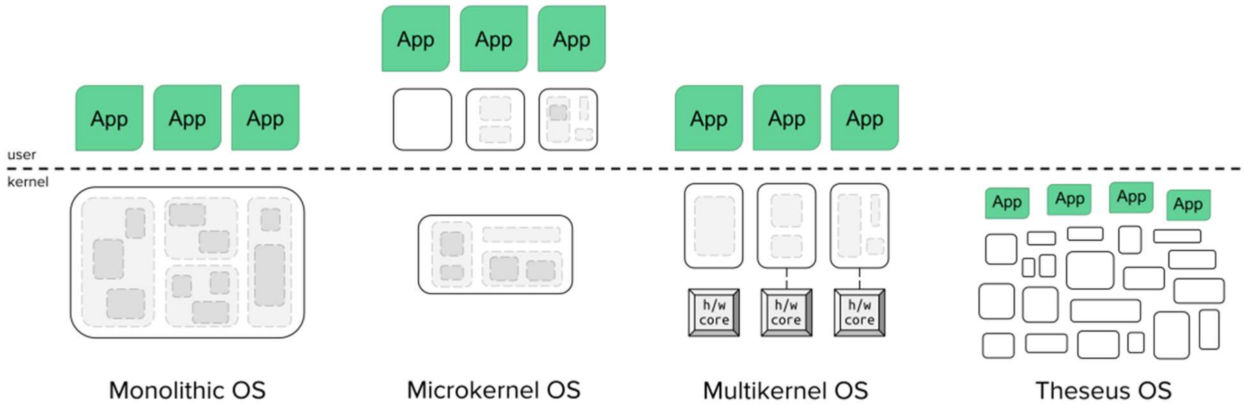
• Implemented as a collection of small cells inspired by biological cells.

• Cell abstraction is present in various forms: a crate at implementation time, a single *.o object file after compile time, and a cell at runtime.

• Distinct from monolithic, microkernel, and multikernel designs, requiring no hardware reliance.




App
App
App
App
App
App
App
App
App
App
App
App
App
user
kernel
h/w core
h/w core
h/w core
Monolithic OS
Microkernel OS
Multikernel OS
Theseus OS

# Key Findings

## System Hardware Requirements

DR Wikrama Arachi IT221360496

• Tested on Intel NUC devices, ThinkPad laptops, and Supermicro servers.

• Main requirement: boot via USB or PXE using traditional BIOS.

• Designed for x86_64 architecture.

## Installation Process

DR Wikrama Arachi IT221360496

• The GitHub repository offers detailed instructions for software building and running.

• Software can be run on Linux, Windows, MacOS, and Docker.

• No standard installation procedures are needed.

• The README file provides instructions for OS.iso image creation.

• Experiments implemented within the source code.

• Pre-built OS images are available for each setup.

