

All-in-One Security Locking Operating System

Nathan Bellew

Colorado Technical University

CS630: Modern Operating Systems

December^{19th}, 2025

Project Outline	2
Introduction	3
Overview	3
OS Processor and Core	3
Benefits of Switching	4
Steps Required to Upgrade.....	4
Requirements	5
Scheduling Algorithms	6
Summary	7
Preemptive Priority Scheduling (PPS) Algorithm and Multilevel-Feedback Queue (MFQ)	7
PPS Description	7
MFQ Description	8
Rationale for Hybrid	8
OS Concurrency Mechanism	9
Event Driven Systems	10
Shared Memory Systems	11
Comparison	12
OS Security Risks and Mitigation Strategy	12
Summary	13
Security Concerns.....	13
Identified Security Risks.....	13
Risk 1: Priority Inversion.....	14
Risk 2: Shared Memory Data Corruption	14
Risk 3: Unauthorized Privilege Escalation	14
Risk 4: Denial-of-Service via Event Flooding	15
Risk 5: Insecure Update	15
Conclusion on Security.....	15
Future Considerations Using Emerging Technology	16
Summary of Future Benefits.....	17
Future Considerations Using Emerging Technology	17

Micro-VM Architectures	17
Edge Computing	18
Trusted Execution Environments (TEE)	18
References.....	19

Project Outline

Introduction

Really Cool Locks LLC (RCL) is a hypothetical company focused on security of information and technology. Their existing solution revolves around a single-core operating setup where a card reader directly unlocks a physical lock. While this system works, there is a clear opportunity to upgrade to a more sophisticated system. I believe to have a design incorporate more advanced, multi-functional security features all within a single, cohesive locking system.

Overview

This Project sets out to illustrate how RCL can transition from their current basic model to a more modernized operating environment. By doing so RCL would be able to incorporate a wider variety of tools for security such as cameras, voice activation, and machine learning. The possibilities are endless when multiple central processing units (CPU) can be utilized. The following sections will go over the design of the operating system including security risks as well as future considerations.

OS Processor and Core

Benefits of Switching

A multi-core system enables effective use of virtualization technologies such as Virtual Machines (VM). With sufficient cores and memory, RCL can host multiple VMs or containers on a single physical platform. For example, one container dedicated to managing emergency power failures and another focused on long-term logging for intermittent analysis. While still reserving additional resources for future services and system expansion.

Upgrading RCL's operating environment to a multi-processor, multi-core configuration provides a foundation for running several security-critical workloads concurrently without degrading performance. Instead of a single processor handling card reads, lock control, logging, and any future features, multiple cores can divide responsibilities. This separation helps ensure that latency-sensitive operations, such as unlocking a door, are not delayed by background tasks.

Steps Required to Upgrade

In order to move from a single-core system to a multi-core system, RCL must follow a set of steps that will ensure a fluid transition. The following is ordered from which should be done first to last sequentially:

- 1. Workload Assessment**

RCL must first identify current and projected workloads, that is the number of locks and card readers, locations available for upgrade, frequency of use, and facilitate a location to setup new devices. This assessment will define how many cores, how much memory, and what level of input/output (I/O) will be required to perform.

- 2. Hardware Upgrade**

Based on the assessment, RCL will select a server-class microcomputer or a set of embedded controllers with multi-core CPUs that support symmetric multiprocessing (SMP) and hardware virtualization extensions. This hardware should be small enough to help maintain the size of the card reader, while also providing fast and reliable speeds.

- 3. Firmware Upgrade**

A new operating system may require a Firmware Upgrade, if the current firmware does not allow for any non-sequential processing. If a new Hardware is implemented it will almost always require a new firmware.

- 4. Software Upgrade**

This could include containers, VMs, logging management, monitoring, video feed management, or any security requirements needed to ensure RCL data. Upgrading software includes updating kernels, operating systems (OS), and any virtualization needed to run them.

- 5. Final Integration and Monitoring**

This will be what RCL will do in order to manage the system long term and how that final integration will move.

