# GNU Hurd: Design, Implementation, and Future Prospects of the Microkernel-Based Operating System

Cameron C. Crowson, Ph.D.
Department of Computer Science, Texas A&M Commerce

**Abstract**

This literature review aims to conduct a comprehensive and critical analysis of the existing research on GNU Hurd, which is a microkernel-based operating system. The review focuses on aspects such as the design, implementation, advantages, disadvantages, complexities, and cost considerations of GNU Hurd. Moreover, it examines the involvement of companies in the GNU Hurd ecosystem and explores future trends along with the potential developments of the operating system. This is followed by a discussion of the theoretical and conceptual framework of GNU Hurd, including its object-based architecture and multi-server structure. The objectives of this review were achieved through a systematic literature search that encompassed relevant research databases, applying predefined selection criteria. Through the examination of theoretical foundations, practical applications, and potential implications of GNU Hurd, a holistic understanding of this innovative operating system has been formulated.

*Keywords*: GNU Hurd, microkernel operating system, design and implementation, object-based architecture, multi-server structure, extensibility and integration, complexity and cost, usability, future trends

**Table of Contents**

## List of Tables

# INTRODUCTION

## What is GNU Hurd?

GNU Hurd is a microkernel-based operating system designed to overcome limitations that afflict Unix and Linux. According to Walfield and Brinkmann [2007], GNU Hurd features an object-based architecture and interfaces that enhance extensibility, integration, and usability. Further, it is built upon the Mach microkernel and adopts a multi-server structure with a distributed naming framework [1]. Fault isolation is ensured in GNU Hurd by implementing objects as closures within user-space servers, while communication between these objects is facilitated through secure and unforgeable references called capabilities, which allows the use of canonical interfaces for tasks such as file system operations, I/O, authentication, and process management [1].

As part of the ongoing community project, Debian GNU/Hurd is an operating system that is currently being developed as a platform for development as well as server and desktop usage. Although it is still under active development and lacks the performance and stability of a production system, efforts are underway to port 75% of Debian packages to GNU/Hurd [2]. Furthermore, individuals can install Debian GNU/Hurd, familiarize themselves with the system, and engage with the mailing lists to understand the current state of development, along with offering assistance [2].

GNU Hurd aims to develop a versatile kernel that is suitable for the GNU operating system in order to empower users and programs by providing them extensive control over their computing environment. Its goal is to provide a practical solution for everyday use, thus ensuring that users can maximize their level of control and customization [3].

## Historical Timeline and Perspectives of GNU Hurd

The following timeline reveals several historical developments in the GNU Hurd microkernel system, providing a comprehensive outline of the project's evolution [4, 5]:

- 1983: Richard Stallman launches the GNU project with the goal of creating a complete operating system.
- Late 1980s: The GNU project is missing a suitable kernel for its operating system.
- 1986: The Free Software Foundation (FSF) starts working on implementing the changes needed in TRIX, a primitive kernel mentioned in the GNU Manifesto.
- 1987: Richard Stallman suggests developing a kernel based on the Mach microkernel, which would later be known as "Hurd."
- 1990: The FSF expresses interest in a multi-process kernel running on top of Mach, with the hopes of using it for the GNU system.
- 1991: The FSF announces that the Hurd running on Mach is GNU's official kernel.

- 1992: Linus Torvalds and Andrew Tanenbaum engage in a debate about microkernels versus monolithic kernels, with Linux eventually overshadowing Hurd.
- 1996: GNU Hurd 0.1 is released.
- 1997: GNU Mach 1.0, 1.1, 1.1.1, and 1.1.2 are released.
- 1998–2003: Development updates start, and personal views of Marcus Brinkmann on Hurd development are published.
- 2002: GNU MIG 1.3, libio-based glibc, GNU Mach 1.3, and Hurd L4 development starts.
- 2003: Crosshurd and Hurd/L4 developments start.
- 2005: Hurd/L4 at Libre Software Meeting
- 2007: FOSDEM participation, Hurd critique and position paper, Hurd on Xen, and more
- 2008: Successful Google Summer of Code (GSoC) projects and Hurd/Viengoos development
- 2009: GSoC projects on unionmount translator and Device Drivers in Userspace; significant progress in building Debian packages
- 2010: Arch Hurd, Nix port, DDE, GSoC projects, and increased package build percentage
- 2011: GNU Hurd 0.401 release, XKB development, attention from the press, GSoC projects, and increased package build percentage
- 2012: DDE progress, continuous testing with Nix, improved debugging, and more
- 2013: GNU Hurd 0.5, GNU Mach 1.4, and GNU MIG 1.4 are released.
- 2013: Debian GNU/Hurd 2013 (“wheezy”) is released, bringing new energy to the Hurd project.
- 2015: Debian GNU/Hurd 2015 (“jessie”) is released, marking another milestone for the Hurd project.
- 2016: GNU Hurd 0.8, GNU Mach 1.7, GNU MIG 1.7, GNU Hurd 0.9, GNU Mach 1.8, and GNU MIG 1.8 are released.
- 2017: Debian GNU/Hurd 2017 (“stretch”) is released, leading to further improvements and updates to the operating system.
- 2021: Debian GNU/Hurd 2021 (“bullseye”) is released, showcasing ongoing development efforts and providing an updated version of the Hurd operating system.

When discussing the evolution of microkernels into the 21st century, four key perspectives can provide additional context for the design, implementation, and future prospects of the microkernel-based operating system. These are as follows:

1. *Technical Perspective:* This perspective focuses on the technical aspects of a system and involves understanding how people and processes interact while managing different technical elements. It encompasses the structure of entities, roles, and processes, which include computational components such as storage, servers, and

databases. Further, it provides patterns for effectively using these components [6].