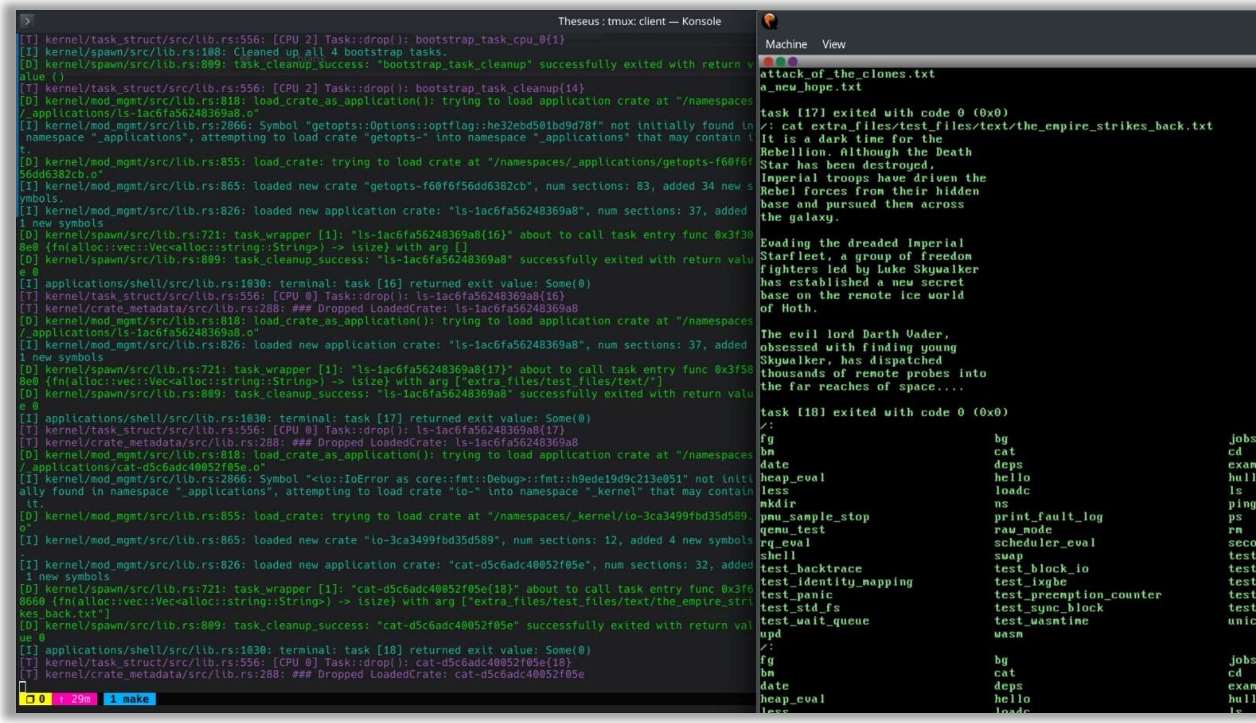


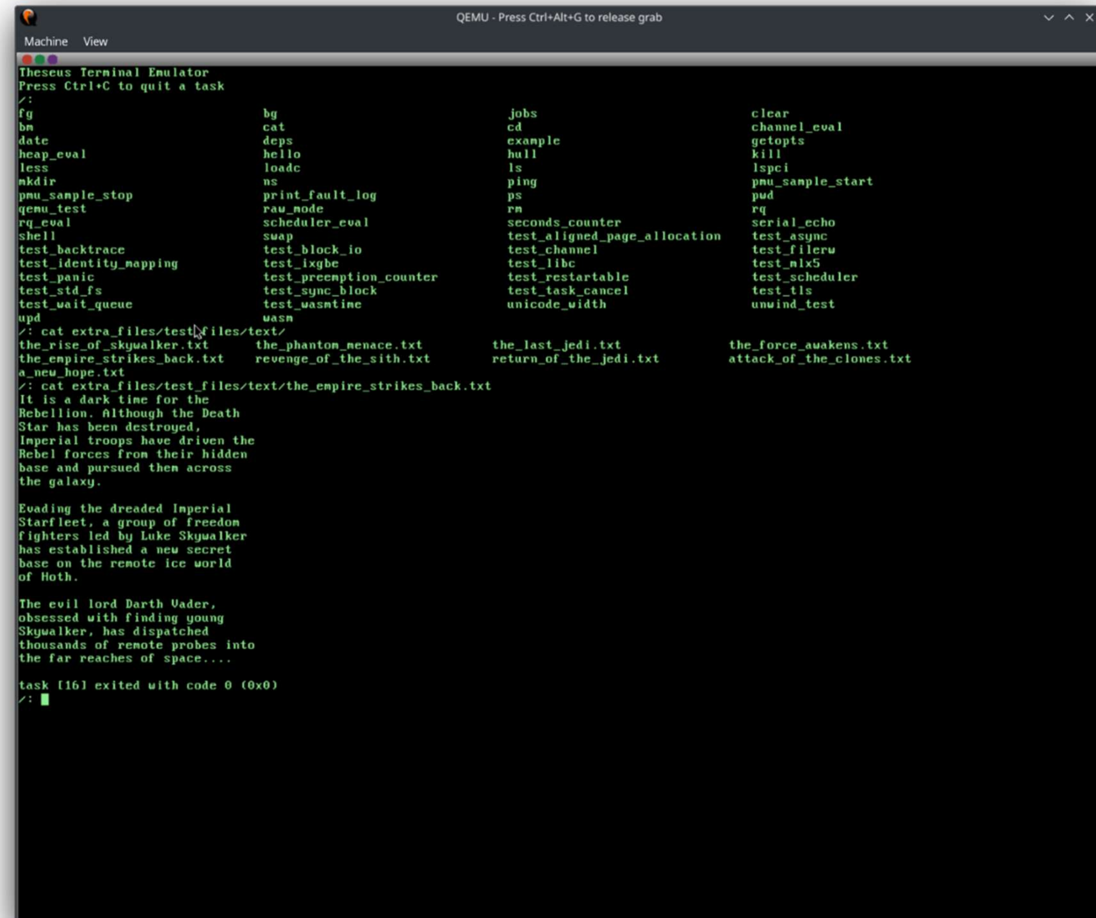

Figure 2.2.3.1: qemu-system-x86_64

## User Interfaces

Sharvajen S IT22231628

By default, Theseus,

- Uses the keyboard as its primary input and optionally a mouse.
- Uses the graphical display as its primary output.
- In headless mode through serial communication, it will spawn a terminal emulator.



## Process Control

Quintus Joyal IT22196088

- The tasking subsystem in Theseus implements full support for multitasking,
- Theseus is a single address space (SAS) OS, it does not have a dedicated address space for each task.
- Does not follow the classic POSIX/Unix-like "process" abstraction.
- The terms "task" and "thread" can be used interchangeably.
- One could also consider the entirety of Theseus to be a single "process" in that all tasks execute within the same address space,

- Context switching from one thread to another thread in the same address space is done by via task_switch().

```
pub fn task_switch(
    next: TaskRef,
    cpu_id: CpuId,
    preemption_guard: PreemptionGuard
) -> (bool, PreemptionGuard)
```

Theseus follows the Rust standard library's model for threading,

- You can spawn a new task with a function or a closure as the entry point.
- You can customize a new task using a convenient builder pattern.
- You can wait for a task to exit by joining it.
- You can use any standard synchronization types for inter-task communication.
- You can catch the action of stack unwinding after a panic or exception occurs in a task.

```
pub struct Task {
    pub id: usize,
    pub name: String,
    pub mmi: Arc<Mutex<MemoryManagementInfo, DisableIrq>, Global>,
    pub is_an_idle_task: bool,
