Security in Real-Time Operating Systems vrs. General Purpose Operating Systems

Colin Smith
Computer Science Department
Missouri University of Science and Technology
Rolla, MO 65409

Riley Carter
Computer Engineering Department
Missouri University of Science and Technology
Rolla, MO 65409

***Abstract*—This project investigates and compares the implementation and management of security in general-purpose operating systems and real-time operating systems. The research focuses on three key areas: access control, real-time constraints, and memory management. For each area, we examine how security is enforced in GPOS versus RTOS, analyze inherent vulnerabilities, and explore the trade-offs between performance and protection. Special attention is given to how the strict timing requirements of RTOS can limit the depth of security checks, potentially exposing them to unique threats.**

I. Introduction

As more technology is developed the imminent threat of being hacked grows greater. As we incorporate more devices into our lives, we incorporate more ways hackers can take advantage of us. This incentives engineers to strongly take security into account when developing technology. In programming, strong security has big tradeoffs. It can often reduce speed and reduce ease of access. Trade offs like these can put limitations on systems that are required to go fast like real time operating systems. Real time operating systems are often used in embedded systems that are required to be fast so often they have to make sacrifices in security for speed. General purpose operating systems on the other hand, are not required to be as fast as a real time operating system. While they still need to be fast for the user, they are used much more in systems that don't have time constraints.

II. General Purpose Operating Systems

General purpose Operating systems (GPOS) are the most common types of operating systems. A few popular GPOSs are Windows, Linux, and MacOS. These operating systems are designed for day-to-day tasks like editing a file or running a browser. Because of the wide variety of jobs the OS may have to support, they must be very versatile. However, this versatility can cost the OS efficiency in many areas compared to other more specific OS such as real time operating systems.

A. Access Control

A big part of security is access control. Access control refers to how a system restricts access to resources. This can include memory, files, or hardware. This is one way to stop unauthorized users from accessing things they shouldn't. One common example is when you download new software off the internet. You will get your executable that you run to install the software. While running the executable, your computer will ask "Do you want to allow this process to make changes to your device?" This is to ensure that a malicious process cant access. Your disk and other important information without you granting it access first. So, for obvious reasons, access control is very important.

In a GPOS access control is handled in a variety of ways. Some systems utilize mandatory access control (MAC) which is a policy enforced across the entire system. With MAC processes are not allowed to share information, or adjust constraints. The privileges are predefined and that's what they are [3]. Another access control method is discretionary access control (DAC). DAC is an access control method which allows users to control who accesses their data [3]. An example of this is if a user created a file, they would have all the permissions for that file like reading and writing. Then if that user decided that they wanted another user to be able to read the file, they could then grant the other user privileges so they can read the file. Then there is also role-based access control (RBAC). With RBAC, multiple roles are defined with varying levels of privileges . Then each user is assigned a certain role. This role determines what they are allowed to do [3].

For each of these access control methods, after deciding who is allowed to do what, the process remains the same. The user with given permission will do something which initiates a instruction. With instructions there are 2 types: privileged and non- privileged. A privileged instruction an instruction that deals with I/O requests or CPU's status or control registers [4]. Privileged instruction are only allowed to be executed by the kernel.

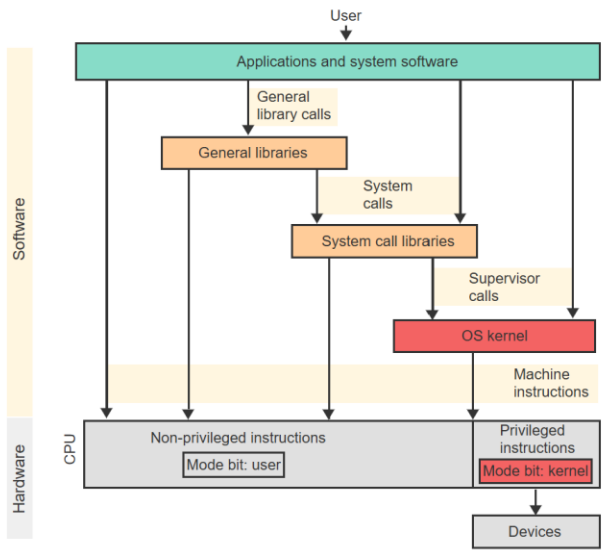

Fig. 1. The OS Hierarchy [4]

In Fig. 1, we can see the process starts with the user using an application/software. If the application then wanted to make a non- privileged call, it would just call a general library or just go straight to making the instruction. In this case the OS would ensure the Mode bit is set to user. If the instruction needed to access I/O or some other resource that requires a privileged instruction, the application would invoke a system call to switch to kernel mode, where it would execute the instruction from, then switch back to user mode. This is where the access control methods come in. When a process issues a system call, the OS will check the users privileges according to the systems control policy. Only if authorized, the OS will then execute the instruction. This ensures that only those with the highest level of access can actually work with the CPU or I/O devices. Without a good access control system, it would be much easier for a malicious user to gain access to the kernel. Having good role policies like MAC, DAC, or RBAC that implement rules system wide and apply to large groups are essential. They greatly help to organize privileges into group helping to ensure malicious users can't utilize privileged operations.

