Comparing uses of Exokernel, Microkernel, and Monolithic kernel

Sanjana Chinnola

Southern New Hampshire University

IT-600 Operation System

Dr. Shauna Beaudin

December 12, 2022

Introduction

As part of the project paper discussion, we put together information about various kernel implementations, environments, and future scope. In the initial section, we described monolithic kernel implementation, architecture, advantages, and disadvantages of the kernel. To overcome some of the disadvantages of a monolithic kernel, a microkernel is the best working solution we have out in the market. As part of the microkernel, we explained the kernel implementation, architecture, and advantages over the monolithic kernel and some real-time disadvantages with examples. Then we went on to the exokernel part and discussed the current working model along with architecture, implementation, advantages, and disadvantages. We did provide a detailed description of how these kernels evolve with respect to various operating systems like Mac OS X, Linux, and Windows.

We also provided a futuristic view of how kernels are evolving from present to future alongside a lot of research happening in the para kernel space. There are also some big tech giants working on building futuristic kernels like zircon (used in Google Fuchsia operating system) and Harmony OS is another operating system to replace the existing operating systems by providing better user experience and cross-platform usage. All this information has been detailed and discussed in the project proper by providing an overall view of the past, present, and future of kernels and operating systems. All the necessary references and links are included at the end of the paper.

Background Research

Kernel

In most computer operating systems, the kernel is its principal component. It is the bridge between the applications and the computer hardware. It is also the mechanism that allows the computer to manage multiple users and multiple tasks simultaneously. Some of the kernel types are the monolithic kernel, microkernel, and exokernel. The kernel manages all system resources of the computer. This includes long-term storage, central processing unit (CPU), short-term memory, and input and output devices (Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J., 2021).

When an application needs one of these resources, the kernel makes the resource available and completes the request. This resource utilization allows operating systems to be both multiuser and multitasking. The operating system does not actually perform more than one task at a time. Instead, the kernel swaps tasks at such high speed that the computer appears to be performing many tasks (Tanenbaum, A. S & Bos, H., 2015). The kernel is also responsible for making sure that the resources used by a user or process do not violate another user's or process's request.

Types of kernels:

There are three types of kernels which are Microkernel, Monolithic kernel and Exokernel.

Microkernel

The microkernel is also known as the µ-kernel. It will reduce the kernel and operating System Size. In this microkernel, the user and kernel services are separated and implemented in different address spaces. Because kernel and user services are separate and hence the OS is unaffected by the failure of any one of the user services (Tanenbaum, A. S & Bos, H., 2015).

Only a small part of the application runs on the kernel whereas all the application runs on the user space. In Microkernel all unnecessary programs kept in the user memory space and necessary system programs are kept in the kernel memory space. The communication between all the components provided by the message passing through IPC (Inter-Process Communication). In microkernel, it is easy to add new features. Along with that, it is portable, safe, and trustworthy.

There are two generations of Microkernels available right now. The first generation is slower than the monolithic kernel. Due to performance problems with process communication, many system services, like device drivers and communication stacks, made their way back into kernel space. In the second generation, the microkernel occupies the entire first-level cache of the processor because it is so small. As a result, I/O performance is strong.

Microkernel Architecture

In the below figure, we can observe Microkernel Architecture. Microkernel Architecture is presented as a solution to the problems with the monolithic kernel. The microkernel is only responsible for providing essential services like Inter-process communication, address space management, virtual memory management, and I/O. It is the only application that runs in kernel mode, which is a level of privilege. All other services such as device drives and files performed in the user mode (Mutia, R. I., 2014). Because all the services are divided into the user mode it is easy to manage the code by the microkernel. Since less code is running in kernel mode, stability and security improved (Roch, B., 2004). Additionally, the micro-kernel provides a simple inter-process communication facility (IPC), which enables system servers to connect with one another and exchange data on their own whether they function in a multiprocessor, multi-computer, or network configuration (Gien, M., 1990).

