JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

A COMPARATIVE ANALYSIS OF FUNDAMENTAL CONCEPTS OF OPERATING SYSTEM

Asna Riaz
Department of Computer Science, University of Southern Punjab Multan
asnariaz6@gmail.com
Zahida Manzoor
Department of Computer Science, University of Southern Punjab Multan
Zahidamanzoor67@gmail.com
Kanwal Saleem
Department of Computer Science, University of Southern Punjab Multan
kanwalsaleem55@gmail.com
Muhammad Azam
Department of Computer Science, University of Southern Punjab Multan
Crossponding Author*muhammadazam.lashari@gmail.com*
Ammad Hussain
Department of Computer Science, University of Southern Punjab Multan
ammadhussain709@gmail.com

https://doi.org/10.5281/zenodo.16312122

RECEIVED	*ACCEPTED*	*PUBLISHED*
08 June 2025	15 June 2025	22 July 2025

ABSTRACT

This paper provides a comparative review of various fundamental cores of operating system, including Time Sharing, Multitasking, Kernel mode, User mode, Threads, File systems, Virtual memory, Paging, Paging and Swapping, Page hit, Page miss. The review synthesizes findings from 40 research and review papers, examining these fundamental core's performance across different applications, methodologies, and optimization tasks. The discussion highlights key findings, identifies research gaps, and suggests future research.

Keywords *– Time Sharing, Multitasking, Kernel mode, User mode, Threads, File systems, Virtual memory, Paging, Paging and Swapping, Page hit, Page miss***.**

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Introduction

Operating systems (OSs) form the foundation for computing infrastructure, governing hardware interaction, resource allocation, and execution environments. In recent decades, OS research has evolved substantially, adapting to changes in computing paradigms—from monolithic kernels to microservices, and from single-core desktops to multi-core cloud and edge environments [1][2][27]. As systems demand greater speed, energy efficiency, and scalability, OSs are expected to perform with minimal latency, maximum throughput, and robust fault tolerance [3][6][14]. This transformation necessitates novel OS designs and methodologies to support increasingly heterogeneous workloads, such as those in IoT, high-performance computing (HPC), and real-time systems [4][15][20].

Many studies in the compiled collection focus on architectural innovations, memory management strategies, and scheduling techniques. Designs like exokernels and object-oriented OSs (e.g., Choices) offer modular, flexible systems [5][27][30], while approaches like MVAS in NUMA systems enhance memory locality and reduce latency [3]. Research also explores specialized OSs like OSv for cloud VMs [1] and SYSFLOW for IoT [14], tailored for single-app workloads or constrained environments. These systems trade traditional abstractions for performance gains and highlight how customized OS designs outperform general-purpose kernels under specific conditions.

Scheduling and concurrency management remain central concerns, particularly in real-time and multi-threaded environments [15][16][33][34]. Several papers examine traditional and hybrid scheduling algorithms—like Round Robin, Priority-Based Scheduling, and Earliest Deadline First (EDF)—in both educational OSs (e.g., xv6) and production environments [18]. These studies often demonstrate that while simpler algorithms offer predictability, they lack responsiveness under heavy or mixed loads. Hybrid and adaptive models are shown to be more efficient in real-time and multicore contexts [33][35].

Memory management is another recurring theme, especially in virtualized or containerized systems [6][25][26]. Techniques like ballooning, deduplication, and hypervisor-level swapping (e.g., in

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

VMware ESX Server) are shown to significantly reduce memory overhead without harming performance [6]. Some papers highlight how limited memory insights in guest OSs

restrict dynamic adaptation, suggesting that future OSs should integrate cooperative memory policies and use machine learning for predictive resource allocation [4][17]. Security and reliability also feature prominently. Behavior-based malware detection systems, such as those using ensemble classifiers on system call sequences, achieve high accuracy and low false-positive rates [13][19], suggesting a viable path for real-time threat detection. Likewise, OS-level enhancements for fault tolerance—like redundant core architectures or replicated thread execution on FPGAs—demonstrate the feasibility of deterministic systems for aerospace and automotive use, albeit with high resource overhead [21].

Despite progress, the reviewed studies collectively indicate that many OS designs and mechanisms remain untested in real-world or heterogeneous environments [2][8][10]. Simulated evaluations and concept-only models dominate the literature, with empirical validations often missing or limited to narrow scenarios. This gap presents rich opportunities for interdisciplinary exploration—especially where hardware-software co-design, AI-based scheduling, and scalable system modeling intersect [4][17][33].

LITERATURE REVIEW

Forty research papers have been reviewed to evaluate the performance of fundamental cores of operating system. This paper offers a comparative overview of modern operating systems, namely Windows, Linux, and macOS, based on these core principles, aiming to highlight differences in implementation, performance, and design philosophy.

Key Concepts

Time Sharing

Time sharing is a method that allows multiple users or processes to use a computer system at the same time by quickly switching between them. The system gives each process a small slice of processor time, creating the illusion that they are running simultaneously.[36][37]

Multitasking

Multitasking refers to the ability of an operating system to handle multiple tasks or applications at once. It ensures that different programs can run in parallel by rapidly switching the CPU among them, so the system remains responsive and efficient. [7][10][36]

Kernel Mode

Kernel mode is a privileged operating mode where the operating system has complete access to all system resources and hardware. In this mode, critical tasks like managing memory, handling devices, and controlling processes are performed. [5][17][28]

User Mode

User mode is the restricted mode where normal applications run. Programs in user mode can't directly interact with hardware or critical system components; instead, they must go through the operating system for those operations, which helps protect the system from crashes and security issues. [1][5][22]

Threads

A thread is the smallest unit of execution within a process. Multiple threads can run inside a single program, sharing memory but executing independently, which improves efficiency and allows tasks to be performed simultaneously.[22][31][32]

File Systems

A file system organizes and manages how data is stored and retrieved on storage devices. It defines the structure for files, folders, permissions, and metadata so that the operating system and users can locate and access data efficiently.[7][24][40]

Virtual Memory

Virtual memory is a technique that allows a computer to compensate for limited physical RAM by using disk space as additional memory. It gives the illusion of having more memory available, enabling large applications to run smoothly.[3][6][25]

Paging and Swapping

Paging and swapping are strategies for managing memory when space runs low. While paging moves parts of programs (pages) in and out of RAM, swapping moves entire processes to and from disk storage to free up space for other active tasks.[6][26][37]

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

TABLE I

LITERATURE REVIEW

Sr.	Authors	Methods	Dataset	Results	Limitations	Research Gap	Performance Metrics
1	Avi Kivity, et al.	OS architecture design, benchmarking	SPECjvm2008, Netperf, Memcached	25% throughput gain, 47% latency drop	Not for multi-app VMs, limited POSIX support	Extend POSIX coverage, support multi-apps	Throughput, latency, boot time
2	Mohammad Marufuzzaman, et al.	Comparative literature review	N/A	Explains OS design trade-offs	No benchmarks, conceptual only	Real-time metrics, IoT-specific OS models	Qualitative only
3	Di Gennaro, Pellegrini, Quaglia	Linux kernel module, memory access tracking	HPC simulation workloads	Improved memory locality, reduced latency	Kernel changes required, workload sensitivity	Portability to other OSes, user-space libraries	Latency, page locality, fault tracking accuracy
4	A.S. Thyagaturu, et al.	Literature review, taxonomy	N/A	Identifies tech trade-offs and bottlenecks	No unified evaluation framework	Hybrid I/O, NUMA-aware memory, container security	Boot time, I/O latency (qualitative)

