ISSN: 2230-9926

Available online at http://www.journalijdr.com

IJDR

International Journal of Development Research
Vol. 13, Issue, 03, pp. 62320-62325, March, 2023
https://doi.org/10.37118/ijdr.26649.03.2023

RESEARCH ARTICLE OPEN ACCESS

A COMPARATIVE STUDY ON HOW TO CREATE AN OPERATING SYSTEM FROM SCRATCH AND A LINUX KERNEL

[1]C G Accamma, [1]Baisakhi Debnath, *[2]Palak Agarwal, [2]Om Jaiswal, [2]Nishant Kumar, [2]Nischal Dhoka and [2]Parag Joshi

[1]Asst.Professor, Center for Management Studies, Jain (Deemed-to-be) University, Bangalore, India
[2]Student, Center for Management Studies, Jain (Deemed-to-be) University, Bangalore, India

ARTICLE INFO

Article History:

Received 17th January, 2023
Received in revised form
06th February, 2023
Accepted 19th February, 2023
Published online 30th March, 2023

KeyWords:

Operating System, Windows, Linux, Mac, UNIX, Android, iOS, Comparative Analysis, quantum computer operating system, Security, Performance.

Corresponding author: Palak Agarwal,

ABSTRACT

There are numerous operating systems available today, each designed to function optimallyon various devices and with their own resource utilization and algorithms. Creating an operating system from scratch would require extensive expertise in the field, as it serves asthe interface to manage multiple user requests, API calls, and channels. Nevertheless, it would be impractical to develop a new operating system today, given the abundance of existing options that are both secure and efficient. However, we can choose from various Linux distributions tailored to our specific needs, such as Kali Linux or Parrot OS for ethicalhacking, Red Hat for server-related tasks, or Cloudera for big data analysis. There is a significant area where a unique operating system is necessary, and that is for supercomputers. These computers operate differently than standard systems, and thus requirea custom operating system to optimize their performance.

Copyright©2023, C G Accamma et al. *This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.*

Citation: ***C G Accamma, Baisakhi Debnath, Palak Agarwal, Om Jaiswal, Nishant Kumar, Nischal Dhoka and Parag Joshi. 2023.*** "Origin, distribution, taxonomy, botanical description, genetic diversity and breeding of tomato (*Solanum lycopersicum* L.)". *International Journal of Development Research*, 13, (03), 62320-62325.

INTRODUCTION

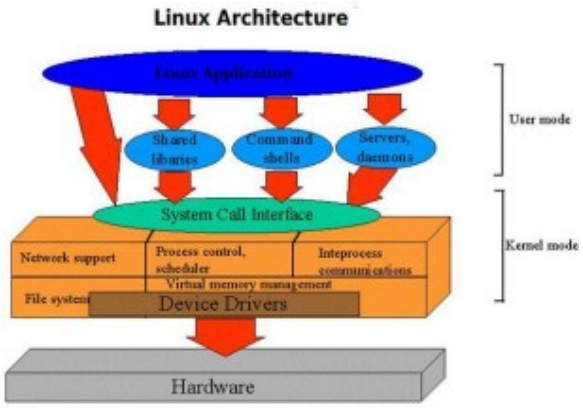
An operating system (OS) is an essential component of modern computer systems, responsible for managing hardware resources and providing services to software applications. Its primary function is to act as a mediator between the applications and the underlying hardware, allowing them to communicate and interact efficiently. OSes play a vital role in offering a stable, secure, and efficient platform for running software applications. There are many different versions of operating systems, each with their own unique features and capabilities, optimized for specific hardware platforms and use cases. Operating systems offer several key benefits, including the ability to run multiple. Applications concurrently, manage system resources, and provide a common user interface forinteracting with applications. They provide an abstract layer that shields applications from the complexities of the underlying hardware, allowing developers to write software that can run on various hardware platforms. Operating systems consist of several layers, including the kernel, device drivers, system libraries, and user interface components. The kernel is the central component of the OS, responsible for managing system resources, such as memory and CPU time, and providing a set of system calls that applications can use to communicate with the operating system.