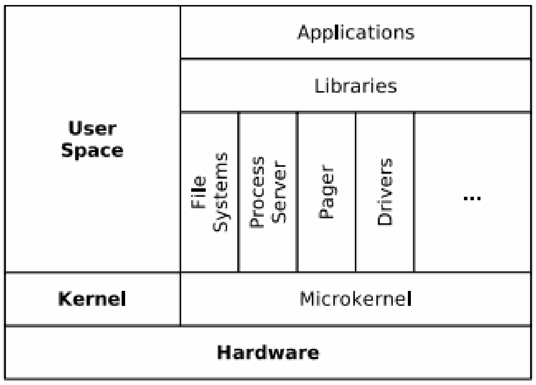

Figure 1 Microkernel-based operating system (Roch, B., 2004)

Microkernel Example

The Mach microkernel manages memory and threads, supports many processors, and facilitates inter-process communication and I/O (Golub, D. B., Julin, D. P., Rashid, R. F., Draves, R. P., Dean, R. W., Forin, A., ... & Bohman, D., 1992). The L4 microkernel provides an easy and effective framework for building operating system software, from single-purpose embedded devices to multi-processor servers. QNIX is the most widely used real-time application of the microkernel. The stability and predictability of an application are important for real-time applications. Examples of real-time applications include embedded devices like microwaves, dishwashers, automobile safety systems, cell phones, etc (Roch, B., 2004). The Minix microkernel includes numerous subsystems, including drivers, the filesystem, and the network stack, in the user space (Tanenbaum, A. S & Bos, H., 2015).

Advantages of Microkernel

The application drivers and window managers are in user space, where they may be rapidly replaced with code written in languages other than those used by the kernel without affecting the kernel (Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J., 2021). Microkernels are flexible and easy to extend due to the ability to change out servers or problem-solving method. Small components are less complicated and more controllable, which makes them simpler to maintain. Easily add new features without recompile (Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J., 2021). The distribution of responsibilities is additionally beneficial for the security and lifespan of the operating system.

Disadvantage of Microkernel

IPC communication results in more context shifts than a monolithic kernel and more overhead than a function call. A microkernel system's performance could be uneven and cause various issues like context switches causing a significant latency (Ulmanu, C., 2019).

Monolithic Kernel

In monolithic kernels, the entire operating system kernel works in the same address space and in the most authoritative way (Roch, B., 2004). They support application programs with a very powerful and rich interface on the hardware they work on. Application programs perform their operations on hardware through system calls through the kernel. These systems, which combine all pieces of code into a single executable file, are almost completely disorganized when viewed at higher levels (Roch, B., 2004).

Monolithic Kernel Architecture

In the below figure, we can observe Monolithic kernel Architecture. It is a kernel architecture in which all operating system tasks are executed in a single core space to improve system performance. Monolithic kernels use a method of control management where all operating system services are run in the same address space called kernel space. Some monolithic kernels can load and unload executable modules. This allows the capabilities of the operating system to continue with the minimum amount of code running in the kernel space at any given time. If an error occurs in any structure in the monolithic kernel, the whole system is affected by this error (Tanenbaum, A. S., Herder, J. N., & Bos, H., 2006).

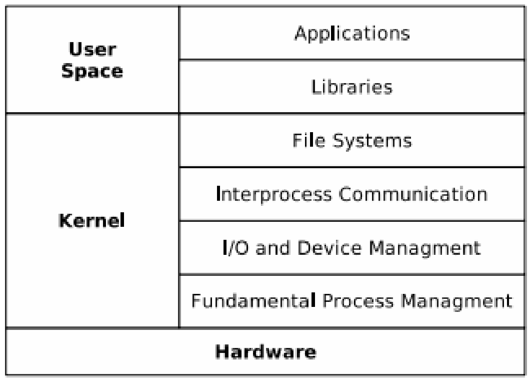

Figure 2 Monolithic kernel-based operating system (Roch, B., 2004)

Monolithic Kernel Example

In this kernel type, all desired functionality is available in the kernel. These are functions such as timing operations, device drivers, and memory management. All these functions are performed in the memory space of the kernel (Roch, B., 2004). This type of kernel is suitable for large server types that provide system services (web servers, database servers, file service servers, etc.) and can provide the largest data transfer. The most popular examples are Linux, BSD, Solaris, AIX, and DOS (Mutia, R. I., 2014).

Advantages of Monolithic Kernel

We can talk about multitasking and more efficient hardware access in Monolithic Kernel. There is no waiting during the process when it is necessary to receive information from a program or perform another operation. In such cases, there is direct realization or access (Roch, B., 2004). This has positive results when it comes to performance (Roch, B., 2004). In the Monolithic Kernel, many processes run in administrator mode and as if they are a single process, so when an error occurs, the entire system is affected. It is easier for processes to communicate with each other and migrate. Transactions run faster in this core because a transaction doesn't have to wait in the queue for it to happen.

