Microkernel operating systems compared to monolithic operating systems: a review on functional safety

Article · December 2023

CITATIONS
0

READS
3,324

2 authors:

Samesun Singh
Polytechnic University of Turin
5 PUBLICATIONS **1** CITATION
SEE PROFILE

Stephen Torri
Mississippi State University
15 PUBLICATIONS **78** CITATIONS
SEE PROFILE

Microkernel operating systems compared to monolithic operating systems: a review on functional safety

Samesun Singh*, Stephen A. Torri†
Department of Computer Science & Engineering
Mississippi State University
ss4791@msstate.edu*, storri@cse.msstate.edu†

***Abstract*—With the trend of digitalization, intelligence, and networking sweeping the world, functional safety and cyber security are increasingly intertwined and overlapped, evolving into the issue of generalized operational safety. This research paper investigates the influence of microkernel and monolithic operating system architectures on functional safety within safety-critical systems. By examining their impact on fault containment, modularity, isolation, and dynamic reconfiguration, we explore how these architectures shape the development and maintenance of secure and dependable safety-critical systems. The DHR architecture is a mature and comprehensive solution, and it is necessary to implement an OS-level DHR architecture, for which the multi-kernel operating system is a suitable carrier. This study examines the role of microkernel operating systems in enhancing functional safety. By comparing microkernel systems with traditional monolithic counterparts, the research assesses their effectiveness in supporting safety-critical applications. Developers and system architects' perspectives provide insights into how microkernel architectures improve functional safety measures. The findings contribute to understanding how microkernel OS solutions can enhance safety in critical environments.**

***Index Terms*—Kernel Operating Systems, Monolithic Operating Systems, Functional Safety, Operating System Architecture, Safety-Critical Systems, Reliability.**

I. Introduction

In modern computing, the design and architecture of operating systems play a pivotal role in shaping software systems' reliability, robustness, and safety, particularly those categorized as safety-critical. As industries increasingly rely on technology to power intricate and safety-sensitive applications, such as autonomous vehicles, medical devices, and industrial control systems, choosing operating system architecture becomes a pivotal consideration. Two prominent architectural paradigms that have garnered attention in this context are microkernel and monolithic operating systems.

This exploration delves into the intriguing interplay between microkernel and monolithic operating systems in enhancing support for functional safety. We navigate the nuanced landscape of architecture and safety, examining how each approach shapes the development and maintenance of safety-critical systems. While both architectures have their merits, this study delves into the key considerations that underpin the perception of developers and system architects when assessing their potential impact on functional safety. We meticulously analyze these architectural paradigms to unravel the intricacies that determine their respective aptitudes for supporting operational safety. By scrutinizing their attributes in fault containment, modularity, isolation, and dynamic reconfiguration; we seek to elucidate how each architecture influences the creation of dependable, secure, and resilient systems. Moreover, we probe into the real-world implications of these architectural choices, understanding the practicalities that decision-makers confront as they strive to align safety objectives with architectural decisions.

This paper delves into the intriguing realm of microkernel operating systems and their role in advancing functional inclusion criteria. It encompasses studies that distinctly examine these architectures, emphasizing their effects on functional safety, with a preference for empirical evidence from peer-reviewed sources. The relevance lies in empirical research illuminating architecture's ramifications for fault containment, modularity, isolation, and dynamic reconfiguration within safety-critical contexts. Exclusion criteria filter out studies unrelated to safety-critical systems, architectures beyond the scope of microkernel and monolithic, non-English or non-peer-reviewed sources, and materials lacking technical depth.

By comparing the attributes of microkernel designs with those of monolithic architectures, we aim to uncover the potential benefits and challenges associated with adopting microkernel systems in safety-critical contexts. Through the lens of developers, system architects, and industry experts, we seek to explore how microkernel architectures can effectively address the complex demands of functional safety, ultimately contributing to the robustness and reliability of critical systems.

In section II, we discuss the research objective and criteria that shape the scope of the survey. In section III, we discuss the primary research findings that adhere to the rules set in section II. Finally, in section III, we address the research questions set in II-A1 with drawbacks and future recommendations.

II. Research Design and Method

A systematic literature review comprises three steps in the guideline paper by [1].