Requirements

By switching to a multi-core system, RCL stands to fulfill the following requirements:

- **Requirements Regarding Scalability**

As RCL expands its security ecosystem with additional locks, cameras, sensors, and analytics tools, the upgraded multi-core architecture ensures the system can absorb these new workloads without strain. By enabling true concurrent processing, RCL can introduce new security capabilities—such as real-time video analysis, biometric authentication, or advanced event logging—without requiring major hardware replacements or disruptive firmware reflashing. This design allows the platform to scale smoothly as demand grows, supporting near-term expansion.

- **Requirements Regarding Virtualization**

This new system would support hosting virtualization environments such as micro VMs or containers to help facilitate the OS functionality. This enables centralized management of policies and updates while keeping latency-sensitive functions close to the physical locks.

- **Security Requirements**

RCL faces increasing concerns about monitoring and validating foot traffic across secured facilities. The upgraded system positions RCL to reliably verify identity, track movement frequency, and enforce access policies in real time. With expanded processing capability and integrated sensors, the platform can accurately identify individuals across multiple security zones and even anticipate movement patterns that may indicate unauthorized behavior. This strengthens overall site security while giving RCL the precision and visibility needed to protect high-value areas.

- **Requirements Regarding Innovation and Future Proofing**

The multi-core platform provides the necessary headroom for future enhancements such as advanced authentication methods, on-device machine learning, and deeper integration with enterprise identity systems. By designing the environment with surplus processing capacity and modular services, RCL can adopt emerging security technologies without needing major hardware replacements or disruptive architectural changes. This ensures the platform remains adaptable and capable of supporting long-term innovation.

Scheduling Algorithms

Summary

Most commercial locks still rely on simplified single-queue scheduling models that don't account for complex emergencies. A good example would be a protocol that tells the system to lock all doors during an emergency shutdown. Certain personnel—like security staff or responders—still need instant access. With a basic scheduler, the system may struggle to handle both demands at the same time, leading to delays or situations where the lock can't respond quickly enough. This forces companies to build complicated workarounds instead of solving the core scheduling problem.

RCL's hybrid model removes this bottleneck. By placing critical operations into a Preemptive Priority Scheduling (PPS) layer and handling everything else through an adaptive Multilevel-Feedback Queue (MFQ), the system avoids high-impact delays. Emergency actions never wait behind routine tasks, and background processes no longer compete with mission-critical events.

Together, these mechanisms create a platform that is not only faster, but fundamentally more secure and more reliable than traditional access-control systems.

Preemptive Priority Scheduling (PPS) Algorithm and Multilevel-Feedback Queue (MFQ)

Upgrading RCL's operating system to a multi-core platform requires a scheduling model that can correctly prioritize life-safety events while still managing background workloads efficiently. The combination of **Preemptive Priority Scheduling (PPS)** and a **Multilevel-Feedback Queue (MFQ)** offers the most effective balance of responsiveness, control, and scalability for RCL's new security ecosystem.

PPS Description

PPS is a real-time scheduling strategy in which “tasks with higher priority immediately preempt lower-priority tasks upon arrival, ensuring bounded and predictable response times for critical operations” (Gao, Huang, & Xu, 2021, p. 114234). In cyber-physical security environments, priority-based real-time scheduling significantly reduces response latency to high-risk events (Hu et al., 2020). Intrusion detection and response systems depend on priority-driven, preemptive execution to interrupt ongoing processes and react immediately to security anomalies (Al-Shaer & Jafarian, 2019). Preemptive priority handling minimizes the attack window by ensuring that high-priority alarms, tamper events, or override requests take precedence over CPU-intensive or routine background tasks (Gao et al., 2021). Modern safety monitoring frameworks further show that PPS enables deterministic execution ordering of emergency events, which is essential for

physical security reliability (Chen & Zhao, 2022). There are recent studies on access control systems for smart buildings that confirm emergency unlock signals, fire overrides, and forced-entry alerts must be placed at the top of a priority hierarchy to ensure immediate system reaction, and PPS provides that exact execution model (Nguyen et al., 2023).