2. *Community Perspective*: This perspective stresses the importance of actively participating and leading in our respective communities. It recognizes the challenges of isolation and emphasizes the need to break free from self-centered concerns. By becoming community leaders, individuals can experience personal growth, develop skills, and make a positive impact on the world. Moreover, engaging in community work connects diverse individuals, fosters skill acquisition, and inspires others to create change [7].
3. *Historical Perspective*: This perspective involves understanding and interpreting the past by considering the social, cultural, intellectual, and emotional contexts that shape people's lives and decisions. It challenges contemporary assumptions by recognizing the differences between historical societies and our own. A historical perspective allows for a deeper understanding of human experiences, thus providing alternative viewpoints for the evaluation of current concerns [8].
4. *Future Perspective:* This perspective involves understanding as well as projecting the potential developments and trends that are expected to shape the future, particularly in the field of higher education. It relies on examining experiences of the recent past and incorporating insights from various experts to form an informed view of what lies ahead. Additionally, it helps anticipate and prepare for future changes in the educational landscape [9].

With the help of the aforementioned timeline, we can observe the historical unveiling of the GNU Hurd microkernel system. This project began in 1983, and its evolution is still ongoing into the 21st century. The timeline also discusses the technological efforts made as part of the ongoing development, such as the release of GNU Hurd, GNU Mach, and GNU MIG. Moreover, as part of the community involvement initiative, the Debian GNU/Hurd distributions were released in 2013 and continue to the present day. We can consider the future trends of the GNU Hurd project on the basis of its current uses in a collection of servers, network protocols, file access control, and other features through integration into Unix and Linux systems [3].

**Purpose of the Study**

This literature review aims to conduct a comprehensive and critical analysis of the existing research on GNU Hurd. The review is concerned with (1) identifying and defining current knowledge concerning GNU Hurd; (2) exploring the design, implementation, and functions of GNU Hurd; (3) evaluating the advantages, disadvantages, complexities, and cost of implementing GNU Hurd; (4) examining the involvement of various

companies in the use of the GNU Hurd system; and (5) identifying the future trends and potential developments of GNU Hurd.

**Research Questions**

The main research question is as follows: “What is the current state of the knowledge, design, implementation, advantages, disadvantages, and future trends concerning GNU Hurd as a microkernel-based operating system?”

This leads us to consider the following sub-questions:

1. How does the design and implementation of GNU Hurd, with its object-based architecture and multi-server structure, overcome the limitations that afflict the Unix and Linux systems?
2. What are the key subcomponents of GNU Hurd?
3. What are the complexities and costs associated with implementing GNU Hurd; moreover, what are the advantages and disadvantages of using this microkernel-based operating system?
4. How do various companies contribute to the GNU Hurd ecosystem, and what is their involvement in developing and supporting the operating system?
5. What are the future trends and potential developments in GNU Hurd, and what advancements can be expected in terms of its functionality, stability, and performance?

**Vocabulary: Key Terminology and Definitions**

- *Canonical Interfaces*: This is a standardized mechanism that allows diverse data models or systems to integrate and interact. It provides a common vocabulary and functions to map and manipulate data across different databases, ensuring interoperability and facilitating information exchange [10].
- *Debian GNU Hurd*: Debian GNU/Hurd is an operating system created by the Debian project that employs the GNU Hurd suite of servers on the GNU Mach microkernel as its foundation, which distinguishes it from Debian GNU/Linux. It strives to provide an extensive array of software applications and establish compatibility with Debian GNU/Linux in due course [11].
- *Extensibility*: Extensibility refers to a system’s ability to expand and enhance its functionality. Unlike Unix, which lacks extensive support for system call interception and modification, the GNOME and KDE projects have developed separate interfaces to provide users with a more integrated experience to overcome these limitations [12].
- *Fault isolation*: This approach involves confining untrusted code to a specific section of memory known as the fault domain. It restricts the code’s ability to affect or interfere with other parts of the system, effectively containing any potential faults or errors within the isolated boundaries [13].

- *GNU Hurd*: GNU Hurd is a kernel created by the GNU project to serve as a replacement for the Unix kernel. It comprises a collection of servers that operate on the Mach microkernel, thereby enabling the implementation of various functionalities such as file systems, network protocols, file access control, and other features that are commonly associated with Unix or Linux kernels [3].
- *Mach Microkernel*: The GNU Mach microkernel forms the core of a GNU Hurd system, providing a means for the Hurd to establish communication between processes and define interfaces for the implementation of essential operating system services using a distributed, multi-server approach. GNU Mach 1.8, which is maintained by the GNU project's Hurd developers, is the latest release. It is compatible with x86 machines and incorporates Linux 2.0 device drivers through the glue code [14]
- *Microkernel*: This refers to a streamlined kernel design approach that focuses on minimizing the core components of an operating system while maximizing implementation flexibility. It originated in the work of Per Brinch-Hansen and has since been explored further by researchers such as Liedtke, Úlfar Erlingsson, and Athanasios Kyparlis, all of who have contributed to the understanding and development of microkernel concepts [15].
- *Multi-server Structure*: This refers to an architectural design where multiple servers collaborate to perform the functions that are typically handled by a single monolithic kernel. This setup enables efficient resource management, real-time capabilities, and quality service. However, it also necessitates mechanisms to account for resources, prevents denial-of-resource attacks, and introduces the concept of resource containers for secure allocation and recovery. These considerations stem from the distributed nature of a multi-server system, the requirement for accurate prediction and control of resource access, and the need to ensure secure and controlled resource allocation at a global scale [1].
- *Object-based architecture*: Object-based architecture refers to a design approach adopted by the Hurd operating system that is characterized by its ability to evolve and adapt while maintaining its core principles. This architecture allows Hurd to undergo significant changes and enhancements without requiring a complete rewrite. Furthermore, it offers compatibility with existing programming and user environments, scalability for efficient performance on various hardware configurations, extensibility for kernel modifications and experimentation, stability through isolated kernel component development, and availability as a functional and usable software in the present [16].
- *Porting*: This entails modifying and adapting software programs from the Debian archive to ensure their successful compilation and functionality on the Hurd platform. It involves resolving build failures,

analyzing errors, collaborating through mailing lists and task trackers, and utilizing available guidelines and resources for effective porting [17].

- *Usability*: This refers to its practicality and functionality for everyday use. While it offers a complete and usable experience, it may not be suitable for production due to existing bugs and missing features, even though it serves as a solid foundation for further development and non-critical application usage [18].