5	Russo, Johnston, Campbell	Object-oriented OS design	N/A	Demonstrated modular, reusable components	No benchmarks provided	Extend model to I/O, networking, and filesystems	Qualitative (modularity, reuse)
6	Carl A. Waldspurger	System implementation, benchmark testing	dbench (Linux I/O)	<5% overhead with overcommit, efficient sharing	Requires balloon driver, OS cooperation	Dynamic adaptation, hardware integration	MB/s throughput, memory footprint
7	Shashank Prabhakar	VMCF-based file system implementation	N/A	Enabled multitasking, concurrent access	VM/CMS-specific, no network support	Extend to distributed FS, replication	Qualitative concurrency handling
8	Christian Bendele	Conceptual architecture, QEMU simulation	Simulated workloads via QEMU	Validates decentralized OS design	Prototype only, no real hardware	Real-world deployment, memory subsystem integration	Lock latency, scheduling (simulated)
9	Shashank Prabhakar	VMCF, command parsing, concurrency primitives	N/A	Same as Paper 7	Same as Paper 7	Add modern networking and file versioning	Responsiveness (qualitative)

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

10	Christian Bendele	OS modeling, prototype extension	QEMU simulation	Validates scalability concepts	No memory/I/O integration	Full stack development, legacy app support	Message latency, core scheduler
11	Shashank Prabhakar	Same as Papers 7 & 9	N/A	Same as Papers 7 & 9	Same as Papers 7 & 9	Fault tolerance, distributed replication	Qualitative concurrency models
12	Christian Bendele	Same as Papers 8 & 10	Simulation-based metrics	Demonstrated messaging and modular scaling	Prototype only, lacks memory modeling	Integration of memory/I/O, app compatibility	Locking, messaging latency (simulated)

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

13	Bingyan Xu, et al.	Generative Adversarial Networks (GANs) to analyze I/O Request Packet (IRP) operations	system logs capturing normal and malicious I/O operations,	significant improvements over traditional ransomware detection methods.	the model's performance in detecting various ransomware variants may need further validation	Future work could focus on optimizing the model for real-time ransomware detection, improving its scalability	The model shows high accuracy in distinguishing between benign and malicious behaviors.
14	Jun Lu, Zhenya Ma, Yinggang Gao, Ju Ren, Yaoxue Zhang	Optimized network IO path, action-based prefetching, kernel-level disk IO interception, multithreading	(JVM, Python, GCC, OpenSSL) on Raspberry Pi 4B and Dell R730.	Latency reduced by 45.1%-75.8% vs Linux; up to 67.7% vs NFS; power use up to 6.7% higher but more energy-efficient overall.	Relies on historical access patterns; limited cache capacity on server; does not address large-scale concurrency.	Integration with AI models, adaptation to distributed systems, scalable predictive caching mechanisms.	Performance evaluated using latency, page fetch times, power usage; no direct "accuracy" metric.

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

15	Gulistan Ahmead Ismael1, et al.	non-preemptive, preemptive, round-robin	RTOS systems and scheduling approaches	Real-Time Operating Systems are vital for critical applications. Hard RTOS dominates the reviewed literature.	Lack of comparative benchmarks for the scheduling algorithms across different systems.	Future work should include experimental validation of reviewed algorithms, real- time simulation environments,	Discussed conceptually (e.g., meeting deadlines, task prioritization).

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

16	M. Shakor, et al.	Comparative analysis of CPU scheduling algorithms in RTOS	Theoretical models and performance metrics	Identified EDF and RMS as optimal for hard RTOS	Complex algorithms may introduce overhead	Exploration of hybrid scheduling approaches	CPU utilization, throughput, turnaround time, waiting time, response time, deadline miss ratio, jitter
17	Hayfaa Subhi Malallah1 *, et al.	TF-IDF, virtualization, ML, kernel patching	74 cited papers covering kernel-level experiments, real-time performance evaluations, intrusion detection mechanisms, and system call behaviors under stress	Linux and Android are the most studied; kernel-level tasks like scheduling, memory, and security are most impacted; ML and virtualization improve robustness	Lack of unified evaluation metrics, inconsistent tools across platforms, limited scalability of existing models	Need for lightweight, adaptive, and secure kernel solutions for modern environments like IoT, cloud, and edge computing	Accuracy and efficiency in kernel tasks like memory management, scheduling, and intrusion detection

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

18	Madan H. T., et al.	Kernel-level modifications and implementation of scheduling algorithms (RR, FCFS, PBS) in xv6 OS; development of strace, procdump, and a custom RISC-V bootloader	10 processes (5 I/O-bound, 5 CPU-bound) executed in a QEMU-based single-core RISC-V environment with 256MB RAM	FCFS achieved the lowest average wait time (37 ticks); RR ensured fair CPU access but led to longer wait times; PBS prioritized efficiently with moderate delays	Evaluation in QEMU emulation, which lacks accurate modeling of hardware-level features like interrupt timing and context-switching	Extension to real-time scheduling, optimization of dynamic priorities, integration of power-aware techniques, and validation on real RISC-V hardware	CPU ticks (average wait time, average runtime); qualitative metrics like fairness and responsiveness used instead of traditional accuracy values

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

19	Badis HAMMI* , Joel HACHE, et al.	Dynamic behavior-based analysis using machine learning (ensemble voting classifier: hard & soft voting with DT, RF, NB, KNN)	. dataset with 42,797 malware API call sequences and 1,079 benign sequences, extracted from Cuckoo Sandbox	Accuracy up to 99%, MCC up to 0.647, especially for Random Forest and Voting Classifiers; low false positives	Computational cost of ensemble methods; dependency on the quality of dynamic traces; limited interpretability of some ensemble outputs	Integration into real-time systems; cross-platform generalization; performance	Accuracy, Matthews Correlation Coefficient (MCC) used for balanced evaluation; Soft/Hard Voting improves generalization across models
20	Md. Ratan Rana, Saikat Baul	Lazy scheduling, recursive mapping, IPC-based sync	Literature review	Fast synchronous IPC, performance-optimized microkernel	Lack of preemption	Modular expansion, compatibility	IPC latency (qualitative)

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

21	Ernest Antolak, Andrzej Pułka	Redundant replicated-core architecture; register-level voting/correction; thread interleaving; FPGA-based simulation and fault injection.	No external dataset. Fault injection is internally generated during simulation using custom-built fault generator in C#.	Achieves fault correction with minimal performance loss; 78%+ injected faults had no effect; remaining faults were correctly handled.	High hardware cost; no parallelism across distinct tasks; minimum task count requirement; limited fault correction under worst-case overlaps.	Enhancing PC/FR protection, improving memory verification, dynamic redundancy strategies, instruction invalidation under real-time constraints.	Not traditional (e.g., accuracy). Metrics include: % of corrected injected errors, timing impact, task capacity vs. hardware utilization.