MFQ Description

MFQ automatically restructures task priority levels according to how CPU-bound or interactive each process becomes during execution (Zheng & Gill, 2018). This dynamic adjustment makes MFQ especially suitable for security systems where routine processes need responsive performance without compromising overall throughput. MFQ's fairness properties further prevent starvation, ensuring that long-running tasks such as analytics, logging, or system synchronization still receive CPU time even during peak operational loads (Rezaei & Dehghantanha, 2019). Studies of hybrid workload optimization demonstrate that MFQ improves efficiency when handling concurrent monitoring, periodic data updates, and computationally intensive modules like facial recognition or camera processing (Xie et al., 2021). In addition, MFQ's hierarchical queue structure scales effectively as new sensors, machine-learning inference modules, or background analytics pipelines are added to a system, making it well-aligned with long-term expansion in cyber-physical security deployments (Dinechin et al., 2020).

Rationale for Hybrid

RCL's upgraded locking system must handle a wide range of tasks at the same time: real-time unlocks, emergency overrides, sensor and vitals monitoring, biometric checks, system health reporting, and all the background logging that keeps the platform compliant and auditable. These workloads all matter, but they don't all matter equally.

That's exactly why pairing PPS with MFQ makes sense for this environment. PPS guarantees that the highest-priority events always run immediately. Like emergency safety interruptions or urgent access requests. MFQ then takes everything else and manages it intelligently without letting routine workloads get in the way. All of the normal day-to-day processes, like card scans, PIN entry, or facial recognition, fit cleanly in the MFQ layer. Even when the system is logging heavily or handling multiple background tasks, MFQ keeps things efficient and prevents lower-priority operations from blocking the important ones.

When paired with PPS, MFQ ensures that RCL's emergency tasks run immediately while all non-urgent operations are handled adaptively, efficiently, and without starvation.

OS Concurrency Mechanism

RCL's locking system operates on a compact micro-PC, engineered to use as little memory and storage as possible while still supporting modern multi-core processing. Although many micro-PCs today offer substantial performance, this design assumes a small, embedded platform capable of running a lightweight 4-core operating system. This upgrade replaces RCL's current single-core architecture and enables true concurrent execution across multiple security functions. Because concurrency is central to how the modernized system will operate, the OS must allocate core responsibilities intelligently ensuring that latency-critical tasks (such as lock control and sensor input) are isolated from heavier processes (such as logging, analytics, or camera handling). The following design outlines how each core will be utilized within this 4-core minimum configuration and how concurrency will shape the behavior of the upgraded locking environment. The following design shows how they will be implemented:

Event Driven Concurrency (EDC) Cores:

- Core 1: Emergency Responses and Safety Events
- Core 2: Base Level Events (Normal User Inputs)

Shared Memory Concurrency (SMC) Cores:

- Core 3: Authentication and Sensor Processing
- Core 4: General Logging and Network Connectivity

Each of these cores serves a specific purpose; this system can be implemented using more cores which would allow for more features. Core 1 is the Emergency Response and Safety Events core; this means any urgent responses that need to take place have their own core to run in. Because safety events are given their own core, certain systems can be down while authentication can remain active. This allows Core 2 to always be active. Core 2 is the Base Level Event core; these events are the daily traffic that might cross the RCL lock. Core 3 is Authentication and Sensor Processing. When something in Core 2 is activated, it sends a request to Core 3. This request is stored in memory using SMC to convert the request into an action; this action can include logging events, beginning emergencies, or unlocking the device. Core 4 is the General Logging and Network Connectivity core; it allows the device to connect to a network for further monitoring and to receive updates.

This design uses a combination of EDC and SMC. Both offer a set of valuable traits that make them undeniably perfect for an RCL lock. The following is the rationale for using either EDC-SMC and a comparison for the two that shows why the combination works.

Event Driven Systems

EDC structures system execution around discrete events such as interrupts, sensor readings, or incoming messages (Schmidt 1994). Which means that instead of relying on continuously running threads, event driven systems dispatch handlers only when an event occurs. This architecture is a form of a reactive system design that minimizes overhead by avoiding unnecessary context switching (Schmidt 1994). EDC has vital strengths that should be discussed as well as weaknesses that should be mentioned.