**Nature of the Study**

This literature review hopes to provide a comprehensive and critical analysis of the existing research on GNU Hurd. This study examined a range of scholarly works, technical documents, and relevant sources with the aim of synthesizing and evaluating the available literature to gain a thorough understanding of GNU Hurd. The analysis encompasses various aspects of GNU Hurd, including its goals and purposes, the standards and platform governing it, its design and implementation approaches, its subcomponents and their functions, its complexity and cost considerations, and its associated pros and cons. Furthermore, a thorough analysis was performed concerning the theoretical and conceptual frameworks, encompassing relevant theories and applications related to the microkernel architecture and structure, the existing extensibility and integration, and user empowerment.

**Summary**

GNU Hurd is a microkernel-based operating system that aims to overcome the limitations that afflict Unix and Linux by adopting an object-based architecture and a multi-server structure. Although it offers extensibility, fault isolation, and canonical interfaces, it is still under active development and lacks the stability and performance of a production system. However, it serves as a solid foundation for further development and non-critical application usage, with ongoing efforts to port Debian packages to GNU/Hurd and provide a practical solution for everyday use.

The rest of the paper provides an overview of the design and implementation of GNU Hurd, including the approaches and techniques used. Moreover, it delves into the subcomponents of GNU Hurd such as GNU Mach, memory manager, process manager, inter-process communication (IPC), dynamic data exchange (DDE) Kit, device driver development libraries, device driver framework, Hurd servers, GNU Hurd libc, system console, and translators in order to provide insights into their functions and interconnections. The complexities and costs associated with the implementation of GNU Hurd are discussed, followed by an analysis of the pros and cons associated with the system. The involvement of various companies in the GNU Hurd ecosystem is also explored. Furthermore, future trends of GNU Hurd are also examined, highlighting potential

developments and advancements. Finally, the paper concludes with a summary of the findings.

## LITERATURE REVIEW

The literature review has a twofold objective, which is to synthesize as well as analyze the existing research on the theoretical and conceptual framework of GNU Hurd. This review encompasses a wide range of aspects related to GNU Hurd, delving into the goals and purposes of GNU Hurd, the standards and platform that govern its operations, and the various design and implementation strategies and techniques employed by it. Further, this review examines the functions of the subcomponents within GNU Hurd, exploring their interconnections and the support they provide to the system as a whole. It also explores the complexities and cost considerations associated with the implementation of GNU Hurd by identifying the advantages as well as the disadvantages of adopting this system.

Furthermore, this literature review investigates the involvement of companies in the GNU Hurd ecosystem, along with the potential future trends that may shape the development and adoption of GNU Hurd. This comprehensive examination encompasses not only the technical aspects of GNU Hurd but also the communal, historical, and future perspectives surrounding this innovative operating system. Thus, by exploring these multiple dimensions, the review aims to provide a holistic understanding of GNU Hurd, shedding light on its theoretical foundations, practical applications, and potential implications for the broader computing landscape.

### Literature Search Procedure

The literature search procedure employed for this review ensured a systematic and comprehensive approach to gathering relevant sources. For this, the following steps were implemented:

1. *Scope and Timeframe*: A clear scope of the study was defined to determine the boundaries of the literature search. The timeframe for selecting relevant technical literature was carefully established to include the most recent and influential publications in the field.
2. *Keywords and Phrases*: Appropriate keywords and phrases were identified to effectively search research databases and retrieve relevant scholarly articles, technical reports, and other valuable sources. The selection of keywords and phrases was based on a thorough understanding of the research topic and the terminology commonly used in the domain of GNU Hurd.
3. *Research Databases*: A range of reputable research databases, relevant to the field of GNU Hurd, were selected for the literature search. These databases provide access to a wide array of scholarly resources, which ensures a comprehensive coverage of the literature.

4. *Selection Criteria*: Specific criteria were predefined to govern the selection of literature for this review. These criteria encompass various factors to ensure the inclusion of high-quality and relevant sources. Considerations included the alignment with the research topic, the robustness of the research methodology employed, the credibility and reliability of the sources, and the currency of the publication. By adhering to these well-defined criteria, the literature review guarantees the inclusion of scholarly works that contribute significantly to the theoretical and conceptual framework of GNU Hurd.

To identify relevant literature on the topic, the researcher conducted a thorough search across specified online databases such as IEEE Xplore, ACM Digital Library, ScienceDirect, Google Scholar, ACM SIGOPS Operating Systems Review, the official GNU Hurd website, and Usenix Association. The search was guided by carefully chosen keywords and phrases, such as “theoretical framework of microkernel operating systems,” “standards and platforms governing GNU Hurd,” “design and implementation of GNU Hurd,” “implementation approaches and techniques,” “subcomponents of GNU Hurd and their functions,” “complexity and cost considerations in implementing GNU Hurd,” “pros and cons of GNU Hurd,” “companies involved in the GNU Hurd ecosystem,” and “future trends of GNU Hurd.”

The search process encompassed a diverse range of scholarly research articles, technical reports, and other valuable sources, contributing to a comprehensive understanding of the microkernel-based operating system. It is important to note that this rigorous approach has been consistently applied throughout all subsequent sections of this literature review.

**Theoretical and Conceptual Framework**

The theoretical framework of GNU Hurd encompasses a set of core principles and conceptual foundations that define its design, development, and operation. This framework addresses limitations in Unix by adopting the following [1]:

a) an object-based architecture
b) a multi-server structure
c) enabling extensibility and integration
d) user empowerment

Thus, the theoretical framework of GNU Hurd aims to overcome the limitations of traditional operating systems and serve as a foundation for understanding its standards, implementation approaches, subcomponents, complexities, and future trends [1]:

### *Object-Based Architecture*

In the early to mid-1980s, the concept of object-based architecture emerged as a means to reduce execution overhead in object-oriented languages. Dally and Kajiya [1985] proposed that this architecture incorporates hardware mechanisms and abstract instructions to provide instruction safety, late binding, and extensibility to optimize the execution of object-oriented code [19]. Moreover, the proposed architecture introduces abstract instructions, floating-point addresses, and hardware support for context allocation [19]. As for the implementation process, this architecture utilizes a communication-oriented processor (COM), which is a specialized processor optimized for communication systems, after considering factors such as data rates, algorithm complexity, adaptability, and support for various interfaces and applications, along with a specific focus on the challenges in the physical layer design [20].