Device drivers' interface between the OS and network adapters. System libraries provide aset of common functions that applications can call, while UI components provide the graphical interface with which the user interacts.The most used operating systems today are Windows, macOS, and Linux, each with their unique strengths and weaknesses. Windows is a proprietary OS developed by Microsoft and has a market share of more than 80%. It provides a user-friendly interface, supports a wide range of hardware devices, and has a rich ecosystem of software applications. Key features ofWindows include its enhanced security features, compatibility with legacy apps, and support for touch input. MacOS is an operating system developed by Apple for its line of Macintosh computers. It is based on the Unix operating system and shares many features with Linux. MacOS offers a sleek, modern interface and tight integration with other Apple products like iPhone and iPad. Key features of macOS include tight integration with Apple's product ecosystem, emphasis on privacy and security, and support for virtualization technology. Linux is an open-source operating system widely used in enterprise environments, cloud computing, and embedded systems. It is known for its stability, security, and flexibility. It ishighly customizable and can be adapted to fit various use cases. Key features of Linux include support for a wide range of hardware architectures, a strong developer and user community, and a powerful command-line interface. Other notable operating systems include Android, iOS, Chrome OS, and FreeBSD.

Androidis a mobile operating system developed by Google and is the most widely used operating.system for mobile devices. iOS is the operating system developed by Apple for its iPhone andiPad devices. Chrome OS is a lightweight operating system developed by Google for Chromebook laptops. FreeBSD is a free and open-source Unix-like operating system used in many enterprise environments. Overall, operating systems play a critical role in providing the foundation for softwareapplication development and deployment. As computer technology advances, the role of theoperating system will continue to play an important role in providing a stable, secure, and efficient platform for running software applications. There are many different operating.systems in use today, each with their unique features and capabilities, but Windows, macOS, and Linux are the most widely used operating systems, each with strong popularity and a rich ecosystem of software application.

Review of Literature Operating system: Operating systems ideas are covered in detail in the book "Operating Systems: Three Simple Parts" by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. It addresses issues including networking, file systems, memory management, and processes. Another thorough operating systems reference is the book "Modern Operating Systems" by Andrew S. Tanenbaum and Herbert Bos. It covers subjects including networking, file systems, memory management, and process management. The widely used textbook "Operating System Concepts" by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne offers a thorough overview of operating systems. It covers subjects including networking, file systems, memory management, and process management. A notable book that covers the UNIX operating system's design ideas and inner workings is "The Design of the UNIX Operating System" by Maurice J. Bach. The authors Mark Russinovich, David Solomon, and Alex Ionescu's book "Windows Internals" offers a thorough analysis of the inner workings of the Windows operating system.It addresses issues including file systems, memory management, and processes.

Linux Kernel: Robert Love's book "Linux Kernel Development" is a thorough introduction to the development of the Linux kernel. Process management, memory management, file systems,and device drivers are some of the subjects it covers. Daniel P. Bovet and Marco Cesati's book "Understanding the Linux Kernel" offers a thorough description of the Linux kernel's internal operations, covering process management, memory management, and device drivers. The "Linux Device Drivers" book by Greg Kroah-Hartman, Alessandro Rubini, and Jonathan Corbet is a thorough instruction manual for creating device drivers for the Linux kernel. It covers subjects including USB devices, network devices, and character devices. Michael Kerrisk's book "The Linux Programming Interface" offers a thorough introduction to Linux system programming, including process management, file systems, and network programming. Overall, there is a significant body of literature on operating systems and the Linux kernel that ranges from scholarly research articles to how-to manuals and tutorials. It is simple for everyone to learn and grasp these complex topics thanks to the resources' broad coverage ofsubjects and ability-level catered content.