There are three core strengths of EDC: low overhead, high responsiveness, and scalability under load. The event-driven architecture of Flash avoids the high context-switching overheads associated with SMC by dispatching handlers immediately upon event demultiplexing, reducing system latency (Schmidt, 1994). This means EDC provides fast response times because handlers run instantly upon event of arrival and avoid unnecessary work. Event-driven architectures demonstrate significantly better scalability under high load than thread-based servers due to reduced scheduling and synchronization overheads (Welsh, Culler, & Brewer, 2001). This shows that EDC system can provide scalability under heavy loads due to staged execution and non-blocking event queues.

EDC possesses a fatal flaw that causes many users to look for a different method of concurrency. Event-driven systems suffer when event handlers perform long-running computations, as such work can block the event's queue and degrade responsiveness (Welsh, Culler, & Brewer, 2001, p. 231). If an event handler takes too long, the whole event loop stalls, preventing the lock from reacting to new inputs.

Shared Memory Systems

SMC is a model in which multiple threads or processors communicate by accessing a common memory space, coordinating through synchronization mechanisms such as locks or atomic operations (Herlihy & Shavit, 2012; Adve & Gharachorloo, 1996). This means that tasks exchange information by reading and writing shared variables rather than sending messages, allowing extremely fast data transfer between concurrent operations. Herlihy and Shavit (2012) emphasize that shared memory models are central to multicore computing because they support fine-grained parallelism and efficient collaboration across threads. SMC has important strengths that should be discussed, as well as weaknesses that must be acknowledged.

There are three core strengths of SMC: low-latency communication, high throughput on multicore processors, and efficient coordination of shared state. Shared-memory systems provide extremely fast communication because data is exchanged directly without message-passing overhead, and Herlihy and Shavit (2012) note that shared-memory multiprocessors achieve “the highest communication efficiency” for this reason. Shared memory models also support high

throughput in multicore environments where multiple cores must collaborate. Anderson, Calandrino, and Devi (2017) show that shared memory multiprocessing significantly improves system throughput by reducing inter-task communication costs. Additionally, shared memory architectures allow threads to coordinate complex shared data structures using synchronization primitives. Adve and Gharachorloo (1996) explain that shared-memory systems support fine-grained concurrency when proper consistency and synchronization techniques are applied, demonstrating that SMC is well-suited for tightly coupled computational tasks. This architecture allows threads to coordinate complex shared data structures using synchronization primitives. Adve and Gharachorloo (1996) explain that shared-memory systems support fine-grained concurrency when proper consistency and synchronization techniques are applied, demonstrating that SMC is well-suited for tightly coupled computational tasks.

SMC also introduces a significant weakness: maintaining a correct and efficient memory-consistency model. Because multiple threads operate the same address space, they may observe updates in different orders unless the system enforces explicit consistency rules. Adve and Gharachorloo (1996) explain that shared-memory multiprocessors must manage both hardware and software memory-ordering constraints to ensure correct program behavior, and that these consistency requirements introduce substantial design complexity

Comparison

In comparison, EDC and SMC each address different aspects of concurrency: EDC offers fast, low-overhead reactions to external events, while SMC provides high-throughput parallel processing and efficient shared-state coordination. EDC struggles with long-running tasks, and SMC introduces memory-consistency complexity, but together they compensate for each other's weaknesses. A combined EDC–SMC approach therefore delivers a more balanced and effective concurrency strategy for small embedded systems that require both rapid responsiveness and efficient multicore processing. EDC could handle a set of cores that can handle short event logic, and SMC could handle the longer more difficult processes. SMC would then cover EDC's major weakness, and EDC can ensure that the majority of tasks are done quickly and efficiently.

OS Security Risks and Mitigation Strategy

Summary

In RCL's proposed transition to a multi-core operating system, several security risks must be carefully evaluated before deployment. Introducing concurrent execution, shared memory, and virtualization expands the system's attack surface and increases the potential impact of operating system failures. The primary security risks identified in this design include scheduling interference, shared memory corruption, privilege escalation, denial-of-service conditions, and insecure system updates or logging mechanisms. Each of these risks is addressed with a corresponding mitigation strategy designed to reduce likelihood and impact. When applied together, these mitigations significantly lessen overall system risk and support the secure operation of RCL's upgraded locking platform.