    pub app_crate: Option<Arc<AppCrateRef, Global>>,
    pub namespace: Arc<CrateNamespace, Global>,
    /* private fields */
}
```

Invariants Upheld in Task Management

1. Spawning a new task must not violate memory safety.
2. All task states must be released in all possible execution paths.
3. All memory transitively reachable from a task's entry function must outlive that task.

## Memory Management

Quintus Joyal IT22196088

- All kernel entities, libraries, and applications are loaded into and executed within a single address space.
- Theseus's single address space is a virtual address space, not a physical address space.
- Virtual and physical addresses are given dedicated, separate types that are not interoperable.

| Description of Type | Virtual Memory Type | Physical Memory Type |
|---|---|---|
| A memory address | VirtualAddress | PhysicalAddress |
| A chunk of memory | Page | Frame |
| A range of contiguous chunks | PageRange | FrameRange |
| Allocator for memory chunks | page_allocator | frame_allocator |

Mapping virtual memory to physical memory

- Mapper: provides functions to map virtual memory to physical memory,

- PageTable: a top-level page table.
- MappedPages: Range of virtually contiguous pages that are mapped to physical frames and have a single exclusive owner.

Invariants and Safety Guarantees at Compile-time

1. The mapping from virtual pages to physical frames must be one-to-one, or bijective.
2. A memory region must be unmapped exactly once, only after no outstanding references to it remain.
3. A memory region must not be accessible beyond its bounds.
4. A memory region can only be referenced as mutable or executable if mapped as such.