The object-based architecture offers a raised level of hardware and software interface, improved software design methodology, integration of system management and control, flexibility and extensibility, enhanced program environments and intercommunication, efficient object addressing and protection, and object-oriented instruction interface [21] Because of the proposed benefits, several applications have adopted object-based architecture due to its ability to offer a high-level description, support for a variety of capabilities, and serve as a basis for system protection [22]. These include companies such as IBM System 38, Carnegie-Mellon experimental C.mmp/Hydra, and Intel iAPX 432.

### *Multi-Server Structure*

The multi-server architecture involves the replication of content across multiple servers, with the replication process including the downloading of modified files from the parent server to subordinate servers [23]. In contrast, the single server configuration consolidates all server components on a single computer, where clients connect through a listener that forwards requests to the appropriate server components, offering options for load balancing at the listener or read/write instances [24]. Moreover, the choice between the two structures heavily depends on factors such as scalability, resource utilization, and the specific requirements of the environment.

As for the examples of the utilization of a multi-server structure, the City University of Hong Kong described the uses of a multi-server structure in their application called CyberWalk. This application utilizes adaptive data partitioning, allowing multiple servers to handle increasing user numbers in a distributed virtual walkthrough system by managing specific regions of the virtual environment and dynamically balancing the load [25]. Another example was proposed by Zhang et al. [2023], which involves the use of a multi-server-assisted data-sharing structure that is utilized for a metaverse healthcare system [26]. This system involves the use of multiple servers to

assist in encryption and decryption operations, providing several benefits such as reduced computational overhead, independent encryption overhead, and assisted encryption and decryption [26].

Overall, the benefits of a multi-server structure are evident in these examples. It offers scalability to accommodate growing user numbers, load balancing to ensure efficient distribution of workload, fault tolerance to maintain system availability, performance optimization through specialized server assignments, encryption options for enhanced security, modular and flexible design for easy integration and maintenance, and improved overall system security.

### *Extensibility and Integration*

Extensibility refers to the ability of a technology or software system to seamlessly integrate new elements and features without significant modifications to its existing structure [27]. Conversely, integration involves combining system components into a unified and interconnected framework, facilitating efficient resource sharing, extensibility, and the flexibility to replace or extend system services and, consequently, enhancing usability and promoting collaboration [27].

### *User Empowerment*

User empowerment is a fundamental principle for the creation of a highly user-friendly system that emphasizes the active involvement of individuals with disabilities in all stages of technology design and development, surpassing their mere participation in user studies [28]. By prioritizing user empowerment, the goal is to establish a comprehensive framework that fosters inclusivity and empowerment for all users.

## Goals and Purposes of GNU Hurd

The following key aspects were identified while addressing the goals and purposes of GNU Hurd:

1. *Overcoming Limitations*: GNU Hurd aims to surpass the limitations that afflict traditional Unix and Linux systems. By adopting an object-based architecture and a multi-server structure, it strives to enhance extensibility, integration, and usability.
2. *Extensive Control and Customization*: GNU Hurd seeks to empower users and programs by providing extensive control over their computing environment. It endeavors to offer practical solutions that enable users to maximize their control and customization options.
3. *Practical Everyday Usage*: While GNU Hurd is still under active development and may not yet possess the stability and performance of a production system, it aspires to provide a complete and usable experience for non-critical applications. Efforts are underway to port Debian packages to GNU/Hurd to make it suitable for everyday use.

4. *Community Involvement*: GNU Hurd is developed as part of an ongoing community project. Users can install Debian GNU/Hurd, participate in mailing lists, and contribute to the development efforts. The project encourages active engagement and collaboration within the community.

**Subcomponents of GNU Hurd and their Functions**

In Mignot's [2005] talk, the microkernel-based design of the GNU Hurd system is described, comprising the following subcomponents [29]:

1. *Microkernel (GNU Mach):* This forms the foundation of the GNU Hurd system, thereby providing essential services such as task management, memory management, IPC, basic I/O primitives, and device drivers.
2. *Servers:* These include the IPC Server for inter-process communication, Scheduler Server for task scheduling and resource allocation, Memory Server for memory management, File System Servers for file operations and access, Network Servers for networking, and additional potential servers for various functionalities.
3. *Libraries:* These are used to enhance the capabilities and services accessible to user programs, enabling easier interaction with servers and utilization of system resources and, thus, benefiting the applications running on the system.
4. *Interfaces:* These are the components of the system that act as communication protocols and application programming interfaces (APIs), facilitating seamless interaction between various components such as servers, libraries, and user programs, thereby ensuring the smooth and efficient operation of the system.
5. *Translators:* These are vital components of the GNU Hurd system that serve as intermediaries between user programs and the underlying resources by associating with nodes in the virtual file system (VFS). They play a crucial role in handling requests, translating them into suitable actions, and granting access to specific servers or services based on the requested file or resource.

Mignot [2005] argued that this microkernel-based design approach offers enhanced modularity, flexibility, and user freedom while also effectively addressing the limitations and drawbacks associated with monolithic kernel-based systems [29].

**Interconnections and Support among GNU Hurd Subcomponents**

The GNU Hurd is an operating system designed with a microkernel-based architecture. It follows a modular approach where essential services are separated into a minimalistic kernel, whereas most functionalities are implemented as user-space servers. The system offers two design

approaches: mono-server systems, where a single server handles all kernel functions, and multi-server systems, where features are split into communicating processes [29]. Moreover, GNU Hurd emphasizes flexible access control, secure collaboration, and persistence. It achieves this through the use of translators to extend the namespace, as well as the mechanisms for identity-based access control and system state restoration [1].