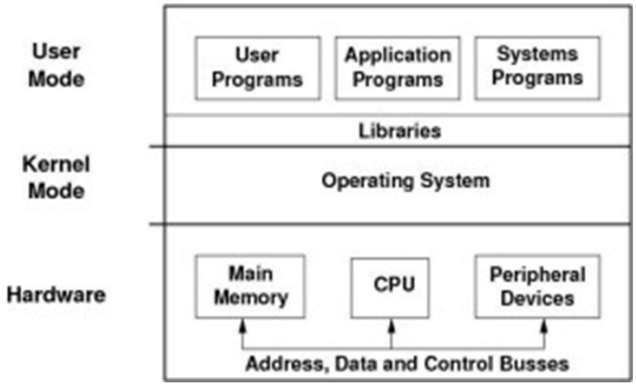


Designing an Operating system from scratch: When designing an operating system, there are several key design principles that must be carefully considered. These principles include the operating system's architecture, kernel design, and system calls. The architecture of an operating system refers to the way in which the system is structured and how it interacts with the underlying hardware. It is important for the architecture to be modular, extensible, and flexible in order to accommodate the addition of new hardware andsoftware components without requiring major changes to the underlying system. The kernel design of an operating system refers to the basic components of the system that manage system resources such as memory, CPU time, and input/output devices. The kernel must be designed to be efficient, reliable, and secure, and should be able to manage system resources in a way that minimizes contention and maximizes throughput. Security should alsobe a key consideration in the kernel design, with mechanisms in place to protect the system from malicious attacks. System calls are the mechanism by which applications interact with the operating system. Itis important for the set of system calls to be comprehensive, consistent, and easy to use, allowing developers to perform a wide range of operations such as memory management and process scheduling. Consistency across different parts of the operating system is also important so that developers can easily learn and use them. Other design principles that are important when building an operating system include performance, reliability, and maintainability. Performance is important in order to provide users with a responsive and smooth user experience. Reliability is also crucial, as minimizing the risk of system crashes and data loss is essential. Finally, maintainability is important for ensuring that the system can be easily documented and developed. Overall, designing an operating system requires careful consideration of several key design principles. These include the architecture, kernel design, and system calls, as well as performance, reliability, and maintainability. By taking these principles into account, it is possible to build an operating system that is efficient, reliable, and easy to use, meeting thedemands of modern computing.

Building an operating system using a Linux kernel: Custom operating systems may not be necessary for minor performance boosts, but they are still useful for specific types of work. For example, developers may want to create an operating system with their own specifications, such as a custom user interface (UI) design. To build such an operating system, one can start by choosing from different flavors or use third-party tools or virtual machine frameworks to load the Linux kernel. Then, developers can choose which applications to install by default, such as LibreOffice or VSCodium. Everything can be customized, including the background and tiling. Once the OS is installed,it can be exported as an ISO file. Advanced users and developers can use the Linux API to design the UI elements of the OS,but it is recommended to leave everything as it is to avoid crashing the OS. Building an operating system using a Linux kernel image has become a popular choice for developers due to its faster development times, access to a large community of open-sourcedevelopers, and pre-built software components. However, there are also limitations and practical considerations to keep in mind. One advantage of using a Linux kernel image is that it provides a solid foundation for the operating system. The Linux kernel has been extensively tested and optimized over the past two decades to ensure high performance, reliability, and security. This means that developers can focus on building higher-level components of the OS, such as the user interface and system services, without worrying about low-level details of hardware interaction.

Another advantage of using a Linux kernel image is access to a large community of open source developers. The Linux community is one of the largest and most active in the world, with thousands of developers contributing to the kernel and associated software components. This can save developers time and effort by allowing them to build on the work of others and create their own operating systems. However, there are also limitations to using Linux kernel images. One limitation is that thekernel may not support all hardware configurations. Although the Linux kernel supports a wide range of hardware, some devices may not be supported or may require custom

device drivers to be written. Another limitation is that using a Linux kernel image can limit the ability to differentiate the operating system from other Linux-based systems. This can make it more difficult to create a unique brand and identity for the operating system and limit the ability to provide functionality not found in other Linux-based systems. Finally, there are practical issues to consider when building an operating system based on an existing kernel. This includes choosing the right distribution on which to base the operating system, understanding licensing requirements for the kernel and associated software components, and ensuring that the operating system is compatible with the target hardware configuration.