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

22	Sadia Sahar, Hamayun Khan, et al.	Systematic Literature Review (SLR), qualitative analysis	Research articles, conference papers, technical and industry reports from academic databases (e.g., IEEE, ACM, arXiv)	Identified benefits and limitations of user-level threads; common applications and language support; analysis of trends and future directions	No empirical benchmarking or implementation-based evaluation; findings based on secondary data only	Need for better OS integration, debugging tools, hybrid thread models, support for blocking I/O, and scalable scheduling mechanisms	performance insights are based on reported results in the reviewed literature
23	Xia Zhao, Huiquan Wang	Online profiling, cluster-aware scheduling, SM partitioning, injection control	GPU simulation with multitasking workloads	12.9% average system throughput improvement, up to 76.5%	Topology dependence, limited workload diversity in evaluation	Needs support for heterogeneous workloads and new GPU designs	System throughput gain (% improvement

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718

VOLUME . 4 ISSUE . 2 (2025)

24	Liu, et al.	Specification-based fuzzing, systematic on-disk snapshotting, state comparison methods	BTRFS, F2FS, ext4, UFS, XFS, BCacheFS, OpenZFS (Linux file systems)	Found 22 new bugs, 44% better coverage than Hydra	Higher runtime overhead; some manual effort in setup	Add image mutation, automate spec writing, optimize snapshot handling	Code coverage, bug count
25	Ernest Antolak, et al.(2024)	Polynomial approximatio empirical measurement FPGA testben	60 custom tasks + 300 system load combinations on KCU105 FPGA board	High-accuracy power estimation (< 4% error), polynomial TF-power model	Limited to FPGA; susceptible to temperature variation	No ASIC validation yet; temperature influence not fully modeled	Estimation accuracy (error ≤ 4%); comparison with VIVADO estimations

26	Shun Kida, et al.	NUMA-based CXL emulation, benchmark-based comparison, synthetic modeling	Graph500, NAS Parallel Benchmarks (NPB), scikit-learn (K-means), YCSB + Memcached	SSD swapping offers comparable or better throughput in 9/14 benchmarks	No real CXL hardware, limited to a specific system and OS version	Lack of adaptive systems choosing between CXL and SSD; evaluation of tiering	Throughput (varies by benchmark: TEPS, ops/sec, inverse execution time)
27	Dawson R. Engler, et al.	Exokernel design, code inspection, inlined calls, type-safe language	None	Enhanced performance and flexibility via safe kernel customization	Security risks, development complexity, lack of benchmark evaluation	Needs real-world validation, better developer tools, and safety metrics	No quantitative accuracy, but performance and security are key concerns

28	Hayfaa Subhi Malallah, et al.	Compared different types of operating system kernels (like microkernel, monolithic, hybrid) and reviewed various operating systems (Windows, Linux, iOS, Android, Mac) based on existing studies	Not experimental; synthesized from literature and technical comparisons of existing OSs (Linux, Windows, Android, iOS, Mac); no empirical dataset used	Found that Linux and Android are the most researched and used. Discussed common kernel issues such as security, compatibility, and performance. Explained the types of kernels and features of each OS	The study is descriptive and lacks real tests or measured results	Need for empirical performance evaluation of kernel implementations across OSs; more experimental studies required to test kernel improvements under real-world scenarios	None

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

29	Roberto Rodriguez - Zurrunero, et al.	Empirical testing using real-world IoT devices; task priority variation; timing analysis	Data collected from tests conducted in two real IoT scenarios	Processing tasks significantly affect communication performance; mutual timing dependency matters	Study limited to specific OS and protocols; findings may not generalize	Lack of research on task interaction in modern IoT OSs; need broader system testing	Not focused on accuracy; evaluates timing efficiency, communication reliability
30	Vincent Russo	Object-oriented system design using C++ class hierarchies	No external dataset; based on implementation and system behavior of Choices	Successful OS implementation using class-based design; modular, efficient, extendable	Limited to one architecture (Encore Multimax); lacks performance benchmarks	Need for broader validation on various hardware platforms; limited prior OOP OS examples	Not focused on accuracy; emphasizes system flexibility, modularity, efficiency
31	Xerox PARC, et al.	Static code analysis, microsecond-level dynamic tracing, thread statistics	Source code from Cedar and GVX systems (~2.5 million lines)	Identified 10 thread usage paradigms; found that thread priorities can be	Potential bias from specific programmer communities; no direct system comparison	Need for tools and abstractions to better manage thread priorities and complex	Focused on responsiveness and performance patterns; not accuracy-based

				problematic		interactions	
32	Okonta O.E, et al.	PMU-based performance analysis, comparison between single-threading and HT modes	PMU data + 4 full-scale CFD scientific applications	HTT improves processor efficiency but not always application performance	Performance gains vary by workload; requires OS support for best results	Better scheduling and OS-level support needed for HT optimization	PMU metrics (e.g., unhalted core cycles, cache hit/miss rates), not accuracy
33	Gulistan Ahmead Ismael, et al.	Review and comparison of 20 scheduling methods used in real-time systems from 2018 to 2020	Information taken from research papers, tests on real systems like IoT devices, processors, and cloud setups	Better use of CPU, less energy used, faster task handling, fewer delays	Some methods are complex, slow to run, and need lots of data or training to work well	More work is needed on saving energy, handling tasks on GPUs, fault-proof systems, and using AI for better scheduling	Includes CPU usage, success in task handling, energy saving (up to 46%), and meeting time deadlines

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

34	Anoop Gupta, et al.	Simulation-based study of scheduling strategies (priority, gang, process control, affinity, handoff) and synchronization methods (blocking vs. busy-waiting)	Four real parallel applications: MP3D, LU, Maxflow, and Matmul, simulated on 12-processor systems	Process control had the best performance (72% CPU utilization); blocking locks improved from 28% to 65%; gang scheduling up to 71%	High overhead from context switches, cache miss issues, limited benefit from affinity and handoff scheduling	Need to test methods on real hardware, explore scalability for large distributed systems, and integrate I/O and mixed workloads	Processor utilization (%); highest observed was 74% (batch), 72% (process control), 71% (gang scheduling)
35	Mohammed Y. Shakor	Literature survey of existing scheduling and synchronization algorithms in OS	No dataset used; conceptual and theoretical analysis	Identifies advantages and limitations of major scheduling/synchronization methods; useful insights into real-	No experimental data or benchmarking; only theoretical discussion	Need for real-world implementation, dynamic scheduling in complex environments, better real-time and multicore solutions	Discussed theoretically: CPU utilization, waiting time, turnaround time, throughput, and response time

				time and multicore OS needs			
36	Nicholas Turner	Custom operating system kernel design using the 68000 microprocessor; emphasis on minimal context switching, no memory management unit used	No formal dataset; performance tested through design goals and system behavior under I/O load	Achieved high speed, compactness (kernel under 20K), smooth multitasking for 20 users over serial lines	No memory protection, system depends on cooperative "polite" applications, limited modularity	No support for memory management or broader dynamic usage; future work could include scalability, protection, or general-purpose application support	Qualitative metrics: fast context switching, minimal delay under load, compact code size (<20K), low overhead

37	Peter J. Denning	Static vs. Dynamic Storage Allocation	Not applicable (conceptual/theoretical paper)	Dynamic (especially automatic) allocation is more effective for modern, modular, and shared systems	Static methods are inflexible and don't handle unpredictable program behavior or memory availability	Need for better automatic memory management in dynamic and multiprogramming environments	Not applicable (no numerical accuracy provided)
38	Donald E. Porter	Added system transactions to Linux (TxOS)	Examples like OpenSSH install, Linux compile, OpenLDAP	Made system actions safer and more reliable with only small slowdowns	Needs changes to the OS; may not work with all old software or hardware	More work needed to support all system parts and full compatibility	10% slower for installs; 2–4× faster for some tasks like writing in OpenLDAP
39	Rodney Van Meter, et al.	Review and analysis of existing NAP projects and OS support	Various NAP projects and systems	Identified key features, benefits, and challenges of NAPs	Rapidly evolving technology; limited coverage of related areas	Need for better OS support and more research on NAP design and usage	