The design and implementation of GNU Hurd are supported by a robust network of interconnections established through the use of a comprehensive set of interfaces, procedures, and protocols. These components seamlessly enable a wide range of functionalities, including access control, identity management, device drivers, file system operations, and process management. Furthermore, crucial procedures such as distributed naming hierarchy, exec_server, exec_thread, ioctl_handler, and standard_exec_entry play indispensable roles in establishing naming hierarchies, executing programs, managing threads, handling specific IOCTL operations, and managing program entry points. Additionally, protocols such as authentication, identity-based access control, ioctl_handler, reverse authentication, remote procedure call (RPC), and translator are instrumental in ensuring secure user/process verification, identity-based access control, efficient handling of mobile code, inter-program communication, and seamless translation between different naming and access conventions.

### *Interfaces*

Table 1 presents the comprehensive analyses conducted by Walfield and Brinkmann [2007] and Hammar [2010], outlining the essential GNU Hurd interfaces that significantly contribute to the functionality, management, and interaction within the microkernel-based operating system [1, 30]. From a thorough analysis of this research, these interfaces can further be classified into six distinct categories, namely, access control, file system, input/output, process management, system fundamentals, and name resolution. These categories effectively illustrate the diverse aspects of system functionality, encompassing access control, file system operations, input/output operations, process management, system fundamentals, and name resolution within the GNU Hurd microkernel operating system [1, 30].

Table 1.
*List GNU Hurd Interfaces*

| GNU Hurd interface | Explanation |
|---|---|
| Access control | Mechanisms for regulating and managing the permissions and privileges associated with accessing system resources |
| Auth interface | Used for managing identities and supporting identity-based access control |

| | |
|---|---|
| Device drivers | Interfaces for interacting with specific devices |
| Dir lookup interface | Used to resolve names relative to referenced objects, such as the root directory or current working directory |
| Exec interface | Used for instantiating programs |
| fs interface | Used for examining as well as manipulating directory and file metadata |
| fsys interface | Used for whole file system operations and obtaining file handles |
| IO interface | Used for reading from data sources and writing to data sinks |
| IO control (ioctl) | A system call interface that provides device-specific operations to files that are hard to express through normal file operations |
| Mach 3 Kernel Interface | Encompasses the set of interfaces and protocols through which user-level processes interact with the Mach 3 microkernel, including task ports, access control mechanisms, and inter-process communication protocols |
| Process interface | Used for process management, including process identifiers (PIDs) and signal delivery |
| Virtual filesystem | An abstraction layer that provides a unified view of different filesystems, enabling consistent access and manipulation of directory and file metadata |

Note. Adapted from *A Critique of the GNU Hurd Multi-Server Operating System* by Walfield and Brinkmann [2007] and *Generalizing Mobility for the Hurd* by Hammar [2010] [1, 30]

### ***Procedures***

Table 2 presents the comprehensive analyses conducted by Walfield and Brinkmann [2007] and Hammar [2010], highlighting the crucial GNU Hurd procedures that contributed significantly to the system's functionality [1, 30]. These procedures can be categorized as file system management, process management, threading and concurrency, device and driver management, or program execution. They play a vital role in managing naming hierarchies, executing programs, handling program execution within threads, managing device-specific operations, and facilitating the entry point of executable programs within the GNU Hurd operating system [1, 30].

Table 2.
*List of GNU Hurd Procedures*

| GNU Hurd procedure | Explanation |
|---|---|

| | |
|---|---|
| Distributed naming hierarchy | Used to create a commonly understood naming hierarchy without special privileges |
| exec_server | A procedure responsible for executing programs |
| exec_thread | A procedure that handles the execution of a program within a thread |
| ioctl_handler | A procedure or function responsible for handling a specific range of IOCTL operations; used in the GNU C Library to handle IOCTL requests that cannot be converted into RPCs |
| standard_exec_entry | A procedure that handles the entry point for an executable program |

Note. Adapted from *A Critique of the GNU Hurd Multi-Server Operating System* by Walfield and Brinkmann [2007] and *Generalizing Mobility for the Hurd* by Hammar [2010] [1, 30]

### *Protocols*

Table 3 presents the comprehensive analyses conducted by Walfield and Brinkmann [2007], Mignot [2005], and Hammar [2010], outlining the essential GNU Hurd protocols [1, 29, 30]. These protocols significantly contribute to various aspects of the GNU Hurd system, including secure authentication, user validation, identity-based access control, handling of mobile code IOCTL operations, communication between client programs, and translation between different naming and access conventions. These protocols and procedures are categorized as authentication and access control, device and driver management, communication and interaction, or naming and translation. These play a crucial role in ensuring security, efficiency, and seamless functionality within the GNU Hurd system [1, 29, 30].

Table 3.
*List of GNU Hurd Protocols*

| GNU Hurd protocol | Explanation |
|---|---|
| Authentication protocol | A set of rules and procedures for verifying the authenticity and validity of users or processes in a secure manner |
| Identify-based access control protocol | Supports identity-based access control (IBAC) and secure collaboration between entities |
| ioctl_handler protocol | A specific protocol introduced in the implementation of mobile code IOCTL handlers in the Hurd; includes three new messages in addition to the four |

| | |
|---|---|
| | messages used by the standard authentication protocol |
| Reverse authentication protocol | A protocol used in the implementation of the mobile code IOCTL handlers; involves obtaining a port to a file's IOCTL handler module using a series of messages exchanged between the client and the server |
| RPC | Used as a way to enable client programs to request tasks from several programs and receive responses, with stub-code generators simplifying the development process by handling the encoding and decoding of parameters and results |
| Translator protocol | Allows the linking of translators, which translate between different naming and access conventions |

Note. Adapted from *A Critique of the GNU Hurd Multi-Server Operating System* by Walfield and Brinkmann [2007] *The GNU Hurd* by G.L. Mignot [2005], and *Generalizing Mobility for the Hurd* by Hammar [2010] [1, 29, 30].

## Complexities and Costs of Implementing GNU Hurd

When discussing the complexities and costs associated with implementing GNU Hurd, it is important to consider several key factors. GNU Hurd is primarily developed and maintained by the FSF, with contributions from a dedicated community of developers, enthusiasts, and organizations who are passionate about free software and open-source principles [31]. Moreover, GNU Hurd follows an open-source model, which means that the development process is characterized by transparency and community involvement. Besides, the open nature of the project fosters innovation and encourages the sharing of ideas and solutions. It is worth noting that the FSF and the wider community strive to minimize costs by leveraging existing infrastructure and collaborative tools [31].