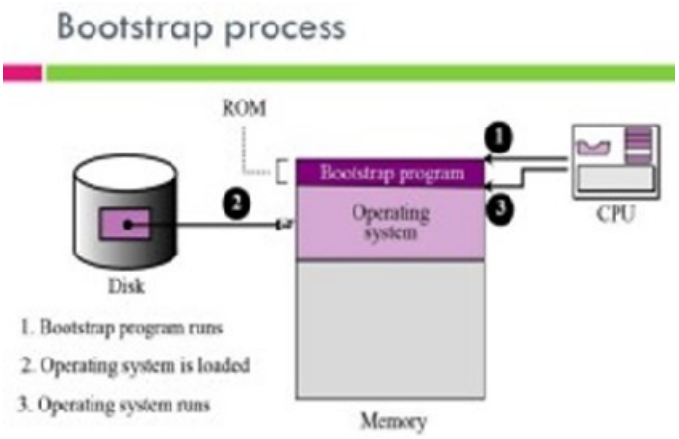


System bootstrapping and initialization

System startup and initialization refers to the crucial process by which an operating system is booted up and its components are initialized. This process is essential for the proper functioning of the operating system as it sets up the environment for running applications and system services. The boot process commences when the computer is turned on or restarted. The first step in this process is the initialization of the hardware and performing a power-on self-test (POST) by the BIOS (Basic Input/Output System) to ensure that all components are functioning properly. Once this is done, the BIOS then loads the boot loader, which is responsible for loading the operating system kernel into memory.

When the kernel is loaded into memory, the initialization process begins. The first step is to configure the memory management of the system. This includes allocating memory for the kernel and other system components, creating page tables, and other data structures needed to manage memory access. Next, the kernel initiates the process of hardware device initialization, such as the keyboard,mouse, and monitor. This involves detecting and configuring each device, as well as configuring device drivers to enable communication between the device and the operating system. Once the hardware devices are initialized, the kernel initializes the system's file system and network interface. This involves mounting file systems and configuring network settings such as IP addresses and network interfaces. Finally, the kernel starts system services and applications, which provide the user interface and other functionality. This includes starting the GUI, starting system daemons, and starting any user applications configured to start automatically.

Overall, system startup and initialization is a complex process that involves many steps, including hardware initialization, memory management, device driver configuration, and application launch. Developers and system administrators need to have a good understanding of this process to diagnose and fix boot problems, optimize system performance, and customize the boot process to their specific needs. It is worth noting that there are potential risks associated with the system startup process. BIOS corruption, for example, can occur, leading to the destruction of hardware components if not reviewed and tested properly. Therefore, it is crucial to ensure that the system is thoroughly tested and configured before the startup process is initiated to minimize the risk of such problems occurring.



Memory management and virtualization: The proper management of memory and virtualization are fundamental aspects of an operating system, as they enable applications to efficiently access resources and safeguard the system against resource competition and security breaches. At a fundamental level, memory management involves allocating and releasing memory as required by applications. This is typically done by using both physical and virtual memory, which allows the operating system to allocate more memory than is physically available, temporarily storing data on disk. Virtualization, on the other hand, entails creating multiple virtual instances of an operating system, each with its own resources and applications. This allows multiple applications to run concurrently on the same hardware without causing interference with each other or the underlying operating system. Memory management and virtualization are closely related as they both involve the allocation. and management of application resources. For instance, in a virtualized environment, a hypervisor distributes memory and other resources to each virtual machine, while an operating system within each virtual machine manages the resources allocated to it.

Preventing resource competition and security breaches is one of the significant challenges in memory management and virtualization. If several applications try to access the same memory space or hardware resource simultaneously, it may result in crashes or other issues. Similarly, if a malicious application gains access to system resources, it can compromise the security of the entire system. To address these challenges, modern operating systems employ various techniques, including process isolation, memory protection, and sandboxing. These technologies are designed to prevent applications from interfering with each other or the underlying system, while limiting the impact of security breaches and other issues. In conclusion, memory management and virtualization are critical functions of an operating system as they enable applications to access resources effectively while ensuring that the system is protected from resource conflicts and security vulnerabilities. Modern operating systems use a range of techniques, such as process isolation, memory protection, and sandboxing, to ensure that applications run reliably and securely in diverse environments.