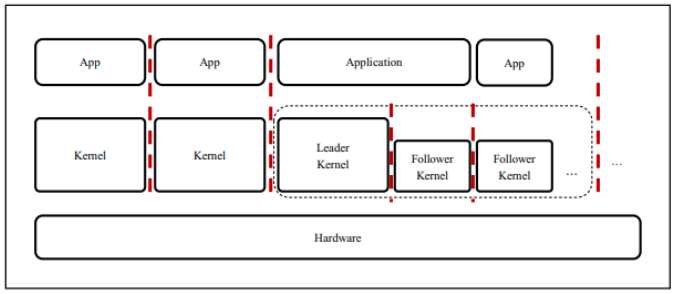

Fig. 1. 1. Multi-kernel Architecture [1]

A. Planning the review

1) Research questions: In this sub-section, we aim to address the background, objective, and outcome of conducting the survey.

- **Q1:** To what extent do microkernel architectures' modular and isolated nature contribute to better fault containment and propagation prevention than monolithic architectures in safety-critical systems?
- **Q2:** How do microkernel and monolithic operating system architectures affect functional safety in safety-critical systems?
- **Q3:** How do microkernels' modularity and isolation principles impact the ease of integrating and verifying safety-critical third-party components compared to monolithic kernels?

2) Search Strategy:

a) Define search terms: In this comprehensive literature review, our goal is to delve into the intricate comparison between kernel-based and monolithic operating systems, with a particular emphasis on their impact on functional safety. Our search for pertinent research materials will encompass a range of carefully selected keywords, including 'kernel operating systems,' 'monolithic operating systems,' 'functional safety,' 'operating system architecture,' 'safety-critical systems,' and 'microkernel versus monolithic kernel.' followed by a meticulous examination of full-text articles to gauge their relevance within the context of functional safety and the comparative analysis of kernel operating systems.

b) Digital libraries used: In software engineering and related domains, we'll explore terminology closely associated with safety-critical software, such as 'safety standards' (such as ISO 26262) and 'safety-critical applications.' To ensure comprehensiveness, these keywords will be thoughtfully combined using Boolean operators (AND, OR) per the syntax requirements of the chosen digital libraries, including ACM Digital Library, IEEE Xplore, and Web of Science. Our selection process will entail an initial evaluation of titles, abstracts, and keywords, IEEE Xplore, a renowned repository of electrical engineering and computer science literature; Science Direct, which offers a wide range of scientific and technical publications; and Web of Science.

c) Any issues using search terms: One critical issue was that different authors and researchers often used various terms and phrases interchangeably to describe similar concepts, which posed a significant challenge in ensuring comprehensive coverage—the issue of finding search item ambiguity and synonym variability. In addition, if we talk about refining, conducting iterative searches, and consulting subject matter experts, they played a pivotal role in mitigating this issue and enhancing the overall robustness of our review.

3) Study selection: In this sub-section, we aim to define the survey's scope by describing the development areas and the filtering process we followed.

The study selection process is constrained by the inclusion criteria (IC) and exclusion criteria (EC). A paper was included if all the following were met:

- IC1 Architecture Types: Include studies that explicitly focus on microkernel and monolithic operating system architectures
- IC2 Modularity and Isolation: Include research that explores the modularity and isolation aspects of both architectures and their implications for fault containment.
- IC3 Functional Safety: Include studies that discuss the impact of architecture on functional safety in safety-critical systems

A paper will be rejected if any of the following are met:

- EC1 Non-Safety-Critical Systems: Exclude studies that do not focus on safety-critical systems, as the primary objective is to explore the impact of architecture on functional safety.
- EC2 Irrelevant Architectures: Exclude studies that discuss operating system architectures other than microkernel and monolithic, as these are the central architectures of interest.
- EC3 Non-Technical Content: Exclude studies that lack technical depth and do not provide substantial insights into the comparison of microkernel and monolithic architectures in safety contexts

4) Data extraction and systhesis: During the data-extraction step, we developed a data-extraction form that helped us extract relevant data from each selected primary study. We meticulously gathered and organized information from the selected studies. Subsequently, we engaged in an attentive synthesis process to distill and analyze the collective insights from these studies. We identified common themes, patterns, and trends across the literature, allowing us to draw meaningful conclusions and insights.