Disadvantage of Monolithic Kernel

A monolithic Kernel does not only manage components such as processors and memory like Microkernel (Roch, B., 2004). Due to the redundancy of written code, basic production is becoming more and more complex. Also, every time we make changes to the kernel, we must recompile the entire kernel. It needs high memory. A problem in any subsystem can affect the entire system (Tanenbaum, A. S., Herder, J. N., & Bos, H., 2006). It requires a large space. Everything works in admin mode. In addition to these, issues such as device drivers and file system management are also within the scope of the management of the monolithic kernel. A monolithic Kernel consists of a single file and all its requirements are in this file. Therefore, it is not modular. This eliminates the disadvantage of processing orders in the transactions to be made.

Exokernel

Exokernel is a type of operating system developed at the Massachusetts Institute of Technology, with its main goal being to provide application-level management of hardware resources. Due to their limited operability, exokernels are typically small. Exokernel is a type of kernel mode which is run at the bottom mode. Its job also involves allocating resources to virtual machines and making sure that no one is trying to make use of another’s resources. Each virtual machine can run its own operating systems, only that each one is restricted to the resources it has asked for and approved for allocation (Tanenbaum, A. S & Bos, H., 2015).

Exokernel Architecture

The below figure talks about the exokernel architecture. The system runs two applications, thus an unmodified UNIX application against the ExOS libOS against a specialized exokernel application using its own TCP and file systems libraries. For the exokernel thus the specialized application, the kernel interface is as close to the hardware as possible whereas UNIX supplies abstractions and doesn't have to deal with the exokernel.

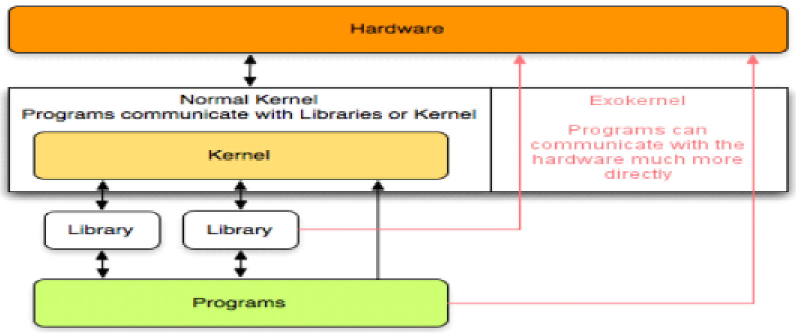
Hardware
Normal Kernel
Programs communicate with Libraries or Kernel
Kernel
Exokernel
Programs can communicate with the hardware much more directly
Library
Library
Programs

Figure 3 Exokernel Architecture, (Ulmanu, C., 2019)

To enable application-specific customization, the exokernel architecture is intended to decouple resource protection from management. The exokernel operating system architecture basically solves the issue of application-level management of physical resources. It does this by securing exports of all hardware resources through a lower-level interface to an untrusted library or virtual operating system. This separation of resource protection from management enables application-specific customization of the default operating system abstractions through the extension, specialization, or even replacement of libraries.

As far as abstractions are concerned, exokernel operating systems try to eliminate the notion that operating systems must provide abstractions upon which applications are built. The idea is to provide them with few abstractions as possible and allow them the liberty to use abstractions when needed Because of the way the exokernel architecture was designed, the kernel relocates hardware abstraction into what are known as the library operating systems—untrusted libraries. The goal is to ensure that there is no force abstraction, making the exokernel different from the micro- and monolithic kernels.

Exokernel Example

Exokernel usually separates the allocation of resources from management. Although each program controls its own disk block and cache, the exokernel enables safe cache page sharing between apps. So, therefore, we can say the exokernel protects disk blocks and pages but doesn't manage them at all (Frans Kaashoek, M., Engler, D. R., Ganger, G. R., Briceno, H. M., Hunt, R., Mazières, D., ... & Mackenzie, K., 1997).

An example is the research paper theme "Application performance and flexibility on exokernels system". This paper evaluates the exokernel architecture by measuring end-to-end applications on Xok, an exokernel for Intel x86-based computers, and makes a comparison of performance with two UNIX systems (Free BSD and OpenBSD). The results prove that Xok performs significantly better or comparably when compared to BSD UNIXes (Kaashoek, M. F., Engler, D. R., Ganger, G. R., Briceno, H. M., Hunt, R., Mazieres, D., ... & Mackenzie, K., 1994).

