EECS 3221 Report

A Comparison of Real Time Operating Systems and the Linux Operating System

Daniel Di Giovanni — 218204818

February 12, 2024

My signature below attests that this submission is my original work:

Following professional engineering practice, I bear the burden of proof for original work. I have read the York University Senate Policy on Academic Integrity and the EECS Academic Honesty Guidelines and confirm that this work is in accordance with the Policy.

Signature	Date
	February 12, 2024

Contents

Introduction and Background **1**

Overview of Linux **2**

Overview of Real-Time Operating Systems **3**

Conclusion **5**

References **6**

Appendix A Code **8**

A.1 Q1.m 8

Introduction and Background

An operating system is a computing layer that separates the hardware of the computer from the programs that run on it. It provides the *environment* for other programs to do useful work [1, p. 4]. The fundamental tasks of an operating system include allocating resources (such as memory and CPU time), handling the control of input/output (I/O) devices, and ensuring proper usage of the computer and preventing errors [1, pp. 3-5].

The most important part of an operating system is the *kernel*. It is the first program loaded into memory on startup and is the one program that is always running on the computer [1, pp. 6-7, 22]. Along with the kernel, operating systems also include *middleware frameworks* that ease application development, and *system programs* that help the system run but are not part of the ever-running kernel. All of this supports the execution of *application programs*, which are the programs that provide functionality to the end user [1, p. 4, 7]. A diagram of system organization with the kernel is shown in Figure 1.

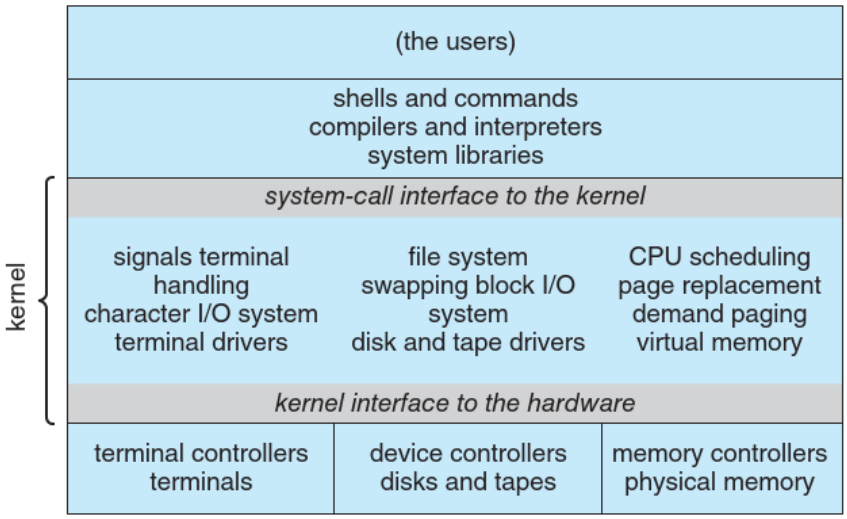

Figure 1: Diagram of a kernel within the operating system [1, p. 82].

In industrial and commercial computing applications, the choice of an operating system is crucial. It affects the performance, security, and maintainability of the system. As an example, consider the secure boot of an embedded system. Secure boot, an important security technique to ensure that the kernel code has not been modified, is often neglected in embedded systems. The absence of secure boot allows the system to boot faster with less memory and energy consumption—at the cost of leaving the boot process and internal software vulnerable. However, it was discovered that the introduction of secure boot software caused boot-up time to increase by only 4%, whereas a hardware implementation of secure boot caused a 36% increase [2, pp. 11-12]. Clearly, the operating system has a significant impact on the overall quality of the system.

Two important classes of operating systems/kernels will be discussed here: the Linux

operating system and real-time operating systems (RTOSs). The Linux kernel is a free and open source implementation of an operating system kernel. It is used ubiquitously not only for desktop computers, but also for servers and embedded devices with a broad range of commercial and industrial applications [3]. The Linux kernel is a tried-and-tested system with high flexibility and extendability. RTOSs are more vague, being defined not by a specific implementation, but by the ability to manage systems with complex time and resource constraints [4]. RTOSs need to be able to meet strict deadlines associated with external events using limited resources. In short, "a real-time system is one whose correctness involves both the logical correctness of the outputs and their timeliness" [5].

The objective of this report is to provide a thorough comparison of the Linux operating system/kernel with RTOS/real-time kernels to aid in the decision of which operating system to use.

Overview of Linux

When "Linux" is referred to, an entire operating system is often being referenced. However, "Linux" is just the kernel. The Linux kernel is used in combination with other software to make a complete operating system. The entire operating system (with the Linux kernel inside) is called a "Linux distribution" (for example, Ubuntu and Debian for PCs) [6]. The ability to extend and modify a Linux operating system is where its flexibility originates.

Since Linux is an open source kernel, anyone can read, use, and modify the code. And since it is just a kernel, it *requires* additional software to be useful. This leads to a wide variety of adaptations of the Linux-based operating systems to fit many different needs [7].