B. Conducting the review

We aimed to provide a rigorous and focused analysis of the topic study selection following a two-phase process, starting with a preliminary evaluation of titles, abstracts, and keywords. An in-depth analysis of full-text articles followed this. Data extraction and synthesis allowed us to distill critical insights and present a well-informed overview of the comparative aspects of the kernel and monolithic operating systems concerning functional safety and better fault containment. The approach ensures the reliability and validity of our systematic review, offering valuable contributions to the Operating system.

1) List the number of papers retrieved from digital libraries: To provide a comprehensive overview of the research landscape and to enable readers to access the primary sources, we have meticulously collected and organized the citations and bibliographic information for these papers into a BibTeX file labeled 'references.bib.'

These papers were reviewed and assessed against our predefined inclusion and exclusion criteria, which considered factors such as title, abstract, and keywords. This meticulous screening process allowed us to filter out papers that did not align with the objectives of our review. Subsequently, we identified a subset of [number] papers that demonstrated potential relevance to our systematic review.

The initial phase of our systematic literature review on "Kernel operating systems compared to monolithic operating systems: A review on functional safety" involved thoroughly exploring reputable digital libraries. This search endeavor aimed to identify and gather relevant scholarly works related to our research focus. Our diligent search process yielded a total of [number] papers retrieved from various esteemed digital libraries, including the ACM Digital Library, IEEE Xplore, and the Web of Science. The complete list of primary studies and their detailed information will be instrumental in the subsequent phases of our systematic review.

TABLE I
LIST OF AUTHORS OF BOOKS ON OPERATING SYSTEMS

ID	Author(s)	Reference Link
S1	Silberschatz, Galvin, and Gagne	[2]
S2	Crowley, Charles	[3]
S3	Andrew S. Tanenbaum, Herbert Bos	[4]

C. Threats to validity

The main validity threats of the conducted SLR are related to researcher study selection bias. The publication can be problematic because it may overestimate a phenomenon's actual effect size or significance, leading to inaccurate conclusions. To mitigate the threat of publication bias, researchers can employ techniques such as funnel plots, regression, or the trim-and-fill method to assess and adjust for potential bias in the included studies. Additionally, efforts to include unpublished studies or gray literature can help reduce the impact of publication bias on the overall findings and conclusions of a review. Moreover, to promote transparency and reproducibility, we comprehensively described the review protocol.

III. RESULTS, DISCUSSION, AND IMPLICATIONS

A. Demographics of the selected studies

1. **Case Studies:** Explore case studies across diverse industries to illustrate the application of microkernel and monolithic architectures in safety-critical systems. Examine notable projects in automotive, aviation, healthcare, and industrial automation, highlighting the architectural choices made and the resulting impact on safety and reliability.

Example: In the automotive sector, investigate the use of microkernels in the development of advanced driver-assistance systems (ADAS) and compare it with cases where monolithic architectures are employed. Analyze how these choices affect the safety performance and adaptability of the systems to evolving industry standards.

2. Industry Adoption: Examine the extent to which different industries have embraced microkernels or monolithic architectures in safety-critical applications. Investigate if there are industry-specific preferences based on factors such as system complexity, real-time requirements, and regulatory compliance.

Example: Evaluate the adoption patterns in the aerospace industry, considering the critical nature of avionics systems. Explore whether microkernels are more commonly used in flight control systems for fault isolation or if monolithic architectures dominate due to historical preferences

2. Industry Adoption: Examine the extent to which different industries have embraced microkernels or monolithic architectures in safety-critical applications. Investigate if there are industry-specific preferences based on factors such as system complexity, real-time requirements, and regulatory compliance.