Advantages of Exokernel

It does well to save a lay of mapping, without the exokernel the virtual system must maintain tables to remap disk addresses however with the exokernel, remapping is not needed. Also, the exokernel can separate multiprogramming from the user operating system code but with less overhead as the exokernel protects divided resources from each virtual machine (Tanenbaum, A. S & Bos, H., 2015). The separation of resource protection by the exokernel allows for application-specific customization of traditional operating systems through extending, specializing, or even replacing non-function libraries or virtual systems (Engler , D. R & Kasshoek, F. M & O'Toole Jr, J., 1995).

Exokernel also shares resources amongst untrusted applications as well as trusted applications, unlike traditional operating systems who don't share resources with untrusted applications. Improved performance of applications. Easier development and testing of new operating systems. Each user-space application is allowed to apply its own optimized memory management.

Disadvantage of Exokernel

Different systems of exokernel might have different interfaces, some complex, others not so complex. Design tends to be complicated due to various levels of abstraction built on top of exokernels.

Difference Between Kernel Types

Table 1 Difference Between Kernel Types

Basic Comparison	Micro Kernel	Monolithic Kernel	Exokernel Kernel
Size	Smaller	Larger than micro	Small Sized
Extendible	Easier	Harder	Easy to extend
Security	More secure	Less secure	Less secure
Speed	Slower	Faster	Faster
Debug	Simple	Harder	Simple
Maintainability	Simple	Extra time and resources needed	Simple

Execution	Most important processes work in kernel space. Others in user space.	Executed under the kernel space in privileged mode.	Easy to execute, mainly must dish out resources through the operating systems

Kernels are important aspects of the operating system, it does the most basic assignment of the operating system, thus assigning hardware resources to software applications so that they can fulfill the task the developer wants them to. The differences between the kernel include the fact that the monolithic kernel executes its operating system instructions in the same address space to improve performances (Kapoor, C., Kathuria, K., & Adlakha, A, 2014). The problem with such a system is that if there is a bug in the kernel, it would affect the whole system thus the address space. This can crash the whole kernel as a whole and a reboot would be needed. Ways to tackle the issue is to write clean code for the monolithic kernel to prevent a crash. Another way is to keep errors to the minimum as possible; however, the issue with that is that 6 million lines of code in the monolithic Linux kernel makes it difficult to avoid many possible bugs.

When it comes to the microkernel the system tried to eliminate the number of bugs or errors. This is done by moving away parts of the kernel from kernel space to user space. Both parts in either user space or kernel space cannot communicate with each other unless permitted to do so. For example, the networking subsystem which crashes the monolithic kernel would have less of an effect on the microkernel subsystem. In the microkernel, because there is not only one address space, the subsystem in the microkernel may crash but other systems would work perfectly. As a matter of fact, crashed servers will automatically reload; nevertheless, there are problems associated with the microkernel system as well.

The first problem with this process is complex. In the microkernel design, only a small subset of the tasks is under the kernel space of the monolithic kernel; all other stuff resides in the user space. Most of the time the part residing in the kernel space is responsible for communication between the servers running in user space thus the ‘inter-process communication’. Such servers provide functionality such as sound, display, disk access, networking, etc (Tanenbaum, A. S & Bos, H., 2015). This process adds much more complexity to the overall process. An analogy is that a piece of pork (the monolithic kernel), chop into small parts (servers) and put each of those parts into plastic bags (the isolation), and then link the individual bags to one another with strings (the IPC). The total weight of everything concerned here would be how the microkernel would look like.

The main issue with the complexity is its relation to performance. On the global level, the microkernel is more complex than the monolithic kernel; however, on the local level, it seems not to matter. In simple terms, the communication between the servers of a microkernel takes time, whereas, in the monolithic design, communication is not required as all servers are tied to one address system or piece of computer code instead of different servers. If a performance test is conducted at this moment considering there are similar features the monolithic kernel will comparably outperform the microkernel.

The exokernel which is the third kernel would be discussed. It is mainly responsible for application-level management of hardware resources. They are small because of their limited operability and are run at the bottom mode. In the cited research paper theme “Application performance and flexibility on exokernels system, it was established that using the application on Xok (an exokernel for Intel x86 computers) in comparison to two UNIX systems; the exokernel performance was comparably better than the UNIX. Application.