Security Concerns

Security in an operating system context refers to the mechanisms and policies used to preserve confidentiality, integrity, and availability (CIA) of system resources while preventing unauthorized access or misuse (Silberschatz et al., 2020). In cyber-physical systems such as RCL's locking platform, operating system security is especially critical because software failures can directly result in physical security breaches. Unlike traditional desktop systems, embedded access-control platforms must defend against both software-based attacks and real-world safety threats, making OS-level risk assessment essential.

Operating system security focuses on protecting core system components that manage execution, memory, I/O, and communication. According to NIST and standard OS literature, a risk assessment should evaluate the following OS areas (Stallings, 2018; Silberschatz et al., 2020):

- Priority Inversion
- Shared Memory Concurrency (SMC)
- Unauthorized Privilege Escalation
- Denial-of-Service via Event Flooding
- Insecure Update

In RCL's system, these areas are tightly coupled with real-time scheduling, shared memory concurrency, and event-driven execution. Weaknesses at the OS level can allow attackers to delay emergency actions, manipulate shared state, or compromise authentication of workflows.

Identified Security Risks

Risk 1: Priority Inversion

Description:

Priority inversion occurs when a lower-priority task blocks a higher-priority task by holding a shared resource, causing critical processes to miss deadlines (Sha et al., 1990). In RCL's environment, this could delay emergency unlocks or safety overrides if background tasks interfere with PPS-managed operations.

Mitigation Strategy:

RCL mitigates this risk by using Preemptive Priority Scheduling (PPS) combined with priority inheritance protocols, ensuring that lower-priority tasks temporarily inherit higher priority when holding shared resources. This approach is well-established in real-time systems and prevents blocking mission-critical operations (Gao et al., 2021).

Risk 2: Shared Memory Data Corruption

Description:

SMC allows multiple cores to access common memory regions, which introduces the risk of race conditions and inconsistent state if synchronization is improperly enforced (Adve & Gharachorloo, 1996). In RCL's system, corrupted shared memory could result in incorrect authentication decisions or stale sensor data.

Mitigation Strategy:

The system enforces strict synchronization primitives, including mutexes and atomic operations, and confines shared memory usage to well-defined data structures. Memory consistency rules are explicitly enforced at the OS level, reducing ambiguity in data visibility across cores (Herlihy & Shavit, 2012).

Risk 3: Unauthorized Privilege Escalation

Description:

Privilege escalation occurs when a process gains access beyond its intended authorization level, often due to improper isolation or flawed access control mechanisms (Stallings, 2018). In an access-control system, this could allow attackers to bypass authentication or issue unauthorized unlock commands.

Mitigation Strategy:

RCL's OS design enforces strict privilege separation, isolating lock control, authentication, logging, and networking into separate execution domains or containers. Only the minimal required privileges are assigned to each service, following the principle of least privilege, which significantly reduces the attack surface (Saltzer & Schroeder, 1975).

*Risk 4: Denial-of-Service via Event Flooding***Description:**

Event-Driven Concurrency systems are vulnerable to denial-of-service attacks if attackers flood the system with excessive events, overwhelming event queues, and delaying legitimate requests (Welsh et al., 2001). For RCL, this could prevent valid unlock or emergency signals from being processed.

Mitigation Strategy:

RCL mitigates this risk by dedicating cores to emergency and base-level events and enforcing event rate limiting at the OS level. High-priority PPS tasks remain isolated from event flooding, ensuring emergency operations cannot be starved by malicious or malformed inputs.

*Risk 5: Insecure Update***Description:**

OS updates and logging subsystems are frequent attack targets. Compromised update paths can introduce malicious code, while insecure logging can expose sensitive access records (Behl & Behl, 2017). In RCL's system, logs contain sensitive access and movement data.

Mitigation Strategy:

All updates are authenticated using cryptographic signatures, and logs are protected using write-once and append-only policies. tasks are Logging isolated to a dedicated SMC core, preventing interference from authentication or event handling processes. This design ensures integrity and non-repudiation of security records.