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

40	JAMES J. KISTLER, et al.	Disconnected Operation using Client-side Caching	Trace-driven simulations; Coda File System usage data	Enabled continued work during network failure; 1–2 days offline with quick sync	Limited to small disk space (100MB); may not suit large- scale data	Exploring longer disconnection periods; large-scale deployment	Reconnect & sync time ~1 minute; disk use /50–100MB

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Summaries

This paper introduces OSv, a new operating system designed specifically for cloud-based virtual machines running single applications. Unlike general-purpose operating systems, OSv eliminates unnecessary abstractions such as user/kernel mode separation, multiple address spaces, and process management—optimizing instead for performance, simplicity, and fast startup. The OS is implemented using a single address space and restructured kernel subsystems like the scheduler, network stack, and file system. It supports running Java applications and other Linux binaries via POSIX compatibility layers. Performance benchmarks using workloads such as Memcached and SPECjvm2008 demonstrate significant improvements: up to 25% higher throughput and up to 47% reduction in latency compared to Linux on KVM. The system offers fast boot times (<1s) and compact image size. However, it is limited to single-application deployments and lacks full POSIX compliance.

This survey explores various OS architectures, focusing on their treatment of scalability, reliability, memory management, and security. It examines several operating systems such as MACH, Hive, TORNADO, and K42, analyzing their suitability for multicore and IoT environments. Each OS showcases different strategies—Hive with its fault containment through cells, TORNADO optimizing for NUMA with object-oriented design, MACH introducing microkernel ideas, and K42 supporting modularity and multiprocessing. The paper outlines how these systems attempt to balance performance and isolation in multicore systems. However, it lacks empirical benchmarks and focuses primarily on theoretical capabilities. Future research is suggested in the direction of OS scalability, especially in heterogeneous multicore and resource- constrained IoT systems.

This paper tackles the inefficiencies in NUMA systems for multi-threaded applications with shared memory. Traditional NUMA optimizations rely on system-wide memory access patterns, which often fail for true multi-threaded workloads. The authors propose Multi-View Address Space (MVAS), a Linux kernel extension that allows each thread to maintain its own memory view using sibling page tables. This

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

enables thread-specific tracking of memory access and supports intelligent memory migration, thereby improving locality. The solution is implemented as a Linux kernel module and evaluated on a 32-core 8-node NUMA machine using HPC simulation workloads. Results show improved memory locality and reduced latency, though the method requires kernel-level changes and performs variably depending on application behavior.

This survey analyzes enabling technologies (OS and hypervisor layers) used for executing network functions (NFs) on general-purpose computing (GPC) platforms. The paper categorizes solutions based on abstraction mechanisms (containers, para-virtualization), memory strategies (in-memory computing, shared memory pools), and I/O optimizations (SR-IOV, DPDK, polling). It assesses performance trade-offs between isolation and throughput, particularly under latency- sensitive NF workloads. Technologies like Kata containers, XDP, and lightweight hypervisors are reviewed. Despite their promise, existing platforms lack unified evaluation frameworks and often struggle to balance flexibility and performance. The authors recommend exploring hybrid I/O techniques, NUMA-aware memory allocation, and container security models in future work.

This foundational paper presents the Choices operating system, an experimental OS built entirely using object-oriented programming (OOP) in C++. It explores modular OS construction for multiprocessor systems by organizing components (like process management and exception handling) into a class hierarchy. This separation of concerns allows for cleaner design, easier maintenance, and hardware portability. Though the system focuses more on design than performance, it demonstrates that OOP can be used effectively for real-time and multiprocessor OS kernels. The paper highlights the need for new software engineering paradigms in OS development but lacks quantitative evaluations or large-scale empirical results.

This paper introduces innovative memory management techniques used in VMware ESX Server, such as ballooning, idle memory tax, content-based page sharing, and hot I/O page remapping. These mechanisms allow the hypervisor to overcommit physical memory across VMs while maintaining performance isolation and minimizing redundancy. The ballooning driver communicates with the guest OS to reclaim underused memory without affecting VM stability. Evaluation with the dbench benchmark shows that memory can be reclaimed with only 1.4%– 4.4% performance degradation. The paper is influential in how modern hypervisors manage memory pressure without modifying guest OS kernels.

This master's thesis from 1986 presents the design and implementation of a multi-user, multitasking hierarchical file system for IBM's VM/CMS platform. The system emulates UNIX-like features—hierarchical directories, access protection, concurrent access—using the Virtual Machine Communication Facility (VMCF) to enable communication between a central file server virtual machine and multiple user VMs. The file server handles all file operations and supports two modes: quiesce mode (users are notified when the server is busy) and queueing mode (commands are queued and processed concurrently). The system incorporates classical concurrent programming concepts like semaphores and monitors for task synchronization. Although the implementation was successful in enabling true multitasking and concurrent file operations in VM/CMS, it is inherently limited by its dependence on a dated environment and does not integrate with modern networking or distributed file systems.

This bachelor's thesis proposes a decentralized OS design intended for future systems with dozens to hundreds of cores, highlighting the limitations of traditional SMP-based OSes like Linux and Windows NT. The author suggests that rather than retrofitting existing monolithic kernels, new OS architectures should treat the many-core system more like a distributed environment. The proposed design splits core responsibilities, using independent binary modules and per-core OS services with communication via message-passing (inspired by Barrelfish and fos). The OS avoids shared-kernel bottlenecks and uses a QEMU-emulated environment for prototyping. Features like decentralized scheduling, dynamic core allocation, and communication channels between services were partially implemented. Although the thesis does not

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

present real hardware evaluation, it sets the groundwork for scalable OS design in the many-core era. Future work includes real hardware deployment, full-feature integration (especially memory and I/O), and support for legacy applications.

This thesis presents the design and implementation of a multiuser, multitasking file system for VM/CMS, modeled after UNIX-style hierarchical file systems. The system is built using the Virtual Machine Communication Facility (VMCF), allowing communication between user VMs and a central file server VM. It supports features like exclusive file updates, queuing, and concurrency via multi-threaded command processing. The architecture ensures concurrent file access and efficient disk space utilization in legacy environments. Limitations include dependence on outdated VM/CMS and lack of integration with modern networking protocols.

This bachelor's thesis proposes a decentralized OS architecture for managing hundreds of cores in many-core systems. It critiques legacy OS scaling limits and advocates a distributed design with minimal shared memory and inter-core messaging. Inspired by systems like Barrelfish and fos, it proposes per-core OS services and scalable core scheduling policies. A partial prototype using QEMU emulation demonstrates initial feasibility. The work identifies several future directions, including refined memory models, realistic workload testing, and integration of legacy support layers.

The thesis builds a UNIX-like file system on top of the VM/CMS environment to enable multi-user, hierarchical directory support, replacing CMS's flat, single-user model. Using message-passing over VMCF, the system supports queuing, exclusive access, and real-time multi- user command execution. It introduces quiesce and queuing modes for concurrent access. The implementation covers command parsing, queuing protocols, concurrency primitives, and a custom file server. Though successful in its context, it is constrained by hardware and outdated OS support.

This paper outlines a novel operating system architecture designed for systems expected to have dozens to hundreds of cores. It critiques traditional locking and kernel designs, proposing an architecture that avoids shared memory bottlenecks by using decentralized services and message-passing. The proposed system adopts lessons from previous distributed OSes and is
aimed at general-purpose computing. Early implementation shows feasibility, though real-world applicability and performance validation are pending. Future work includes full system build-out and legacy compatibility support.