As regards the costs of implementing GNU Hurd, this mainly revolves around the essential resources required to support the development process. The utilization of the object-based architecture framework, multi-server structure, and the system's capacity to extend and integrate with other frameworks may also contribute to these costs. According to Rattner and Cox [1980], the object-based architecture framework is known to emphasize integrated data abstraction and domain-based protection by utilizing capability-based addressing and supporting high-level system functions [21]. Furthermore, promoting effective software design methodology enhances reliability and reduces the overall cost of the software's life cycle [21]. All this contributes to a lower cost and

comprehensive software design methodology. The cost arises from the association with rich abstractions, while the cost associated with the rich abstractions provided in GNU Hurd arises from the potential circumvention of these abstractions for flexibility or efficiency [1].

According to Walfield and Brinkmann [2007], the implementation of a multi-server structure in operating systems introduces costs related to message passing, resource scheduling, and accounting, whereas the absence of transparent resource management and real-time support in such systems can result in inefficient resource utilization, reduced usability, and potential security vulnerabilities [1]. Consequently, the perceived cost of utilizing a multi-server structure is considered to be significant [1].

## Pros and Cons of GNU Hurd

An extensive literature review was carried out to formulate an inclusive compilation of the strengths and weaknesses of the GNU Hurd system. The resulting list is presented in Table 4, which provides an overview of the fundamental pros and cons associated with this microkernel-based operating system. The next sections of this study explore the strategies for addressing the identified drawbacks.

Table 4.
*List of the Basic Pros and Cons of the GNU Hurd System*

| Pros | Cons |
|---|---|
| Community-based development | Cost considerations for multi-server structure |
| Comprehensive design methodology | Compatibility challenges |
| Extensibility and integration | Cost of rich abstractions |
| Open-source (free software) | Dependency on community support |
| Performance of multi-server structure | Inefficient resource utilization |
| Object-based architecture | Still in the development stages |
| Overcoming limitations with Linux | Limited adoption |
| User empowerment with the system | Some performance issues |

Note. Adapted from a comprehensive review of the literature (various scholarly articles published between 1980 and 2023)

## Future Trends of GNU Hurd

The GNU Hurd microkernel operating system showcases its versatility and potential across diverse industries by effectively addressing identified drawbacks. These applications not only utilize GNU Hurd but also leverage its strengths to enhance its performance and overcome its limitations. Tsai

et al. [2014] proposed that incorporating software solutions such as Library OSes and Graphene presents a promising approach for efficient application execution, which offers advantages such as security isolation and reduced memory usage [32]. These solutions align with the goals of GNU Hurd, which aims to optimize performance and support multi-process APIs [32]. Moreover, applications such as myThOS strive to follow a similar structure to GNU Hurd by emphasizing efficiency, parallelism, adaptability, and effective utilization of hardware resources [33].

King and Wilson [34] compared the evaluation and usage of object-oriented languages in the context of evaluating RAID systems. This study highlights the significance of GNU's flexible and open-source nature in assessing the performance and functionality of complex algorithms such as Yex, which is a novel solution for the emulation of IPv7 that addresses the challenges of distributed communication as well as demonstrates the importance of networking in large-scale configurations [34].

Therefore, the future trend of microkernel-based operating systems is focused on enhancing versatility and performance while addressing the identified limitations. This includes leveraging software solutions such as Library OSes and Graphene for efficient application execution, emphasizing security isolation, reduced memory usage, and support for multi-process APIs.

## DISCUSSION, IMPLICATIONS, AND RECOMMENDATIONS

This section provides a concise summary of the literature review and explores its implications while offering recommendations for the GNU Hurd system. The section encompasses the following key topics: (a) summarizing the findings derived from the literature review, (b) interpreting the significance of these findings, (c) providing recommendations for the advancement of GNU Hurd, and (d) concluding the study. These aspects have been addressed to gain a comprehensive understanding of the GNU Hurd system and its potential implications.

### Summary of Findings

The purpose of this literature review was to conduct a comprehensive and critical analysis of the existing research on GNU Hurd. The review aimed to achieve the following objectives: (1) identify and define the existing literature on GNU Hurd; (2) explore the design, implementation, and functionalities of GNU Hurd; (3) evaluate the advantages, disadvantages, complexities, and implementation costs associated with GNU Hurd; (4) examine the involvement of various companies in utilizing the GNU Hurd system; and (5) identify future trends and potential developments related to GNU Hurd.

This study attempted to enhance our understanding of the theoretical and conceptual framework of GNU Hurd. The investigation yielded valuable insights into the goals and purposes of GNU Hurd, the subcomponents of

the system and their respective functions, the interconnections and support mechanisms among these subcomponents, the intricacies and costs involved in implementing GNU Hurd, the companies that are actively engaged in its utilization, and the pros as well as cons associated with GNU Hurd. Additionally, the review aimed to shed light on the future trends and potential advancements within the GNU Hurd ecosystem.

**Interpretation of the Findings**

This review underscores the significance of extensibility, integration, and user empowerment within the GNU Hurd framework, highlighting how these factors contribute to its effectiveness and adaptability. Collectively, the findings of this review contribute to a comprehensive interpretation of GNU Hurd and its importance in the field of operating systems.

This review also offers a critical analysis of the existing research in this area in response to the following research questions addressed in this study:

1. How does the design and implementation of GNU Hurd, with its object-based architecture and multi-server structure, overcome the limitations that afflict Unix and Linux?
2. What are the key subcomponents of GNU Hurd?
3. What are the complexities and costs associated with implementing GNU Hurd, and what are the advantages and disadvantages of using this microkernel-based operating system?
4. How do various companies contribute to the GNU Hurd ecosystem, and what is their involvement in the development and support of the operating system?
5. What are the future trends and potential developments in GNU Hurd, and what advancements can be expected in terms of its functionality, stability, and performance?