## Deadlock Management

Quintus Joyal IT22196088

- Utilizes resource cleanup via unwinding within drop handlers.
- Tasks own resource objects directly, with ownership tracked by the Rust compiler.
- Lock guards are automatically released during unwinding for efficiency.
- Custom-built unwinding process is independent and triggered only during exceptions or task termination.
- Enables intralingual resource revocation, reducing deadlock risks and enhancing fault isolation.
-

## Secondary Disk Scheduling Management

Sharvajen S IT22231628

- Optimizes disk operations by reducing costly calls to storage medium.
- Enhances system efficiency with increased memory usage.
- Limited by hardcoded references to specific storage device type.
- May produce inconsistent results if other system crates write to the device directly.
- Calls for a more flexible caching solution for secondary disk scheduling management.

## Standard Support

Thimira

Conceptual Operating System, thus limited support.

Official GitHub page

Read or Open Issues

Discussion Forum

Theseus Blog

Documentation

Contacts

## State Management

Sharvajen S IT22231628

- Prioritize minimizing state spill within its cells
    - Ensures that no unnecessary state changes occur due to interactions between cells
- Opaque exploration in client-server interactions
    - Clients own progress states independently
        - Note: Clients can manage states without constant com with server
    - Reduce OS overhead
    - Eliminate handle-based abstractions
        - Note: Handles represents resources/objects. By eliminating abstractions -> Design simplify + Reduce Overhead
- Special States (soft states, unavoidable hardware related states) managed with state_db to ensure persistence.

State Spill: program's state, such as variables or registers, needs to be temporarily stored in memory[RAM] (spilled)

# Comparison

## User Interface

| Theseus | Conventional OS |
|---|---|
| - Focus- System Level Interactions<br>- Redesign needed for improved efficiency | - Established GUI frameworks and libraries<br>- Extensive Support for application GUI |

## Process Management

| Theseus | Conventional OS |
|---|---|
| - Tasks as threads, with same address space<br>- Lifecycle Management of tasks<br>- **Preemptive**<br>**OS interrupts currently running task to give resources to another task.**<br>- **cooperative multitasking**<br>Tasks volunrarily free memory for other tasks | - Traditional POSIX process model<br>- Separate process management mechanisms<br>- Context switching for task management |

## Memory Management

| Theseus | Conventional OS |
|---|---|
| - Single Address Space (SAS) architecture<br>- Utilizes Rust's memory safety features<br>- Dedicated memory types for clarity | - Hardware-based memory protection<br>- Reliance on hardware mechanisms<br>- Less precise terminology |

## Deadlock Management

| Theseus | Conventional OS |
|---|---|
| - Resource cleanup via unwinding • Drop handlers for timely resource releas | - Manual deadlock detection and resolution • Lock-based deadlock prevention |

## Secondary Disk Scheduling

| Theseus | Conventional OS |
|---|---|
| Implementation of caching layer for block-based storage devices<br>- Reduction of disk access calls for improved efficiency<br>- Limitations include hardcoded references and inefficiencies | - Utilization of traditional disk scheduling algorithms<br>- Reliance on disk scheduling policies for disk access optimization<br>- Established disk scheduling algorithms with optimizations |

## State Management

| Theseus | Conventional OS |
|---|---|
| - Minimization of state spill in cells<br>- Opaque exportation for client-server interactions<br>- Management of soft states for convenience and performance | - Standardized state management models<br>- Emphasis on encapsulation and state preservation<br>- Focus on critical state preservation |

## Limitations and Extensions to the Case Study

Research limited to summarizing Key findings of the original research.

system hardware requirements, installation process, user interfaces, process control, memory management, deadlock management, secondary disk scheduling management and standard support.

Extensions to Research

Secondary Disk Scheduling management

State management

Report ~ 2k Words

# Refrences


[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.theseusos.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.theseusos.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.theseusos.com/Theseus/doc/block_cache/index.html (accessed Apr. 08, 2024)