B. *Memory Management*

Memory Management is very complicated in GPOS due to years of optimizing it for efficiency. It has been optimized over many years to become as efficient as possible. As a result, most GPOS systems utilize virtual memory through paging. Paging is the process of partitioning the memory into fixed sized blocks which represent small sections of the file. Virtual memory is being able to load certain pages of a program into memory, waiting to load others until they are needed. This saves the computer space by not loading in parts of a file that are going to be left unused for a while.

Virtual memory not only optimizes efficiency, but also security. Virtual memory ensures memory isolation between processes. This means that each process cannot mess with the memory for another process. This is great for security because if a malicious process starts, it can't go and immediately modify another processes memory. Another benefit of virtual memory and paging is that everything isn't stored in main memory. Because of this, a malicious user with access to main memory will only have a fraction of the data.

III. Real Time Operating Systmes

Real time operating systems (RTOS) are designed for predictable, deterministic, low-latency applications where the security constraints are very different from those in GPOS. RTOS often compromise security and flexibility to achieve the real time constraints required by the system. The application of RTOS requires that systems be configurable per application. Systems like FreeRTOS provide many methods of memory management and access control with varying levels of latency and complexity intended to be applicable for whatever application of RTOS is necessary [2].

A. *Access Control*

RTOS systems are often applied in embedded systems where safety is critical (e.g. medical devices, automotive control systems, industrial controllers). Access control in RTOS ensures reliability, security, and deterministic behavior. Due to the low latency application of RTOS, processes require quick memory access with low overhead. The balance between security and low latency performance is what makes access control a difficult task to implement in these systems. There are plenty of unique mechanisms for access control that depend on the systems latency and security requirements.

Access control is managed in a variety of ways, some systems use process privilege levels where certain tasks are allowed run in privileged or user mode and hardware access is restricted to privileged tasks. Another access control method is physical memory protection [2], where a Memory Protection Unit (MPU) is used to enforce strict memory boundaries for processes. Other methods include capability-based security where tasks are assigned permissions to interact with resources like semaphores or devices. This ensures that only those tasks can perform operations like inter-process communication or peripheral access.

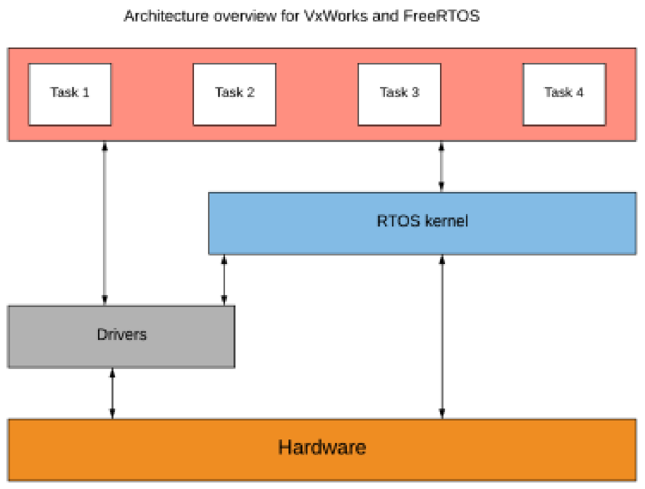

Fig. 2. Arehitecture Overview for VxWorks and FreeRTOS [1]

Additionally, role-based access control (RBAC), applied in more complex RTOS environments, where processes are assigned roles defining their permissions. For example, high priority control tasks are allowed to preempt others but are not allowed to modify system settings. Mandatory access control (MAC) is less common in RTOS due to high overhead and its impact on determinism. However, it may be used in modified GPOS systems, such as RTLinux [1]. DAC is rare in RTOS applications because RTOS lack the need for user permissions found in most GPOS.

Due to the low latency application of RTOS, processes require quick memory access with low overhead. Some implementations of access control in these systems require tasks direct access to hardware. This requires rigid control so that the system will never induce latency with unnecessary security checks and always meet the systems real-time constraints.

B. *Memory Management*

RTOS often prefer static memory allocation over dynamic allocation. This is due to the unpredictable latency and fragmentation caused by dynamic allocation. Static allocation eliminates dynamic allocation overhead and ensures that high priority processes are waiting for memory access as little as possible. However, static allocation limits the size of programs a system can execute.