Many Linux distributions are desktop-focused, creating an easy user-interface, similar to Microsoft's Windows and Apple's MacOS, with the ability to run virtually all of the programs expected from a desktop computer. Linux operating systems are also a popular choice for web servers and have become the backbone of enterprise computing [8]. According to the Linux Foundation, Linux powers the majority of the public cloud [9], and some companies, like Amazon Web Services, have developed their own Linux distribution for use in their products [10].

Most pertinent to this report, however, is the use of Linux in embedded applications. While some embedded devices do not employ an operating system (these are called *bare-metal* systems, and they forgo an operating system to conserve resources [11]), most are complex enough to require an operating system. And when an embedded system requires an operating system, Linux is an appropriate choice for its kernel.

The Linux Foundation estimates that 62% of embedded systems use a Linux-based operating system [9]. This is possible because of the high degree of modularity within the Linux kernel, making it easy to configure the kernel to specific hardware [12]. Further, developers of embedded Linux distributions have the ability to exclude packages specific

to desktops, like user systems and GUI environments, opting instead for packages suited for embedded development, like cross-development tools, different types of drivers, and debugging and profiling tools [12]. This extensibility and flexibility makes embedded Linux a great choice.

Overview of Real-Time Operating Systems

Real-time systems are systems with rigid timing requirements [1, pp. 46]. RTOSs can be used to provide a layer of abstraction between the software handling real-time events and the underlying hardware. These operating systems must be high-performance with predictable and deterministic behavior [13].

Real-time systems can be categorized into *soft* and *hard* systems. According to Intel, hard real-time systems refer to systems where missing a deadline means failure of the system [14]. In soft real-time systems, however, the focus in on optimizing quality of service. Missing a deadline would result lower-quality service, not in a system failure [15]. As an example, a robotic arm needs to be instructed to stop *before* it hits a wall, and failure to meet this deadline will cause the arm to break. This means the robotic arm is a hard real-time system [1, pp. 45-46]. A consumer smartphone would be a soft real-time system because while a lagging phone is frustrating to the user, it is not catastrophic.

There are many strategies for implementing a RTOS. The focus is on responding to external (real-time) events in a time period that is minimal and predictable. The time taken to respond to an event is called *dispatch latency*. As shown in Figure 2, dispatch latency consists of many aspects, like interrupt processing, preempting and managing conflicts with other processes, and executing the actual process. Thus, the operating system's process scheduling algorithm is of utmost importance.

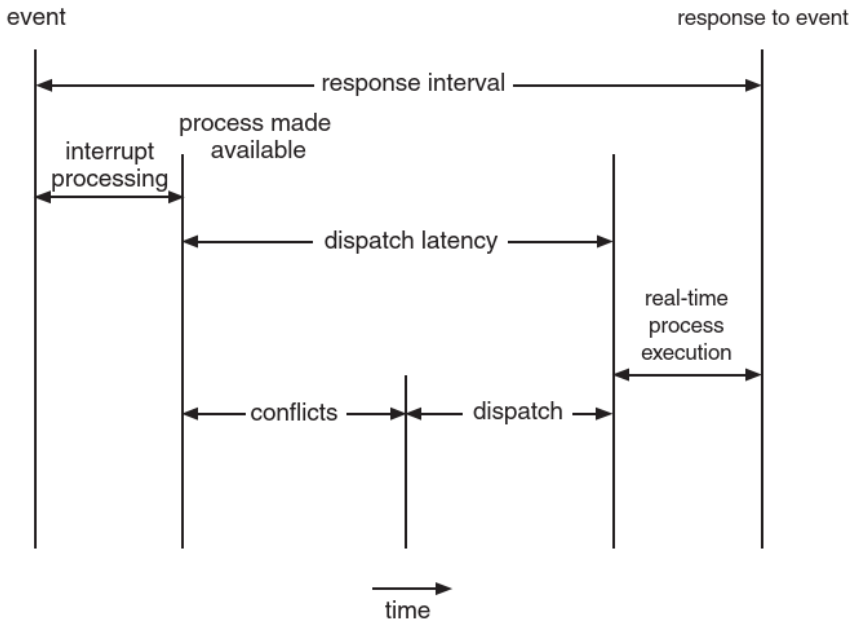

Figure 2: Dispatch latency of an event [1, p. 229].

To meet strict deadline requirements, a priority-based scheduler is necessary. Such a scheduler enables higher-priority processes to preempt lower-priority ones [1, p. 229]. For hard real-time systems, both the CPU burst time of the process and its deadline must be known. When a process is ready to execute, the scheduler can decide whether or not it is possible to meet the process's deadline, and choose to schedule it accordingly. This is called an *admission-control algorithm* [1, pp. 229-230].