This research paper presents a novel approach to ransomware detection on Windows operating systems by leveraging Generative Adversarial Networks (GANs) to analyze file system I/O Request Packet (IRP) operations. The suggested method significantly increase ransomware detection by dynamically monitoring IRP operations and differentiating between benign and malicious behaviors with high accuracy. Unlike traditional detection systems that rely on predetermined threat signatures, the use of GANs allows for adaptability to evolving ransomware tactics, enabling the system to detect unknown threats. Through rigorous testing, the model outperforms conventional detection methods, showcasing a potential shift as regard machine learning-based solutions for cybersecurity, mostly in combating the increasing sophistication of cyber threats. While the approach shows promise, future work may focus on optimizing the model for real-time detection and testing it against various ransomware variants.

SYSFLOW is a smart executable delivery system designed for resource-constrained IoT devices, contributing an alternative to traditional methods like RPM and NFS, which areineffective due to high storage and I/O demands. It streams executables on-demand from a server by monitoring client disk I/O, predicting future block access using historical patterns, and asynchronously delivering required data. Using multithreading, asynchronous communication, and dynamic caching, SYSFLOW significantly reduces latency—up to 75.8% over Linux and 67.7% over NFS—while slightly increasing power use but lowering total energy through faster execution. even with its reliance on historical patterns and limited scalability, SYSFLOW shows strong potential, with future work aimed at AI-based prediction and distributed system support.

This review paper investigates the architecture, scheduling mechanisms, and classification of Real-Time Operating Systems (RTOS), which are necessary in systems requiring timely and deterministic responses. The study discusses hard, soft, and firm deadlines and emphasizes the significance of meeting timing constraints in critical applications like healthcare, aerospace, and nuclear systems. It categorizes scheduling strategies—preemptive, non-preemptive, and round- robin—and examines 20 related works from 2018 to 2020. The paper concludes that RTOS is integral for embedded systems due to its time-sensitive execution capabilities. The review identifies the need for further optimization in scheduling algorithms and real-time decision-making under complex workloads.

This study analyze CPU scheduling algorithms used in real-time operating systems (RTOS), categorizing them into preemptive and non-preemptive types. While simpler algorithms like FCFS and Round Robin are easy to implement, they often fail to meet the strict timing needs of hard real-time systems. In contrast, advanced algorithms like Earliest Deadline First (EDF) and Rate-Monotonic Scheduling (RMS) are better suited for such environments due to their ability to reliably meet deadlines. This paper identifies applications in critical systems like air traffic control and medical devices, but also notes the trade-off between scheduling complexity and resource limitations. Future research ismotivated to develop hybrid models that balance performance and efficiency. Key metrics analyzed include CPU utilization, deadline miss ratio, and jitter.

This research paper offers a detailed review of kernel architectures—monolithic, microkernel, and hybrid—across major OS platforms like Windows, Linux, Android, iOS, and macOS. Analyzing 74 research papers, it examines key kernel functions such as memory management, scheduling, and system security, particularly in contexts like IoT, cloud computing, and web servers. Linux and Android receive special focus due to their open-source nature and vulnerability to security threats. The study examines the use of techniques like machine learning, TF-IDF, and virtualization for increasing kernel robustness.

This study assess Round Robin, FCFS, and Priority-Based Scheduling in the xv6 operating system on RISC-V architecture to assess their performance in resource-constrained environments. Modifications included implementing the algorithms, system call tracing (strace), process monitoring (procdump), and a custom bootloader. Experiments on QEMU with I/O- and CPU- bound workloads revealed that FCFS minimized wait time but suffered from the convoy effect, RR ensured fairness with longer waits for CPU-heavy tasks, and PBS balanced priorities with moderate overhead. Even with QEMU's limitations, the research highlights xv6's educational value and suggests future work in hybrid scheduling, real-time support, and hardware validation.

This study suggest a behavior-based malware detection approach by examining Windows system call sequences using a voting classifier. The methodology employs ensemble learning techniques, combining Decision Tree, Random Forest, Naive Bayes, and K-Nearest Neighbors (KNN) to improve detection accuracy and decrease false positives. The dataset comprises over 43,000 API call sequences (42,797 malicious and 1,079 benign) collected via Cuckoo Sandbox reports. The aim is to detect novel and signature-evasive malware without relying on traditional static analysis. Results show high detection performance with up to 99% accuracy and a Matthews Correlation Coefficient (MCC) up to 0.647, especially with Random Forest and Voting Classifiers. Applications include Endpoint Detection and Response (EDR) systems, where the model can identify previously unseen threats.

This research paper reviews eleven microkernel-based operating systems to analyze their approaches to core OS functions like scheduling, memory management, and inter-process communication (IPC), and how these impact system security, performance, and modularity. Through comparative analysis of systems like L4, seL4, QNX, and RC 4000, it finds that most microkernels use priority-based round-robin scheduling, varied memory models, and message- passing IPC to achieve fault tolerance and real-time capabilities. While these designs benefit modularity and adaptability—especially in embedded and secure environments—they also introduce challenges such as complexity, performance overhead, and limited ecosystem support.

This study establish a fault-tolerant, time-predictable multitasking system based on the PRET architecture, designed for safety-critical real-time applications in harsh environments like aerospace or automotive systems. The system uses replicated single-core execution with majority voting among identical cores to detect and correct register-level faults, preserving deterministic timing through innovations like "cycle stealing" during idle cycles. Implemented on an FPGA and tested with fault injection, it effectively challenges include limited OS integration, debugging difficulties, and poor support for CPU-heavy workloads. Future research should focus on improving performance, debugging, and scheduling for real-time and dynamic environments.

This study enhances GPU spatial multitasking by introducing a cluster-aware scheduling policy that considers the GPU's internal network-on-chip (NoC) structure, which current methods corrects up to two simultaneous register faults and showed over 75% fault immunity. Limitations include high hardware demands, a minimum task count, and lack of support for diverse parallel tasks.

This study analysis the current research on first-class user-level threads, or green threads, which are lightweight threads managed in user space rather than by the operating system. Using a systematic literature review, the authors found that green threads offer fast context switching and are easy to use for handling many small tasks, making them popular in languages like Go, Erlang, and JavaScript. They're great for web servers, network apps, and games, but struggle with tasks that need true parallelism or involve blocking I/O. Key overlook. By optimizing how streaming multiprocessors (SMs) are grouped and how applications share network ports, the approach reduces contention and boosts performance. Using low-overhead profiling and dynamic scheduling, the method increases system throughput by an average of 12.9%, with gains up to 76.5%, without major hardware changes. It's useful for HPC, AI, and GPU cloud workloads, though its current design is limited to specific NoC topologies and may face challenges with diverse workloads.

SnapCC is a file system crash consistency testing framework designed to overcome the limitations of existing tools like Hydra and B3 by systematically exploring on-disk states. It uses specification-based fuzzing, snapshotting (via QEMU and BTRFS), and automatic valid state verification to detect consistency violations across various Linux file systems. SnapCC discovered 22 new bugs and achieved up to 44% higher coverage than prior tools, proving its effectiveness and adaptability. While it incurs slightly higher overhead and requires some manual setup, future improvements aim to introduce smarter image mutation, reduce manual intervention, and optimize snapshot handling.

The study establishes accurate and practical method for estimating power consumption in multitask, time-predictable real-time systems using a task frequency (TF)-based model. different traditional tools like VIVADO, which offer probabilistic and often inaccurate estimations, the authors use FPGA-based empirical measurements and polynomial modeling to capture the nonlinear relationship between power, system frequency, and task load. Implemented on a Xilinx Kintex UltraScale FPGA, the method achieves high accuracy (≤4% error) and supports power- aware system design in safety-critical applications.