Process management and scheduling: Process management and scheduling are fundamental functions of an operating system because they control how system resources are assigned to different applications and services. These functions involve creating, executing, and terminating processes, as well as prioritizing and scheduling those processes to keep the system running efficiently. At a basic level, a process is an instance of an application or system service that is currently running. The operating system manages each process by allocating resources like memory, CPU time, and I/O devices to each process, and ensuring that they have the resources they need to operate efficiently. This includes creating new processes as needed, managing the state of each process (e.g. running, waiting, or paused), and stopping processes when they are no longer required. Scheduling algorithms are used to determine process priority and manage process execution. Prioritizing processes can be based on various factors, such as the importance of the process,

the amount of resources required, and the execution time. The operating system then uses these priorities to determine which processes should be run first and how much CPU time should be allocated to each process. One of the biggest challenges in process management and scheduling is managing the competing demands of different processes and users. For instance, if one process is using alot of CPU time, it may cause other processes to run slowly or become unresponsive. To address this challenge, modern operating systems use various techniques such as time sharing, multitasking, and preemptive scheduling. Time-sharing involves dividing the CPUtime into fixed intervals and allocating a portion of CPU time to each process during each interval. Multitasking allows multiple processes to run simultaneously by dividing the CPU time between them. Preemptive scheduling involves interrupting a process that is using toomuch CPU time, and allocating the CPU time to another process with higher priority. Process management and scheduling are critical operating system functions because they determine how system resources are assigned to various applications and services. To achievethis, modern operating systems use various techniques such as process creation and termination, scheduling algorithms, and resource allocation policies. These techniques ensure that the system runs efficiently, while providing a fair and responsive environment for users and applications.

In addition to managing processes, modern operating systems also provide mechanisms for inter-process communication (IPC). IPC allows processes to communicate with each other and share resources. For instance, a process may need to share data with another process or coordinate with another process to accomplish a task. The operating system provides various IPC mechanisms such as pipes, message queues, and shared memory to facilitate inter process communication. Another important aspect of process management is process synchronization. Process synchronization is the process of coordinating the execution of multiple processes to ensurethat they do not interfere with each other. This is necessary when multiple processes access shared resources, such as a file or a database. Synchronization techniques such as semaphores, monitors, and locks are used to ensure that only one process can access a sharedresource at a time, to prevent data corruption and inconsistency. Overall, process management and scheduling are critical functions of an operating system, as they control how system resources are assigned to various applications and services. By usingvarious techniques such as process creation and termination, scheduling algorithms, resource allocation policies, and inter-process communication and synchronization mechanisms, modern operating systems ensure that processes run efficiently and fairly, while providing a stable and responsive environment for users and applications.

File systems and I/O: File systems and input/output (I/O) operations are fundamental components of an operating system as they enable users and applications to store and retrieve data on the system. In essence, a file system is a method of organizing and storing data on a storage device, such as a hard drive or USB flash drive. The operating system manages the file system by providing a set of tools and Application Programming Interfaces (APIs) that allow applications to interact with files, create directories, and perform other file-related tasks. Moreover, I/O operations refer to the process of reading and writing data from and to a device, such as a keyboard, mouse, printer, or network. The operating system manages I/O operations using device drivers, which are software components that interact with hardwaredevices and provide applications with a standardized means of accessing those devices. One of the major challenges in managing file systems and I/O is to ensure that data is stored and retrieved in an efficient and reliable manner. Modern operating systems use a variety of techniques to achieve this, including caching frequently accessed data in memory, optimizing disk access patterns, and ensuring that data is written to disk consistently and reliably. Another challenge is to limit the amount of bandwidth each application can use to avoid resource contention and ensure that applications run smoothly. Efficient file system and I/O management are essential components of any operating system.These components play a vital role in ensuring that users and applications can store and retrieve data efficiently and reliably. To achieve this, modern operating systems use various techniques, such as caching frequently accessed data in memory, optimizing disk access patterns, and prioritizing I/O operations according to their importance. For example, file system caching is a technique used by the operating system to store frequently accessed data in memory, which allows for faster access to that data. This technique is particularly useful for applications that require quick access to data, such as video editing software or database management systems. Additionally, the operating system may use techniques such as read-ahead and write-behind to optimize disk access patterns, which involves predicting which data the application will need next and loading that data into memory before the application requests it. Another important aspect of file system and I/O management is ensuring the reliability of data storage. For instance, the operating system may use techniques such as journaling toensure that data is written to disk consistently and reliably, even in the event of power outages or system crashes. This technique involves keeping a record of all changes made to the file system, which allows the operating system to recover from failures and restore the file system to a consistent state.