Rate-monotonic scheduling can be used to schedule periodic real-time tasks. In this algorithm, processes with shorter periods are assigned higher priorities, since they need the CPU more often. For this type of scheduling to work its CPU burst time must be the same every time it executes, and must be shorter than its deadline and period. A pitfall of rate-monotonic scheduling is that it is not always able schedule processes even if there is enough CPU time for them [1, pp. 230-232].

Another real-time scheduling technique is called *earliest-deadline-first (EDF)*. In this algorithm, higher priority is given to processes whose deadline is sooner. This way, a process near its deadline can preempt a process that has more time to spare. EDF scheduling can achieve a CPU usage near 100%, thus being able to schedule more processes than rate-monotonic scheduling [1, pp. 232-233].

Implementing an RTOSs is more complex than using a Linux-based operating system, but the complexity is the cost for performance. For hard real-time systems, an RTOS is necessary to meet performance requirements. For soft real-time systems, tradeoffs must be balanced to determine whether the improved service quality of the system is worth the added complexity and cost.

Conclusion

References

[1] A. Silberschatz, P. B. Galvin, G. Gagne, *Operating System Concepts*, 10th ed., John Wiley and Sons, Inc., 2018.

[2] J. Ingelhag, “How to choose an operating system for an embedded system”, Örebro Universitet, 2023. Accessed February 9, 2024. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1773441/FULLTEXT01.pdf.

[3] The Linux Foundation, “What is Linux?,” *The Linux Foundation*. Accessed February 7, 2024. [Online]. Available: https://www.linux.com/what-is-linux/.

[4] W. Cedeño, P.A. Laplante. “An Overview of Real-Time Operating Systems,” JALA: Journal of the Association for Laboratory Automation, 2007, ch. 12, sec. 1, pp. 40-45. Accessed February 7, 2024. [Online]. Available: https://doi.org/10.1016/j.jala.2006.10.016.

[5] P. A. Laplante, “Real-Time Systems Design and Analysis,” 3rd ed., Hoboken, NJ, Wiley, 2004, p. 505. Accessed February 7, 2024. [Online]. Available: https://doi.org/10.1002/0471648299.fmatter.

[6] R. Stallman, “Linux and the GNU System,” *Free Software Foundation*. Accessed February 9, 2024. [Online]. Available: https://www.gnu.org/gnu/linux-and-gnu.en.html.

[7] A. Adekotujo, A. Odumabo, A. Adedokun, O. Aiyeniko, “A Comparative Study of Operating Systems: Case of Windows, UNIX, Linux, Mac, Android and iOS,” International Journal of Computer Applications, 2020. Accessed February 10, 2024. [Online]. Available: https://www.researchgate.net/profile/Adedoyin-Odumabo/publication/343013056_A_Comparative_Study_of_Operating_Systems_Case_of_Windows_UNIX_Linux_Mac_Android_and_iOS/links/61f2b50a9a753545e2fe8300/A-Comparative-Study-of-Operating-Systems-Case-of-Windows-UNIX-Linux-Mac-Android-and-iOS.pdf.

[8] Grand View Research, “Server Operating System Market Size, Share & Trends Analysis Report By Operating System (Windows, Linux), By Virtualization (Virtual Machine, Physical), By Deployment, By Region, And Segment Forecasts, 2022 - 2030,” *Grand View Research*, 2020. Accessed February 10, 2024. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/server-operating-system-market-report.

[9] The Linux Foundation, “Linux Runs All of the World’s Fastest Supercomputers,” *The Linux Foundation*, November 20, 2017. Accessed February 10, 2024. [Online]. Available: https://www.linuxfoundation.org/blog/blog/linux-runs-all-of-the-worlds-fastest-supercomputers.

[10] L. Clark, “How Amazon Web Services Uses Linux and Open Source,” *The Linux*

Foundation, September 8, 2014. Accessed February 10, 2024. [Online]. Available: https://www.linux.com/news/how-amazon-web-services-uses-linux-and-open-source/.

[11] M. Salehi, D. Hughes, B. Crispo, "MicroGuard: Securing Bare-Metal Microcontrollers against Code-Reuse Attacks," *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, Hangzhou, China, IEEE, 2019, pp. 1-8, doi: 10.1109/DSC47296.2019.8937667. Accessed February 10, 2024. [Online]. Available: https://doi.org/10.1109/DSC47296.2019.8937667.

[12] P. Raghavan, A. Lad, S. Neelakandan, *Embedded Linux System Design and Development*, Boca Raton, FL, Taylor and Francis Group, LLC, 2006.

[13] A. S. Gillis, "DEFINITION real-time operating system (RTOS)," TechTarget Accessed February 11, 2024. [Online]. Available: https://www.techtarget.com/searchdatacenter/definition/real-time-operating-system

[14] Intel, "Real-Time Systems Overview," Intel. Accessed February 11, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/robotics/real-time-systems.html

[15] G. Lipari, L. Palopoli, "Real-Time scheduling: from hard to soft real-time systems," arXiv, 2015. Accessed February 11, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.1512.01978

Appendix A Code

A.1 Q1.m