Conclusion on Security

Security in RCL's modernized operating system is not an add-on feature but a foundational design requirement. By evaluating scheduling, memory management, privilege separation, concurrency models, and system services, this risk assessment identifies realistic threats that directly impact physical safety and access integrity. The combination of **PPS, MFQ, EDC, and**

SMC, when paired with established OS security principles, provides a resilient and scalable defense against both software-based and operational threats. This layered mitigation approach ensures that RCL's locking platform remains responsive, secure, and reliable in high-risk environments.

Future Considerations Using Emerging Technology

Summary of Future Benefits

Emerging technologies provide RCL with a clear path for extending its operating system beyond current multi-core and scheduling improvements. Micro-VM architectures enable strong hardware-enforced isolation between lock control and auxiliary services, allowing new features to be added without increasing systemic risk. Edge computing shifts critical decision-making to the lock itself, reducing latency, improving reliability during network disruptions, and limiting exposure of sensitive data. Trusted Execution Environments further strengthen the platform by protecting credentials and cryptographic operations at the hardware level, even if the main operating system is compromised. Together, these technologies enhance fault containment, responsiveness, and security without disrupting real-time execution guarantees. They also align with RCL's long-term goals of scalability, resilience, and regulatory readiness. By planning these architectures as future enhancements, RCL positions its locking system to evolve safely while maintaining deterministic and secure operation.

Future Considerations Using Emerging Technology

As RCL continues to evolve beyond a traditional access-control system, future operating system requirements will be shaped by increasing computational demand, stronger security expectations, and tighter integration with enterprise infrastructure. While the proposed multi-core architecture with hybrid scheduling and concurrency mechanisms addresses current operational challenges, emerging computing technologies offer additional opportunities to extend system functionality, resilience, and scalability. This section explores three emerging technologies that have the potential to support RCL's future strategic requirements: micro-VM architectures, edge computing, and trusted execution environments.

Micro-VM Architectures

A micro-virtual machine (micro-VM) architecture is a virtualization approach that combines the strong isolation properties of traditional virtual machines with the low overhead and fast startup characteristics typically associated with containers. Micro-VMs use hardware-assisted virtualization extensions to run workloads in isolated virtual environments while intentionally minimizing device emulation, memory footprint, and management complexity (Agache et al., 2020). Unlike full VMs, which emulate a wide range of hardware devices and incur significant startup and runtime overhead, micro-VMs expose only a minimal set of virtualized resources necessary for secure execution.

This design enables micro-VMs to achieve near-native performance while maintaining strong isolation boundaries enforced by the processor and hypervisor. This capability directly advances RCL's locking platform by ensuring that the core lock control logic operates in a hardened execution boundary, protected from faults or compromise in adjacent services. This makes micro-VM support a strategically valuable investment that enables feature growth, improves resilience, and protects the integrity of the lock as the system evolves.

Edge Computing

Edge computing is a distributed computing architecture in which data processing and decision-making are performed close to the data source rather than being sent to centralized cloud infrastructure. Instead of transmitting raw sensor data to remote servers for analysis, edge systems execute computation locally on embedded devices or nearby controllers, significantly reducing communication latency and bandwidth dependence (Shi et al., 2016). Advances in embedded processors, system-on-chip (SoC) designs, and energy-efficient accelerators have made it feasible to deploy complex analytics directly at the device level without relying on constant cloud connectivity.

For RCL, edge computing directly enhances the lock system by allowing security decisions to be made locally. Core functions such as identity verification, access policy enforcement, anomaly detection, and sensor fusion can be executed at the lock or controller itself rather than being deferred to external services. This eliminates round-trip network delays that could otherwise slow unlock operations or emergency overrides. Edge computing also significantly improves system reliability and fault tolerance. If network connectivity to a central server is disrupted, the lock remains fully operational, enforcing access rules based on locally available data and policies. By investing in edge capabilities, RCL positions its platform to deliver faster response times, improved resilience, and stronger data protection.