Furthermore, modern operating systems prioritize I/O operations based on their importance to ensure that critical operations are completed in a timely manner. For example, an operating system may prioritize printing jobs over other I/O operations if the printer is being used for atime-sensitive task, such as printing an important document. This technique ensures that important operations are completed in a timely manner and that the system runs smoothly. In conclusion, file systems and I/O operations are essential components of any operating system as they allow users and applications to store and retrieve data on the system. To achieve efficient and reliable file system and I/O management, modern operating systems use a variety of techniques, such as caching frequently accessed data, optimizing disk access patterns, and prioritizing I/O operations according to their importance. These techniques ensure that the system runs smoothly and that users and applications can get the most out oftheir computing experience.

Security and system administration: System security and management are integral components of any operating system as they ensure the protection of the system from unauthorized access while guaranteeing the efficient use of system resources. User accounts are a fundamental aspect of security that allow users to access the system and its resources. Operating systems manage user accounts by providing tools and APIs that enable administrators to create, modify, and delete user accounts while setting permissions and access levels for each user. This ensures that users only have access to the resources thatthey need to perform their tasks and that unauthorized access is prevented. Apart from user accounts, operating systems use various security measures to safeguard thesystem from unauthorized access. Examples of such measures include firewalls, antivirus. software, and intrusion detection systems. These security measures are designed to detect andprevent malicious activities on the system, such as viruses, malware, and hacking attempts, and to alert administrators of potential security breaches.

System administration is another key aspect of operating system management, and it involves tasks such as software updates, system monitoring, and resource allocation. The operating system provides tools and APIs that enable administrators to manage system resources, such as memory, disk space, and CPU usage, and monitor system performance to detect issues such as bottlenecks or system errors. This ensures that system resources are used efficientlyand that the system is running smoothly. To ensure system security and reliability, the operating system also provides a variety of tools and APIs that enable administrators to control access to system resources and limit the operations that users can perform. For instance, administrators can set access levels and permissions for different users, restrict access to specific files or folders, and monitor user activity for potential security breaches or misuse of system resources. These security measures ensure that the system is protected against unauthorized access and that systemresources are used effectively. In

summary, security and system administration are crucial components of any operating system as they guarantee that the system is secure and that system resources are used efficiently. Operating systems provide tools and APIs that allow administrators to manage user accounts, control access to system resources, and perform system administration tasks such as software updates and maintenance. By providing a secure and robust system management infrastructure, operating systems enable users and administrators to effectively manage and protect their systems.

Summary of Findings

After going through the report and the ROL on how to create an operating software andLinux kernel from scratch the following things can be summarized Creating an Operating System from Scratch:

- Firstly Books, online courses, forums, and other resources are all accessible forcreating an operating system from scratch.
- Secondly Most publications offer detailed instructions for creating a simple operating system, covering subjects like boot loaders, memory management, and device drivers.
- Lastly A thorough understanding of computer architecture and operating system ideas is necessary for the time-consuming and difficult process of creating an operatingsystem from scratch.
- Creating the Linux Kernel
- A Linux kernel can be created using a variety of resources, including books, onlineguides, and the kernel source code itself.
- Many publications offer thorough explanations of the core operations of the Linux kernel, including process management, memory management, and device drivers.
- It takes specialized expertise and abilities to create device drivers for the Linux kernel infields including character devices, network devices, and USB devices.
- It can be difficult and requires solid knowledge of computer architecture and operating system ideas to build a bespoke Linux kernel.

Basically, in order to create a operating software you require significant investments of time and effort. However, there are many books and resources to guide the developers for better and deeper understanding of operating system concept and computer architecture.