In this literature review, these research questions have been critically analyzed, providing valuable insights into each aspect. The review (1) offers a comprehensive evaluation of how GNU Hurd's design and implementation overcome limitations found in Unix and Linux systems; (2) identifies and explores the key subcomponents of GNU Hurd; (3) discusses the complexity, cost, advantages, and disadvantages associated with implementing this microkernel-based operating system; (4) examines the involvement of various companies in the GNU Hurd ecosystem; and (5) explores the potential future trends and advancements in terms of functionality, stability, and performance.

First, the design and implementation of GNU Hurd with its object-based architecture and multi-server structure overcome the limitations that afflict the Unix and Linux systems by providing greater flexibility and extensibility [1, 21, 34]. The microkernel design allows for better modularity and isolation of components, leading to improved fault tolerance and easier

debugging. Further, the multi-server structure enables different services to run independently, allowing for better resource utilization and scalability [5]. Additionally, the object-oriented approach facilitates easier development and maintenance of the system [21, 34].

Second, the key subcomponents of GNU Hurd include the microkernel (Mach), the GNU servers, and the Hurd-specific utilities. The microkernel provides essential abstractions and low-level services, whereas the GNU servers handle higher-level functionality such as file systems, networking, and device drivers [1, 3]. Further, the Hurd-specific utilities provide tools for system administration and user interactions [3, 28].

Third, implementing GNU Hurd can be complex and costly due to its unique design and relatively limited user base. The microkernel architecture requires careful design and implementation, and the development of compatible GNU servers can be a time-consuming process [1]. Moreover, since GNU Hurd is not as widely used as Unix or Linux, there may be a lack of readily available documentation, support, and software compatibility. However, the advantages of using GNU Hurd include its flexibility, extensibility, and potential for customization. It also offers a unique development platform and can be tailored to specific needs.

Fourth, various individuals contribute to the GNU Hurd ecosystem through development, support, and collaboration. Because this is an open-source software, individuals may provide financial support to the GNU project or allocate resources to contribute code and improvements [31]. Additionally, individuals that rely on GNU Hurd for their software applications may have dedicated teams working on the development and maintenance of the operating system. Besides, the involvement of companies can help drive the development of GNU Hurd, bring in expertise from different domains, and increase the overall adoption of and support for the operating system [31].

Finally, advancements in functionality, stability, and performance can be expected through ongoing development efforts. This could include improvements in the compatibility with existing software, enhancements to the Hurd-specific utilities, and optimizations in the microkernel and GNU servers. Moreover, the development community may also focus on addressing user feedback, security vulnerabilities, and expanding the ecosystem of supported hardware and software [25, 32, 34]. This has been explored in the leveraging software solutions such as myThOS, Library OSes, Graphene, and Lex, as well as in applications such as the metaverse healthcare system.

**Recommendations for GNU Hurd**

To ensure the long-term success and competitiveness of GNU Hurd, attention should be paid to software and hardware design aspects. Moreover, addressing security vulnerabilities and ensuring the implementation of robust security measures are crucial. By learning from the practices employed by popular operating systems such as Apple OS and

Windows OS, GNU Hurd should invest in software solutions that support mainframe vulnerabilities, thereby enhancing the reliability and trustworthiness of the system's software and hardware communication. Furthermore, efforts should be made to expand hardware and software compatibility. By working closely with hardware manufacturers and developers, GNU Hurd can ensure that a wider range of devices and applications can effectively utilize the system. This expansion will not only increase the overall usability and market adoption of GNU Hurd but also stimulate innovation and development within the ecosystem.

Additionally, GNU Hurd can greatly benefit from improvements in documentation, support, and collaboration. To enhance the usability of the microkernel-based operating system, comprehensive documentation and user-friendly resources should be provided. Clear and well-documented instructions can help users navigate the system effectively and troubleshoot issues more easily. Further, GNU Hurd should actively seek collaboration with companies and organizations to bring in diverse perspectives and resources. While bringing individual developers can contribute to the ecosystem, establishing partnerships with companies and organizations can lead to additional expertise and funding. Collaborative efforts can lead to the development of new features, improved compatibility with existing software, and the expansion of hardware support. Besides, by fostering a collaborative environment, GNU Hurd can benefit from a wider range of perspectives, accelerate development, and attract more users and contributors to the system.

By focusing on these areas of improvement in documentation, support, collaboration, software design, and hardware compatibility, GNU Hurd can strengthen its position as a reliable and competitive microkernel-based operating system, meeting the needs of users and attracting a broader community of developers and supporters.

## Conclusion

In conclusion, this study has provided a comprehensive analysis of GNU Hurd, along with its design, implementation, functionalities, and potential implications. The literature review yielded valuable insights into the goals and purposes of GNU Hurd, the subcomponents of the system and their functions, the advantages and disadvantages associated with GNU Hurd, the involvement of various companies in its utilization, and the future trends and potential advancements within the GNU Hurd ecosystem.

The findings of this study highlight the significance of GNU Hurd's design and implementation, which overcome limitations found in Unix and Linux through its object-based architecture and multi-server structure. Further, the study identified key subcomponents of GNU Hurd, which include the microkernel, GNU servers, and Hurd-specific utilities. It also provided insights into the complexities and costs associated with implementing GNU Hurd, along with the advantages and disadvantages of

using it. Additionally, the study examined the involvement of various companies in the GNU Hurd ecosystem and explored potential future trends and advancements.

Overall, the GNU Hurd system incorporates a unique and promising structure for the development of microkernel-based operating systems by offering flexibility, extensibility, and user empowerment. With continued development and support, GNU Hurd has the potential to make significant contributions to the field of microkernel-based operating systems and provide users with greater control over their computing environments.