Trusted Execution Environments (TEE)

A Trusted Execution Environment (TEE) is a secure area of a processor that provides an isolated execution context for sensitive code and data, protected from the rest of the system, including the operating system, hypervisor, and other privileged software. TEEs rely on hardware-enforced memory isolation, secure boot mechanisms, and cryptographic attestation to ensure that only trusted code can be executed within the protected environment (Sabt, Achemlal, & Bouabdallah, 2015). Examples of TEE implementations include secure enclaves that restrict access to designated memory regions and prevent unauthorized inspection or modification. TEEs establish trust at the hardware level. If the main operating system is compromised, code running inside a TEE remains confidential and tamper resistant.

For RCL, TEEs directly enhance the security of the lock by hardening the most sensitive operations against compromise. In an access-control system, compromise of credentials or cryptographic logic can result in catastrophic security failure. This approach significantly raises the difficulty of both software-based and physical attacks, providing strong resistance against reverse engineering and tampering. Planning for TEE support as a future enhancement allows RCL to introduce stronger authentication mechanisms, support regulatory compliance, and meet increasing enterprise security expectations without redesigning the system architecture. By anchoring critical security operations in protected execution environments, RCL gains a decisive advantage in safeguarding credentials, maintaining integrity, and preserving trust in environments where compromise is a realistic threat.

References

- Agache, A., Brooker, M., Iordache, A., Liguori, A., Neugebauer, R., Piwonka, P., & Popa, R. A. (2020).
Firecracker: Lightweight virtualization for serverless applications. Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 419–434.
<https://www.usenix.org/conference/nsdi20/presentation/agache>
- Adve, S. V., & Gharachorloo, K. (1996).
Shared memory consistency models: A tutorial. *Computer*, 29(12), 66–76.
<https://doi.org/10.1109/2.546611>
- Al-Shaer, E., & Jafarian, J. H. (2019).
Security-aware real-time scheduling for intrusion response systems. *IEEE Transactions on Dependable and Secure Computing*, 16(5), 789–802.
<https://doi.org/10.1109/TDSC.2017.2754245>
- Anderson, J. H., Calandrino, J. M., & Devi, U. C. (2017).
Real-time scheduling on multicore platforms. *Real-Time Systems*, 43(2–3), 1–33.
<https://doi.org/10.1007/s11241-009-9081-9>
- Behl, A., & Behl, K. (2017).
Cyberwar: The next threat to national security and what to do about it. Oxford University Press.
- Chen, W., & Zhao, F. (2022).
Priority-based secure scheduling for real-time safety monitoring systems. *Journal of Systems Architecture*, 124, 102430.
<https://doi.org/10.1016/j.sysarc.2022.102430>
- Gao, J., Huang, L., & Xu, H. (2021).
Preemptive real-time scheduling for time-critical security services in embedded systems. *IEEE Access*, 9, 114233–114245.
<https://doi.org/10.1109/ACCESS.2021.3104587>
- Herlihy, M., & Shavit, N. (2012).
The art of multiprocessor programming. Morgan Kaufmann.
- Sabt, M., Achemlal, M., & Bouabdallah, A. (2015).
Trusted execution environment: What it is, and what it is not. Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 57–64.
<https://doi.org/10.1109/Trustcom.2015.357>
- Saltzer, J. H., & Schroeder, M. D. (1975).
The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278–1308.
<https://doi.org/10.1109/PROC.1975.9939>
- Schmidt, D. C. (1994).
Reactor: An object behavioral pattern for demultiplexing and dispatching handles for

synchronous events. In J. O. Coplien & D. C. Schmidt (Eds.), *Pattern languages of program design* (pp. 529–545). Addison-Wesley.

Sha, L., Rajkumar, R., & Lehoczky, J. P. (1990).

Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9), 1175–1185.

<https://doi.org/10.1109/12.57058>

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016).

Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637–646.

<https://doi.org/10.1109/JIOT.2016.2579198>

Silberschatz, A., Galvin, P. B., & Gagne, G. (2020).

Operating system concepts (10th ed.). Wiley.

Stallings, W. (2018).

Operating systems: Internals and design principles (9th ed.). Pearson.

Welsh, M., Culler, D., & Brewer, E. (2001).

SEDA: An architecture for well-conditioned, scalable Internet services. *Proceedings of the ACM Symposium on Operating Systems Principles*, 230–243.

<https://doi.org/10.1145/502034.502057>