Limitations

It can be a useful exercise to contrast the creation of an operating system from scratch with the use of an established operating system like Linux, but there are a few drawbacks to consider. The time and money needed to build an operating system from scratch are a serious limitation. This lengthy, highly specialized procedure necessitates considerable expenditures in knowledge and resources. It might not be fair or accurate to compare such a system to a well-known operating system like Linux, which has been created and improved over many years. Lack of standardization in operating system development and deployment is another drawback. Direct comparisons may be challenging due to the distinctive features and design decisions that each operating system may have. It may be difficult to meaningfully compare various operating systems due to this lack of standardization. Additionally, comparative studies might only cover a narrow range of operating system implementation and design considerations. This might lead to a limited or incomplete awareness of the distinctions between using an established operating system like Linux and designing an operating system from scratch. It might also be challenging to duplicate the outcomes of a project to build an operating system from scratch. This process requires significant expertise and knowledge, making it challenging to find individuals with the necessary skills and resources to replicate the project in a meaningful way. Finally, the opinions and prejudices of the people performing the study could have an impact on comparison studies. This might lead to a distorted or insufficient knowledge of the distinctions between using an established operating system like Linux and designing an operating system from scratch. As a result, while comparing the development of an operating system from scratch with the use of an established system like Linux can be useful, these studies should be handled cautiously due to the constraints mentioned above. To guarantee that any conclusions drawn from these investigations are precise and significant, it is crucial to be aware of these limitations.

CONCLUSION

Creating an operating system from scratch requires a team of developers and is not advised due to the existence of numerous operating systems that offer excellent performance. Some of these systems are lightweight, while others are optimized for memory and CPU usage. However, if you are building a quantum computer, the principles of operating systemdevelopment would change as quantum computers can handle billions of processes concurrently. If you aim to develop a better operating system, you could consider using modern programming languages like Rust. Rust is memory-safe and offers the ability to develop acompletely new and customized operating system. While it may be tempting to build an operating system from scratch, it is important to consider the effort, resources, and expertiserequired to do so. It is often more practical to work with existing operating systems and modify them to suit your needs. With the increasing complexity of modern operating systems, it is essential to have a strong understanding of system design and programming to develop an operating system that is stable, efficient, and secure. Utilizing modern programming languages like Rust can help to achieve these goals and provide a safer and more efficient operating system. In summary, while creating a new operating system from scratch is possible, it requires significant resources and expertise, and it may be more practical to work with existing operating systems and modify them to meet your needs.

REFERENCES

Aroca, R. V., Tavares, D. M., & Caurin, G. (2007, September). Scara robot controller using real time linux. In 2007 IEEE/ASME international conference on advanced intelligent mechatronics (pp. 1-6). IEEE.

Barabanov, M. (1997). A linux-based real-time operating system.

Baumann, A., Barham, P., Dagand, P. E., Harris, T., Isaacs, R., Peter, S., ... & Singhania, A.(2009, October). The multikernel: a new OS architecture for scalable multicore systems. In

Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., ... & Yassour, B. A. (2010, October). The Turtles Project: Design and Implementation of Nested Virtualization. In Osdi (Vol. 10, pp. 423-436).

Black, M. D. (2009, March). Build an operating system from scratch: a project for an introductory operating systems course. In Proceedings of the 40th ACM technical symposiumon Computer science education (pp. 448-452)

Boyd-Wickizer, S., Chen, H., Chen, R., Mao, Y., Kaashoek, M. F., Morris, R. T., ... & Zhang, Z. (2008, December). Corey: An Operating System for Many Cores. In OSDI (Vol. 8, pp.43-57).

Chen, H., Chen, R., Zhang, F., Zang, B., & Yew, P. C. (2006, June). Live updating operating systems using virtualization. In Proceedings of the 2nd international conference on Virtual execution environments (pp. 35-44).

Clock Synchronization for Measurement, Control and Communication (pp. 116-121). IEEE. Eder, M. (2016). Hypervisor-vs. container-based virtualization. Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), 1.

Cochran, R., & Marinescu, C. (2010, September). Design and implementation of a PTP clockinfrastructure for the Linux kernel. In 2010 IEEE International Symposium on Precision

Gerofi, B., Santogidis, A., Martinet, D., & Ishikawa, Y. (2018, June). Picodriver: Fast-path device drivers for multi-kernel operating systems. In Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (pp. 2-13).