Some RTOS use dynamic allocation with strict restrictions. Depending on the application, fixed-size stacks may be implemented instead of heap allocation. Advanced OS like FreeRTOS support both static and dynamic allocation depending on their application [2]. FreeRTOS ensures a process cannot access memory that don’t belong to it by The implementing of virtual memory using a Memory Management Units (MMU) [2]

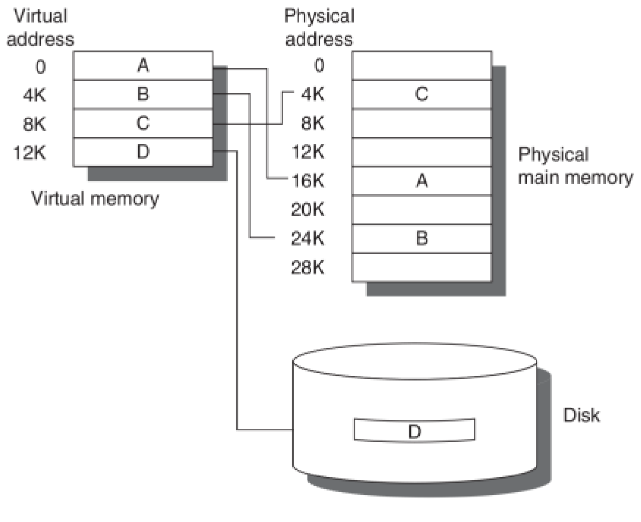

Fig. 3. The logical program in its contiguous virtual address space is shown on the left. It consists of four pages, A, B, C, and D. The actual location of three of the blocks is in physical main memory and the other is located on the disk.

Virtual memory allows the OS to isolate processes in memmory. In the event of a malicious process or unexpected physical memory access, that section of memory is already isolated. RTOS systems do not often use virtual memory. If a malicious process has access to memory in a system that uses static memory allocation, that process has access to all of memory as it is not isolated by the OS.

This is the key issue with security in RTOS. They sacrifice security, isolation, and flexibility for quick deterministic allocation allowing for low latency real time operation.

IV. COMPARISON

A. *Access Control*

RTOS and GPOS use very different methods of access control due to their operational requirements. GPOS typically have complicated user-based security models, using DAC and MAC to control memory access. RTOS access control prioritizes process-level restrictions as opposed to user permissions as the systems typically lack multi-user environments.

GPOS are able to use access control implementations that have large overhead because of the lack of real time constraints in most systems. RTOS use many access control methods that GPOS use in their simplest and quickest implementations. RBAC used in RTOS applies concepts similar to those used in GPOS. However, the real-time implementation of RBAC often focuses on process roles - for instance, allowing a high priority control task to preempt other processes while restricting its access to unrelated kernel functions.

B. *Memory Management*

Memory management has a significant tradeoff between RTOS and GPOS. Since GPOS is not bound by the timing requirements that RTOS is bound by, it can implement virtual memory to isolate processes. On the other hand RTOSs have to avoid virtual memory to maintain the performance requirements.

The difference introduces a major security gap. In GPOS, a malicious process is limited to only the memory in its virtual address space. In RTOS without memory isolation, one process could access all system memory posing a big security threat.

V. CONCLUSION

Security in GPOS and RTOS is fundamentally shaped by their different priorities. GPOS prioritizes flexibility and versatility. GPOS benefits from virtual memory that optimizes memory efficiency and security. In contrast, RTOS prioritizes low-latency operations at the expense of security measures that GPOS has. The strict timing requirements of RTOS limits its ability to implement complex security protocols. Overall, GPOS benefits from better security due to not having the same time constraints as RTOS.

REFERENCES

[1] A. Serino, “A Survey of Real-Time Operating Systems.” Available: https://engineering.lehigh.edu/sites/engineering.lehigh.edu/files/_DEPARTMENTS/cse/research/tech-reports/2019/LU-CSE-19-003.pdf

[2] C. Larmann, “Secure Task Management in FreeRTOS,” *Tudelft.nl*, 2024. https://repository.tudelft.nl/record/uuid:25ac1dc8-7d80-485b-be33-b4f43b6c379e (accessed May 07, 2025).

[3] Halimah Olaolohun Abdul-Azeez, Adeola Anjolajesu, O. Emmanuel, and Y. Idris, “OPERATING SYSTEM ACCESS CONTROL,” May 24, 2024. https://www.researchgate.net/publication/388195320_OPERATING_SYSTEM_ACCESS_CONTROL

[4] L. Bic, “Operating Systems | zyBooks,” *zyBooks*, Apr. 22, 2025. https://www.zybooks.com/catalog/operating-systems (accessed May 07, 2025)