## REFERENCES

[1] N. Walfield and M. Brinkmann, 2007. A critique of the GNU hurd multi-server operating system. *Operating Systems Review* 41, 4 (July, 2007), 30–39. https://doi.org/10.1145/1278901.1278907

[2] Debian, 2019. Debian GNU/Hurd. (March, 2019). https://www.debian.org/ports/hurd/

[3] GNU Hurd, 2016. GNU Hurd. (December, 2016). https://www.gnu.org/software/hurd/#:~:text=What%20is%20the%20GNU%20Hurd,kernels%20(such%20as%20Linux)

[4]

Linux New Media USA, LLC, 2023. Exploring GNU/Hurd – The Lost Operating System. (2023). https://www.linux-magazine.com/index.php/layout/set/print/Issues/2013/154/Exploring-the-Hurd/(tagID)/39#article_i10

[5] GNU Hurd, 2018. GNU Hurd/Open Issues/Anatomy of a Hurd System. (November, 2018) https://darnassus.sceen.net/~hurd-web/open_issues/anatomy_of_a_hurd_system/

[6] D. W. Enstrom (2018). Guideline: Technical Perspective. Retrieved from https://www.unified-am.com/UAM/UAM/guidances/guidelines/uam_technical_pers_C469D0B5.html#:~:text=The%20Technical%20Perspective%20defines%20the,and%20use%20(technical%20entities)

[7] A. Ahmed, 2023. The Importance of Developing a Community Perspective. https://blogs.illinois.edu/view/8605/748590137

[8] The Historical Thinking Project, 2023. Historical Perspectives. https://historicalthinking.ca/historical-perspectives#:~:text=Taking%20historical%20perspective%20means%20understanding,and%20actions%20in%20the%20past.

[9] U. Teichler, 2022. Globalization and the shifting geopolitics of education. *International Encyclopedia of Education* 1, (2023), 239–249. https://doi.org/10.1016/B978-0-12-818630-5.02001-7

[10] M. Manukyan and G. Gevorgyan, 2016. Canonical data model for data warehouse? *East European Conference on Advances in Databases and Information Systems*, (August, 2016), 72–79. https://doi.org/10.1007/978-3-319-44066-8_8

[11] Debian, 2021. Debian_GNUHurd. (September, 2021). https://wiki.debian.org/Debian_GNU/Hurd

[12] GNU Hurd, 2010. GNU Hurd/Extensibility. (November, 2010). https://www.gnu.org/software/hurd/extensibility.html

[13] GNU Hurd, 2009. GNU Hurd/sfi. (May, 2009). https://www.gnu.org/software/hurd/sfi.html

[14] GNU Hurd, 2016. GNU Hurd/Microkernel/Mach/Gnumach. (December, 2016). https://www.gnu.org/software/hurd/microkernel/mach/gnumach.html

[15] GNU Hurd, 2013. GNU Hurd/Microkernel. (September, 2013). https://www.gnu.org/software/hurd/microkernel.html

[16] GNU Hurd, 2015. GNU Hurd/Advantages. (March, 2015). https://www.gnu.org/software/hurd/advantages.html

[17] GNU Hurd, 2011. GNU Hurd/Hurd/Running/Debian/Debian Packages That Need Porting. (July, 2011). https://www.gnu.org/software/hurd/hurd/running/debian/porting.html

[18] GNU Hurd, 2015. GNU Hurd/Hurd/Status. (May, 2015). https://www.gnu.org/software/hurd/hurd/status.html

[19] W. J. Dally and J. T. Kajiya, 1985. An object oriented architecture. *SIGARCH Comput. Archit* 13, 3 (June, 1985), 154–161. https://doi.org/10.1145/327070.327151

[20] S. Rajagopal and J. R. Cavallaro, 2005. Communication Processors. (July, 2005). https://scholarship.rice.edu/bitstream/handle/1911/20241/Raj2005Jul10Communica.PDF?sequence=1

[21] J. Rattner and G. Cox, 1980. Object-based Computer Architecture. *Computer Architecture News* 8, 6 (1980). https://dl.acm.org/doi/pdf/10.1145/641914.641915

[22] Oxford University Press, 2019. Object-oriented Architecture. (2019). https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/object-oriented-architecture#:~:text=Examples%20of%20object%2Doriented%20architecture,and%20the%20Intel%20iAPX%20432.

[23] IBM, 2021. Multiple Server Architecture. (March, 2021). https://www.ibm.com/docs/en/tpmfod/7.1.1.3?topic=guide-multiple-server-architecture

[24] IBM, 2021. Single Server Configuration. (March, 2021). https://www.ibm.com/docs/en/contentclassificatio/8.8?topic=architecture-single-server-configuration

[25] B. Ng, A. Si, R. W. Lau, and F. W. Li, 2002. A Multi-server Architecture for Distributed Virtual Walkthrough. *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology,* (November, 2002), 163 - 170. https://doi.org/10.1145/585740.585768

[26] T. Zhang, J. Shen, C.-F. Lai, S. Ji, and Y. Ren, 2023. Multi-server assisted data sharing supporting secure deduplication for metaverse healthcare systems. *Future Generation Computer Systems* 140, (March, 2023), 299–310. https://doi.org/10.1016/j.future.2022 .10.031

[27] R. Kazman, S. Echeverria, and J. Ivers, 2022. Extensibility. Carnegie Mellon University. (April, 2022). http://doi.org/10.1184/R1/18863639

[28] R. E. Ladner, 2015. Design for user empowerment. *Interactions* 22, 2 (March, 2015), 24–29. https://doi.org/10.1145/2723869

[29] Mignot, G. L. (2005). The GNU Hurd. *Libre Software Meeting.*

[30] Hammar, F. (2010). Generalizing Mobility for the Hurd. (2010). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4cce9abb177cc58c199e13b82d498f37010c2bfc

[31] GNU, 2023. Home. Retrieved from https://www.gnu.org/home.en.html

[32] C.-C. Tsai, K. S. Arora, N. Bandi, B. Jain, W. Jannen, J. John, . . . D. E. Porter, 2014. Cooperation and security isolation of library OSes for multi-process applications. EuroSys '14: Proceedings of the Ninth European Conference on Computer Systems, (April, 2014), 1–14. https://doi.org/10.1145/2592798.2592812

[33] R. Rotta, J. Nolte, V. Nikolov, L. Schubert, S. Bonfert, and S. Wesner, 2016. MyThOS — Scalable os design for extremely parallel applications. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing*. Toulouse, France, 1165-1172. https://doi.org/ 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0179

[34] J. King and L. Wilson, 2018. Decoupling model checking from raid in 64 bit architectures. *Journal of Computer Science and Software Engineering* 10, 1 (2018). 1-6.