Example: Evaluate the adoption patterns in the aerospace industry, considering the critical nature of avionics systems. Explore whether microkernels are more commonly used in flight control systems for fault isolation or if monolithic architectures dominate due to historical preferences

B. Topic 1 (RQ1)

Microkernel architectures and monolithic architectures represent two different design paradigms in operating system design, and their differences have implications for fault containment and propagation in safety-critical systems. Let's explore how the modular and isolated nature of microkernel architectures compared to monolithic architectures in safety-critical systems:

1. Modularity and Isolation in Microkernel Architectures:

a. Modularity:

Advantage: Microkernel architectures are inherently modular. Each component (such as device drivers, file systems, and network protocols) runs as a separate user-level process. This modularity means that if one component fails, it typically does not impact other components. Fault Containment: Faults in one module can be contained within that module, preventing them from affecting the entire system. This containment limits the impact of faults on the overall system's functionality.

b. Isolation:

Advantage: Microkernels enforce strict process isolation. Components communicate through well-defined inter-process communication (IPC) mechanisms. Processes cannot directly access each other's memory space. Fault Containment: Isolation ensures that a fault in one component does not corrupt the memory of another component. Isolated address spaces

prevent faults from propagating to unrelated parts of the system.

c. Flexibility:

Advantage: Microkernels can be customized for specific applications. Unnecessary components can be removed, reducing the attack surface and the potential impact of faults. Fault Containment: By reducing the codebase to only essential components, the system becomes more robust against faults, as there are fewer potential points of failure.

d. Dynamic Reconfiguration:

Advantage: Microkernels support dynamic loading and unloading of modules. Faulty modules can be replaced or restarted without bringing down the entire system. Fault Containment: Faulty components can be quickly identified, isolated, and replaced, minimizing system downtime and maintaining critical functionality

2. Monolithic Architectures:

a. Lack of Modularity:

Disadvantage: Monolithic architectures lack the modular structure of microkernels. Most components run in kernel mode, and the failure of one component can potentially crash the entire system. Fault Propagation: Faults in one part of the monolithic kernel can easily propagate to other parts since there is less isolation between components.

b. Limited Isolation:

Disadvantage: Monolithic kernels do not provide the same isolation level between components. All parts of the kernel share the same memory space. Fault Propagation: A fault in one module can corrupt the memory space shared by other modules, leading to the propagation of faults and potential system-wide failures.

C. Topic 2 (RQ2)

The modularity and isolation principles of microkernels have significant implications for integrating and verifying safety-critical third-party components compared to monolithic kernels

1. **Microkernels:**

a. Modularity:

Impact: Microkernels are inherently modular, allowing third-party components to operate as separate modules or processes. Integration is often easier because these components can be developed and tested independently without extensive knowledge of the entire system. Ease of Integration: Integrating third-party components is relatively straightforward due to the clear interfaces between modules. Developers can focus on implementing the specific functionality they need without affecting other system parts.

b. Isolation:

Impact: Microkernels enforce strict isolation between components. Third-party modules run in their own protected address spaces, reducing the risk of unintended interactions with the core system or other components. Ease of Verification: Isolation ensures that third-party components can be thoroughly tested and verified in isolation. Their behavior can be predicted and controlled, simplifying the verification process.

c. Inter-Process Communication (IPC):

Impact: Microkernels rely on well-defined IPC mechanisms for communication between components. Third-party components can communicate with the rest of the system through standardized interfaces. Ease of compatibility: As long as third-party components adhere to the IPC protocols, they can communicate seamlessly with other modules, ensuring compatibility and ease of integration.

2. Monolithic Kernels:

a. Lack of Modularity:

Impact: Monolithic kernels lack the clear module boundaries found in microkernels. Third-party components might need to interact deeply with the kernel, making integration more complex. Integration Challenges: Integration of third-party components can be challenging as they might need to be tightly coupled with the kernel's internal structures, leading to potential conflicts and difficulties in isolating issues.

b. Limited Isolation:

Impact: Monolithic kernels have limited isolation between components. Third-party modules might share the same memory space, making isolating and containing faults harder. Verification Complexity: Verifying third-party components in a shared memory space can be complex. Faults in one component can impact others, making it challenging to assess the component's behavior in isolation.

D. Topic 3 (RQ3)

Microkernels and monolithic kernels represent different architectural approaches to designing operating systems. The modularity and isolation principles of microkernels can significantly impact the ease of integrating and verifying safety-critical third-party components compared to monolithic kernels. Let's explore how these principles affect both types of kernels:

Microkernels:

Modularity: Impact on Integration: Microkernels are designed with a modular architecture where essential services such as device drivers, file systems, and network protocols run in user space as separate processes. This modularity allows for easier integration of third-party components. Since components operate in user space, they don't directly impact the kernel. They can be added or removed without affecting the core functionality.