This study compares Compute Express Link (CXL) memory and SSD-based memory swapping as strategies for expanding memory capacity in data-intensive applications. Using 14 macro-benchmarks across diverse workloads (graph processing, scientific computing, machine learning, and key-value stores) and a synthetic benchmark to control access patterns, the authors find that SSD swapping delivers comparable or better performance than emulated CXL memory in 9 out of 14 cases. The results reveal that SSD swapping performs well in workloads with skewed, hot data access, while CXL

ONLINE ISSN 3006-9726
PRINT ISSN 3006-9718
VOLUME . 4 ISSUE . 2 (2025)

excels in latency-sensitive, memory-intensive tasks.

The study introduces Exokernel, a novel operating system architecture that enhances performance and flexibility by minimizing kernel responsibilities and exposing raw hardware resources directly to applications. This approach eliminates high-level abstractions within the kernel, allowing user-level libraries and applications to implement their own, more efficient versions. To test this idea, the authors developed Aegis, a prototype exokernel, and employed three key techniques to ensure safe kernel-level customization: code inspection, inlined cross-domain calls, and the use of type-safe languages. These enable safe execution of application code in privileged mode, forming what the authors call a secure programmable machine. Rather than relying on traditional datasets, the evaluation is based on the Aegis prototype and conceptual performance analysis. Results show that exokernels improve modularity, performance, and customizability, making them ideal for use in custom operating systems, embedded systems, and high-performance computing. However, limitations include security risks from running user code in kernel mode, increased development complexity, and the lack of extensive real-world performance data. Future research should focus on broader comparisons with traditional OS designs, better developer tools for safe customization, and strategies for balancing security and performance in diverse environments.

This comprehensive review investigates kernel issues, structures, and functions across various operating systems—specifically Windows, Linux, Android, iOS, and Mac OS. The studyemphasizes the importance of the kernel as the central component of an OS, focusing on its structure (monolithic, microkernel, hybrid), performance, and associated challenges like security, multitasking, and resource management. The paper aggregates findings from several research works that analyze kernel vulnerabilities, performance bottlenecks, and improvements using techniques such as virtualization, hardware support, machine learning, memory management strategies, and real-time systems evaluation. The analysis spans desktop and mobile environments, cloud infrastructure, IoT devices, and embedded systems. The review identifies that Linux and

Android kernels dominate the research landscape due to their open-source nature and wide deployment. Applications of the reviewed research include enhanced intrusion detection, improved real-time performance, optimized memory usage, and secure embedded systems. While the reviewed solutions improve OS kernel robustness and efficiency, limitations persist, such as high inter-process communication overhead in microkernels, outdated system support, and insufficient protection against evolving rootkits. Future research directions include integrating cloud-IoT infrastructure more efficiently, refining machine learning-based detection systems, and developing adaptable kernel architectures that meet modern performance and security demands.

This study explores the interaction between processing and communication tasks in IoT end-devices within the context of edge computing, highlighting the challenges that arise due to increased computational demands. The research is motivated by the shift from simple Wireless Sensor Networks (WSNs) to more complex IoT environments where end-devices are expected to generate high-level insights rather than just collect raw sensor data. The study uses an empirical approach involving multiple tests across two real-world scenarios using a specific IoT operating system and wireless communication protocols. These experiments vary processing loads, task priorities, and communication parameters such as the radio duty cycle. The results reveal a significant cross-influence: high processing loads can degrade communication performance and vice versa. Importantly, this performance impact is not only due to computational load but also how processing and communication timings are scheduled and managed. Applications of this work include optimizing OS and protocol design for IoT systems to ensure efficient and dependable communication and processing. However, limitations include a focus on specific OSs and protocols, which may limit generalizability. Future research could explore a broader range of operating systems, more diverse network scenarios, and refined scheduling strategies to mitigate these cross-effects.

This study presents the design and implementation of the Choices operating system, which applies object-oriented programming (OOP) principles—specifically class hierarchies

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

in C++—to construct a multiprocessor OS architecture. The goal is to explore whether class hierarchical design can effectively support the development of low-level OS primitives such as interrupt handling, process switching, scheduling, and synchronization. The Choices OS aims to support real-time, high-performance applications on heterogeneous multiprocessor systems by allowing modular customization and efficient system behavior without the overhead of general-purpose OS features. The system was tested on the Encore Multimax platform, and results howed that using OOP facilitated modular design, hardware specialization, and separation of mechanism and policy. The study confirms the feasibility of building full OSs using C++ and shows that such an approach improves maintainability and performance in specific-use environments. However, the paper is limited to one system architecture and does not include comparisons with traditional kernel designs. Future work could explore performance benchmarks, real-time constraints in more diverse hardware settings, and integration with modern development tools.

This study investigates how threads are used in two large systems developed at Xerox— Cedar (a research platform) and GVX (a commercial product). The purpose was to identify common thread usage paradigms, both effective and problematic, by analyzing over 2.5 million lines of code across 10,000 modules. The researchers used a three-pronged methodology: static code analysis, dynamic event timing analysis at microsecond resolution, and examination of macroscopic thread statistics. They identified ten paradigms of thread use, including well-known ones like defer work and lesser-known ones like slack processes and encapsulated fork. The slack process was found to significantly improve performance but was also hard to manage. The study revealed that thread priorities can lead to unintended issues, highlighting the complexity of designing thread-based systems. The research emphasizes thread behavior as a tool for structuring software rather than just exploiting parallelism. Though limited by programmer bias and lack of cross-comparison between systems, the study offers practical insights and identifies future research directions in thread abstractions and design tools.

This paper explores Intel's Hyper-Threading Technology (HTT),

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

focusing on how it improves processor resource utilization by allowing two threads to run on each physical core simultaneously. The study uses Performance Monitoring Unit (PMU) data, including metrics like instructions retired, unhalted core cycles, and cache performance (L2/L3), to evaluate HTT effectiveness. The goal is to assess whether HTT improves overall efficiency and performance. By analyzing real-world computational fluid dynamics (CFD) applications, the study finds that while HTT often improves processor resource utilization, it doesn't always translate to better overall application performance. The paper also emphasizes that operating system awareness of HTT is critical; otherwise, HTT can negatively affect performance. Limitations include the dependency on workload type and OS support. Future research is needed to refine HTT scheduling and resource management across varying workloads.

This study reviews various scheduling algorithms used in Real-Time Operating Systems (RTOS), which are crucial for systems that must respond quickly and correctly within a specific time, like hospital monitors or autopilot systems. The paper compares several scheduling methods, explaining their purpose, strengths, and weaknesses. It also discusses how different models and technologies like GPUs, DAG tasks, and embedded systems are used in real-time scheduling. The study uses past research (2018–2020) involving simulations and real-world tests to show results like improved CPU use, reduced delays, and better energy savings. These findings help in designing better real-time systems but also reveal limitations like high overhead, complexity, and the need for better multi-task and energy-efficient models. The paper highlights the need for future research in areas like energy harvesting, hardware acceleration, deep learning integration, and more accurate scheduling under complex conditions.