Gerofi, B., Takagi, M., Hori, A., Nakamura, G., Shirasawa, T., & Ishikawa, Y. (2016, May).

Han, S., & Jin, H. W. (2014). Resource partitioning for Integrated Modular Avionics: comparative study of implementation alternatives. Software: Practice and Experience, 44(12), 1441-1466.

Herder, J. N., Bos, H., Gras, B., Homburg, P., & Tanenbaum, A. S. (2006, October). Construction of a highly dependable operating system. In 2006 Sixth European Dependable Computing Conference (pp. 3-12). IEEE.

Jabeen, Q., Khan, F., Hayat, M. N., Khan, H., Jan, S. R., & Ullah, F. (2016). A survey: Embedded systems supporting by different operating systems. arXiv preprintarXiv:1610.07899.

Khomh, F., Yuan, H., & Zou, Y. (2012, September). Adapting Linux for mobile platforms:An empirical study of Android. In 2012 28th IEEE international conference on software maintenance (ICSM) (pp. 629-632). IEEE.

Lange, Matthias, Steffen Liebergeld, Adam Lackorzynski, Alexander Warg, and Michael Peter. "L4Android: a generic operating system framework for secure smartphones." In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobiledevices, pp. 39-50. 2011.

Lankes, S., Pickartz, S., & Breitbart, J. (2016, June). HermitCore: a unikernel for extreme scale computing. In Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers (pp. 1-8).

Lee, C. T., Lin, J. M., Hong, Z. W., & Lee, W. T. (2004). An application-oriented Linux kernel customization for embedded systems. J. Inf. Sci. Eng., 20(6), 1093-1107.

Maheshwari, S., Deochake, S., De, R., & Grover, A. (2018). Comparative study of virtualmachines and containers for DevOps developers. arXiv preprint arXiv:1808.08192.

Min, C., Kashyap, S., Lee, B., Song, C., & Kim, T. (2015, October). Cross-checking semantic correctness: The case of finding file system bugs. In Proceedings of the 25th Symposium on Operating Systems Principles (pp. 361-377).

Min, C., Kashyap, S., Lee, B., Song, C., & Kim, T. (2015, October). Cross-checking semantic correctness: The case of finding file system bugs. In Proceedings of the 25th Symposium on Operating Systems Principles (pp. 361-377). network stack. Proceedings of netdev.

On the scalability, performance isolation and device driver transparency of the IHK/McKernel hybrid lightweight kernel. In 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 1041-1050). IEEE.

Patel, A., Daftedar, M., Shalan, M., & El-Kharashi, M. W. (2015, March). Embedded hypervisor xvisor: A comparative analysis. In 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (pp. 682-691). IEEE.

Petroni Jr, N. L., & Hicks, M. (2007, October). Automated detection of persistent kernel control-flow attacks. In Proceedings of the 14th ACM conference on Computer and communications security (pp. 103-115).

Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (pp.29-44).

Purdila, O., Grijincu, L. A., & Tapus, N. (2010, June). LKL: The Linux kernel library. In 9th RoEduNet IEEE International Conference (pp. 328-333). IEEE. Rusling, D. A. (1999). The linux kernel.

Sabri, C., Kriaa, L., & Azzouz, S. L. (2017, October). Comparison of IoT constrained devices operating systems: A survey. In 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA) (pp. 369-375). IEEE.

Sally, G. (2010). Creating a linux distribution from scratch. In Pro Linux Embedded Systems(pp. 107-141). Berkeley, CA: Apress.

Sapuntzakis, C. P., Chandra, R., Pfaff, B., Chow, J., Lam, M. S., & Rosenblum, M. (2002). Optimizing the migration of virtual computers. ACM SIGOPS Operating Systems Review, 36(SI), 377-390.

Sharma, P., Chaufournier, L., Shenoy, P., & Tay, Y. C. (2016, November). Containers and virtual machines at scale: A comparative study. In Proceedings of the 17th international middleware conference (pp. 1-13).

****