Impact on Verification: The modular design facilitates individual verification of components. Safety-critical third-party components can be thoroughly tested and verified independently of the kernel. This isolation ensures that a failure in one component is less likely to affect others, enhancing the overall system reliability. Isolation:

Impact on Integration: Microkernels enforce strong isolation between components. Inter-process communication mechanisms are used to allow components to communicate without compromising the integrity of the kernel. This isolation ensures that safety-critical components can run alongside non-critical components without interference, simplifying integration efforts.

Impact on Verification: Isolation aids in the verification process. Safety-critical components can be tested and verified in isolation, reducing the complexity of the verification task. Since these components are decoupled from the kernel, they can be replaced or upgraded without affecting the entire system.

Monolithic Kernels:

Modularity:

Impact on Integration: Monolithic kernels, by design, have tightly integrated components. Adding or replacing components often requires modifying the kernel, making blending third-party components more challenging. Changing the kernel can impact stability and introduce compatibility issues with existing drivers and services.

Impact on Verification: The integrated nature of components in monolithic kernels makes it challenging to verify safety-critical components in isolation. Verifying a specific component might require considering its interactions with various parts of the kernel, complicating the verification process. Isolation:

Impact on Integration: Monolithic kernels have less inherent isolation between components. A third-party component with a flaw or bug could crash the entire system or cause other components to malfunction. Integrating safety-critical components without compromising the whole system's stability can be challenging.

Impact on Verification: Verifying safety-critical components becomes more complex due to the lack of strict isolation. Interactions with other kernel components must be considered during the verification process, making it harder to isolate and test specific functionalities.

IV. Conclusion

Microkernel architectures, with their modular and isolated nature, significantly enhance fault containment and propagation prevention in safety-critical systems compared to monolithic architectures. The modularity allows individual components to operate independently, limiting the impact of faults on specific modules without affecting the entire system. The strict isolation between components ensures that faults or failures in one module are contained and do not propagate to other parts of the system. These features are crucial in safety-critical systems where failures can have severe consequences.

Microkernel architectures offer advantages regarding functional safety due to their fault containment capabilities. Failures are localized, reducing the risk of systemic issues. This containment is essential for maintaining the overall functionality of safety-critical systems even in the presence of faults.

Moreover, microkernels' modularity and isolation principles greatly simplify the integration and verification of safety-critical third-party components. Independent verification of modules is possible, ensuring that safety-critical components can be thoroughly tested in isolation. This isolated testing and integration process enhances the reliability and safety of the entire system, making microkernels a preferred choice for safety-critical applications.

In summary, microkernel architectures, with their modularity and isolation principles, provide superior fault containment, prevent fault propagation, enhance functional safety, and ease the integration and verification of safety-critical components. These advantages make microkernels a robust and reliable choice for designing safety-critical systems.

A. Future Work

The rapid advancement of microkernel architectures is reshaping the landscape of safety-critical systems, ushering in a new era for professionals working in this domain. This paper explores the future of work in safety-critical systems, focusing on the transformative impact of microkernel architectures. This research anticipates a future where collaboration, innovation, and skillsets are redefined by analyzing current trends and emerging technologies, enabling engineers, researchers, and practitioners to create more reliable, fault-tolerant, and secure safety-critical applications.

This paper explores the implications of microkernel architectures in enhancing fault containment, preventing fault propagation, ensuring functional safety, and easing the integration of safety-critical components. Through a comprehensive analysis of current trends, challenges, and emerging technologies, this research paper provides valuable insights into the future of work for engineers, researchers, and practitioners involved in developing safety-critical systems.

References

[1] silberschatz-operating systems. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Mississippi state university, 07 2007.

[2] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley, 10th edition, 2018.

[3] Douglas Comer. *Operating System Design: The Xinu Approach*. CRC Press, 2nd edition, 2015.

[4] Andrew S. Tanenbaum. *Modern Operating Systems*. Pearson, 4th edition, 2014.

[5] Jean Labrosse. MicroC OS II: The Real-Time Kernel. *Awesome Journal*, 2002.