This study, conducted by researchers from Stanford University, explores how different operating system scheduling methods and synchronization strategies affect the performance of parallel applications on multiprocessor systems. Using detailed simulations, the authors tested five scheduling policies—priority-based, gang (coscheduling), process control, handoff, and affinity scheduling—on real applications like matrix multiplication, LU decomposition, and graph-based algorithms. The study also compared busy-waiting versus blocking synchronization. Results showed that busy-waiting led to poor performance due to wasted processor time, while blocking locks and gang scheduling significantly improved efficiency. Among all, process control offered the best overall performance by aligning the number of running tasks with available processors, leading to better CPU usage and cache behavior. However, some methods like handoff and affinity scheduling provided only small improvements. The paper highlights limitations such as increased context-switch overhead and cache miss issues and suggests future work should explore real-world implementation and scalability for larger multiprocessors.

This paper by Mohammed Y. Shakor provides a detailed survey of scheduling and synchronization algorithms in operating systems, highlighting their importance in managing CPU resources and ensuring smooth multitasking. It covers popular scheduling techniques like First- Come First-Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, Round Robin (RR), Multilevel Queue, and Multilevel Feedback Queue, along with synchronization problems like race conditions and critical sections. The study also examines real-time operating systems, multiprocessor and multicore scheduling, and the use of dynamic versus static scheduling strategies. Each algorithm's strengths and weaknesses are discussed, with FCFS being simple but inefficient under load, and RR being fair but leading to longer waiting times. The purpose of this study is to guide new researchers by presenting foundational knowledge and challenges in this field. The paper does not use a specific dataset but relies on theoretical and conceptual models. Its limitations include a lack of experimental validation and real-world benchmarks. Future research is suggested in optimizing scheduling for dynamic environments and enhancing synchronization in real-time and multicore systems.

This article by Nicholas Turner describes the development of a custom 32-bit multitasking kernel built on the 68000 microprocessor for Terra Nova Communications. The goal was to create a fast, compact, and low-cost operating system suitable for handling up to 20 real-time users over serial lines, with minimal delay and consistent performance. Instead of using commercial operating systems—which were too slow, bulky, and full of

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

unnecessary error-handling code—the team designed their own kernel tailored to known applications. They avoided memory management units and reduced context-switching time by minimizing what data had to be saved, resulting in a smaller, faster system. The project shows how a targeted, streamlined OS can outperform general- purpose systems when exact requirements are known. Its main limitations include lack of memory protection and reliance on well-behaved applications. Future improvements could explore more dynamic task handling and increased modularity for broader use.

Early computers used a memory hierarchy due to the high cost of fast memory. Programmers first managed memory manually, but this became hard with complex programs and high-level languages. Two methods of memory allocation emerged: static (planned ahead) and dynamic (adjusts as the program runs). Static methods became less useful due to changing program needs. Dynamic, automatic memory allocation became essential, especially for multitasking and time-sharing systems.

This paper talks about TxOS, a version of Linux that adds system transactions to help programs safely access system resources like files. These transactions make sure that groups of actions happen all at once, without errors from crashes or other programs running at the same time. This helps prevent common problems like file corruption or security bugs. TxOS adds simple commands for starting and ending transactions. It runs well on normal computers with little slowdown. For example, installing software with TxOS is safer and only 10% slower. TxOS makes programs easier to write and more reliable.

This paper explains Network-Attached Peripherals (NAPs), which connect to computers through a network instead of regular cables. NAPs are becoming more common but need special operating system support to work properly. They can be shared by many computers, work over long distances, and communicate directly with each other. However, they face challenges like slower speeds, security issues, and data problems. The paper focuses on three main uses: device communication, multimedia, and storage. It aims to help understand NAPs better and encourage more research and improved system designs.

This paper discusses the problem users face when remote failures in distributed file systems stop their work, even though their own computers are powerful enough to work independently. Distributed file systems like NFS and AFS are helpful for sharing and managing data, but they create a dependency on network access. The authors propose "disconnected operation," which lets users keep working during network outages by using cached data. They implemented this in the Coda File System at Carnegie Mellon University. Their tests showed it works well, allowing users to work offline for 1–2 days and then quickly sync changes. This approach improves both availability and user experience

DISCUSSION

Across the 40 studies, several core tensions emerge: performance vs. modularity, generality vs. specialization, and simplicity vs. scalability. While minimalist kernels like OSv and exokernels show exceptional performance in constrained applications, they lack flexibility or compatibility with legacy applications. Conversely, complex hybrid kernels support a broader range of workloads but suffer from increased overhead and security risks. Similarly, while static scheduling techniques perform well in controlled environments, dynamic, energy-aware scheduling models are needed for modern, variable workloads, especially in edge and IoT systems.

Another key insight is the increasing convergence of OS research with other disciplines—such as AI (for predictive scheduling and memory allocation), formal verification (for fault tolerance and security), and hardware design (e.g., FPGA-based implementations). This multidimensional approach is critical in handling diverse modern workloads, from cloud-scale distributed services to embedded real-time systems. However, several proposals remain conceptual or simulation- based, underscoring the need for real-world implementations and broader hardware validation. Despite limitations, these studies collectively expand the knowledge base of OS design and performance management. They point toward future systems that are not only faster and more efficient but also more secure, scalable, and intelligent.

Identified Research Gaps

Many studies lack consistent evaluation metrics or frameworks.

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Designs validated in QEMU or simulations need deployment on actual hardware.

Few kernel-level performance studies deeply integrate modern security mechanisms.

Despite efforts real-time scheduling under mixed loads needs deeper investigation.

Future Research Directions

To address identified gaps, future research should:

There's a need for standardized benchmarking and performance metrics across OS prototypes and scheduling algorithms to allow more direct comparisons.

Future work should integrate machine learning for predictive scheduling, adaptive memory management, and anomaly detection—especially for cloud and edge computing environments.

Decentralized and message-passing OS designs (e.g., Barrelfish-inspired systems) need to mature, incorporating full memory and I/O stacks and real-world workload testing.

Further research is needed into real-time threat detection, system-level transaction management (e.g., TxOS), and resilient OS architectures that minimize downtime under faults.

Hybrid scheduling models must be designed with energy awareness, particularly for RTOSs in power-constrained IoT and mobile environments.

Many papers rely on simulations. Future research should validate new OS designs and kernel modifications on actual hardware platforms (e.g., RISC-V, FPGA, or CXL-enabled systems).

CONCLUSION

In summary, the core components of an operating system—such as time-sharing, multitasking, memory management, and file systems—work together to provide a stable and efficient environment for running applications. The collected body of research reflects a vibrant and evolving landscape in operating system (OS) design and implementation, driven by diverse application domains including cloud computing, real-time systems, embedded platforms, and high-performance computing. Innovations span across kernel architectures, scheduling algorithms, memory management techniques, and OS-level security mechanisms. From exokernels and object- oriented designs to decentralized, many-core systems, researchers are continuously rethinking the traditional abstractions and boundaries of OS functionality to meet the demands of modern computing. Several contributions demonstrate significant improvements in performance metrics like latency, throughput, and energy efficiency, while others highlight conceptual advancements that pave the way for modular, secure, and scalable systems.

Despite notable progress, a significant portion of the literature remains theoretical or simulation-based, with limited deployment on real-world hardware. Many approaches show promise in controlled environments but face challenges in terms of scalability, interoperability, and generalizability. There is also a consistent call for unified benchmarking frameworks, better developer tooling, and empirical validations. Going forward, the integration of AI, support for heterogeneous hardware, and co-design of hardware-software architectures will be crucial to bridging these gaps and realizing robust, adaptive, and efficient operating systems for the next generation of applications.

REFERENCES

Xu, B., & Wang, S. (2024). Examining windows file system irp operations with machine learning for ransomware detection.

Ismael, G. A., Salih, A. A., AL-Zebari, A., Omar, N., & Merceedi, K. J. (2021). Scheduling Algorithms Implementation for Real Time Operating Systems: A Review. Asian Journal of Research in Computer Science, 11(4), 35-51.

Shakor, M. Y. (2021). Scheduling and synchronization algorithms in operating system: a survey. Journal of Studies in Science and Engineering, 1(2), 1-16.

Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J. (2021). A comprehensive study of kernel (issues and concepts) in different operating systems. Asian Journal of Research in Computer Science, 8(3), 16-31.

Madan, H. T., MANJUNATHA, H., & Vidyashankar, M. (2025). CPU scheduling algorithms performance analysis in the RISC-V xv6 operating system environment. Journal of Integrated Science and Technology, 13(3), 1053-1053.

Hammi, B., Hachem, J., Rachini, A., Khatoun, R., & Aissaoui, H. (2024, November). Malware detection through windows system call analysis. In 2024 Ninth International Conference On

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Mobile And Secure Services (MobiSecServ) (pp. 1-7). IEEE.

Rana, M. R., & Baul, S. (2024). An Overview of Operating Systems Based on Microkernel Technology and their Essential Components. International Journal of Information Engineering and Electronic Business, 16(6), 10-5815.

Antolak, E., & Pułka, A. (2024). Fault-Tolerant Multitasking System Based on Interleaving of Threads. Electronics, 13(23), 4701.

Sahar, S., Khan, H., Tariq, M. I., Ashraf, A., & Zahra, S. A. (2025). An Interactive System Based on First-Class User-Level Threads: A Systematic Review. In International Conference on Computing & Emerging Technologies (pp. 380-394). Springer, Cham.

Liu, J., Shen, Y., Xu, Y., Sun, H., & Jiang, Y. (2025). Snapcc: Effective file system consistency testing using systematic state exploration. ACM Transactions on Software Engineering and Methodology.

Antolak, E., & Pułka, A. (2024). Power consumption prediction in real-time multitasking systems. Electronics, 13(7), 1347.

Kida, S., Imamura, S., & Kono, K. (2025, February). Revisiting Memory Swapping for Big- Memory Applications. In Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (pp. 33-42).

Engler, D. R., Kaashoek, M. F., & O'Toole Jr, J. W. (1995). The operating system kernel as a secure programmable machine. ACM SIGOPS Operating Systems Review, 29(1), 78-82.

Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J. (2021). A comprehensive study of kernel (issues and concepts) in different operating systems. Asian Journal of Research in Computer Science, 8(3), 16-31.

Rodriguez-Zurrunero, R., Utrilla, R., Rozas, A., & Araujo, A. (2019). Process management in IoT operating systems: Cross-influence between processing and communication tasks in end- devices. Sensors, 19(4), 805.

Russo, V., Johnston, G., & Campbell, R. (1988, January). Process management and exception handling in multiprocessor operating systems using object-oriented design techniques. In Conference proceedings on Object-oriented programming systems, languages and applications

Hauser, C., Jacobi, C., Theimer, M., Welch, B., & Weiser, M. (1993). Using threads in interactive systems: A case study. ACM SIGOPS Operating Systems Review, 27(5), 94-105

Okonta, O. E., Ajani, D., Owolabi, A. A., Imiere, E. E., & Uzomah, L. (2015). Performance evaluation of hyper threading technology architecture using Microsoft operating system platform. West African Journal of Industrial and Academic Research, 15(1), 52-67

7. Ismael, G. A., Salih, A. A., AL-Zebari, A., Omar, N., & Merceedi, K. J. (2021). Scheduling Algorithms Implementation for Real Time Operating Systems: A Review. Asian Journal of Research in Computer Science, 11(4), 35-51

8. Gupta, A., Tucker, A., & Urushibara, S. (1991, April). The impact of operating system scheduling policies and synchronization methods of performance of parallel applications. In Proceedings of the 1991 ACM SIGMETRICS conference on Measurement and modeling of computer systems (pp. 120-132)

9. Shakor, M. Y. (2021). Scheduling and synchronization algorithms in operating system: a survey. Journal of Studies in Science and Engineering, 1(2), 1-16

10. Turner, N. (1986). A Simple Multitasking Operating System for Real-Time Applications. Dr. Dobb's Journal, 1, 44-58

Kivity, A., Laor, D., Costa, G., Enberg, P., Har'El, N., Marti, D., & Zolotarov, V. (2014).

{OSv—Optimizing} the Operating System for Virtual Machines. In 2014 usenix annual technical conference (usenix atc 14) (pp. 61-72).

Marufuzzaman, M., Al Karim, S., Rahman, M. S., Zahid, N. M., & Sidek, L. M. (2019). A review on reliability, security and memory management of numerous operating systems. Indonesian Journal of Electrical Engineering and Informatics (IJEEI), 7(3), 577-585.

Di Gennaro, I., Pellegrini, A., & Quaglia, F. (2016, May). OS-based NUMA optimization: Tackling the case of truly multi-thread applications with non-partitioned virtual page accesses. In 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (pp. 291-300). IEEE.

Thyagaturu, A. S., Shantharama, P., Nasrallah, A., & Reisslein, M. (2022). Operating systems and hypervisors for network functions: A survey of enabling technologies and research studies. IEEE Access, 10, 79825-79873.

Russo, V., Johnston, G., & Campbell, R. (1988, January). Process management and exception handling in

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

multiprocessor operating systems using object-oriented design techniques. In Conference proceedings on Object-oriented programming systems, languages and applications (pp. 248-258).

Waldspurger, C. A. (2002). Memory resource management in VMware ESX server. ACM SIGOPS Operating Systems Review, 36(SI), 181-194.

Date, S. P. (1986). A multi-tasking hierarchical file system for VM/CMS using virtual machine communication (Master's thesis, Dept. of Computer Science.University of Houston-University Park).

Bendele, C. (2010). Core and process management for future many-core architectures.

Date, S. P. (1986). A multi-tasking hierarchical file system for VM/CMS using virtual machine communication (Master's thesis, Dept. of Computer Science.University of Houston-University Park).

Bendele, C. (2010). Core and process management for future many-core architectures.

Date, S. P. (1986). A multi-tasking hierarchical file system for VM/CMS using virtual machine communication (Master's thesis, Dept. of Computer Science.University of Houston-University Park).

Bendele, C. (2010). Core and process management for future many-core architectures.

Date, S. P. (1986). A multi-tasking hierarchical file system for VM/CMS using virtual machine communication (Master's thesis, Dept. of Computer Science.University of Houston-University Park).

Bendele, C. (2010). Core and process management for future many-core architectures.

Budzinski, R. L., & Davidson, E. S. (1981). A comparison of dynamic and static virtual memory allocation algorithms. IEEE Transactions on software Engineering, (1), 122-131

Porter, D. E., & Witchel, E. (2010, July). Transactional system calls on Linux. In Linux Symposium (p. 231

Ben-Yehuda, M., Breitgand, D., Factor, M., Kolodner, H., Kravtsov, V., & Pelleg, D. (2009, June). NAP: a building block for remediating performance bottlenecks via black box network analysis. In Proceedings of the 6th international conference on Autonomic computing (pp. 179- 188

Mazer, M. S., & Tardo, J. J. (1994, December). A client-side-only approach to disconnected file access. In 1994

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

First Workshop on Mobile Computing Systems and Applications (pp. 104- 110). IEEE