In comparison to microkernels, exokernels are much faster because of the simple task system. Just like the monolithic kernels both exokernel and monolithic kernels differ due to the level of complexity in both systems. When comparing global performance of the exokernel to that of the microkernel, the microkernel is more complex as there are many steps involved in the microkernel. The exokernel decentralization of resources management allows for performance of the individual application to be improved (assuming the exokernel is Xok) (Kaashoek, M. F., Engler, D. R., Ganger, G. R., Briceno, H. M., Hunt, R., Mazieres, D., ... & Mackenzie, K., 1994).

Security and Stability

The kernel is an important thing to protect system processes from changes made by users and other processes. As a result of multitasking, problems started to appear in memory (Chen, J. B., & Bershad, B. N., 1993). These difficulties include things like memory protection, race situations, and system security itself. If a process crashes, it has an impact on kernel and system performance. This is a simple task when discussing processes running in user space. If a process fails within the kernel, it can be assumed that due to the "hardware" of system processes and subsequent dependency on the monolithic approach, other processes will crash as well, resulting in a system-wide halt. One solution to these problems is to keep system processes out of kernel space (Tanenbaum, A. S., Herder, J. N., & Bos, H., 2006).

Another problem is code size. The Windows XP kernel is more than twice as big as the Linux kernel, which has more than 2.5 million lines of code (Tanenbaum, A. S., Herder, J. N., & Bos, H., 2006). We found out that there are two to 75 faults per 1,000 lines of executable code, while another found that there are six to 16 faults per 1,000 lines of executable code. The Windows XP operating system has at least twice as many defects as the Linux kernel, assuming a conservative estimate of six bugs per 1,000 lines of code (Tanenbaum, A. S., Herder, J. N., &

Bos, H., 2006). It is simpler to guarantee the accuracy of a small code than for a large one. In this sense, this method makes it simpler to fix stability problems.

Protect the OS

Operating systems, being based on kernels, tend to be extremely vulnerable to both the sheer size of the kernels and the inherent unreliability of drivers within the operating system (Tanenbaum, A. S., Herder, J. N., & Bos, H., 2006). In fact, failures related to drivers tend to be many times more frequent than errors within lines of kernel code. To protect the operating system at the kernel and driver level, safeguards can be added at the individual level. The least intrusive method of achieving this is the Nook method, wherein each driver in an operating system is enveloped by a lightweight program that actively monitors the transactions of each driver independently and steps in to terminate erroneous processes when detected (Tanenbaum, A. S., Herder, J. N., & Bos, H., 2006). This method also allows system administrators to protect their devices' operating systems while still maintaining any legacy systems that they are not ready or willing to upgrade.

Windows OS, MAC OS, and Linux OS implementing kernels

Windows OS:

The Windows Operating System has been launched in the market in 1985. It is a powerful and complete type of software, that seems to have a nearly 90% market share compared to other operating systems with a large and powerful presence in commercial buildings, industrial facilities, and home computers. Microsoft has made major investments in marketing and finance in recent years to demonstrate that Windows integration is unquestionable and that it has everything that it requires as a platform to execute any enterprise application.

LINUX OS:

Linux is a FREE operating system. Downloads, modifications, and redistribution are all free. Linux is a relatively new operating system. It was written in 1991 and has been updated for existing use. When compared Linux is more flexible and Windows is rigid. One cannot go beyond what Microsoft has designed. Linux is not like the original UNIX source tree code; however, it uses UNIX standards to behave like it is a UNIX.

MAC OS:

Mac OS is significantly older than Windows OS. It was launched one year before its Windows main competitor, and it is the first and only successful graphical-oriented operating system. Mac OS has gone through two key design turnarounds and is now in its third stage (Adekotujo, A., Odumabo, A., Adedokun, A., & Aiyeniko, O., 2020).

How does each OS implement these kernels

The kernel manages the operations of an operating system. Mac OS X employs a microkernel architecture. User services and kernel services are implemented in different address spaces in the microkernel, both user space and kernel services are kept in the kernel address space. The microkernel performs basic operations such as memory, process scheduling mechanisms, and inter-process communication. Other functions are removed from kernel mode and run-in user mode. If a service fails, it has no effect on the operation of a microkernel. Servers communicate through IPC. Because microkernel architecture is small and isolated, it can function better. For example, if a service fails, it does not affect the kernel space but only the user space involved in that space. Modules can be replaced, reloaded, and modified without affecting the kernel. This architecture makes it easier to add or remove new services without disrupting the kernel. Mac OS X uses a microkernel architecture to keep small servers running different operations. It not only improves efficiency, but it also protects the kernel from crashes and makes it simple to replace problematic parts without touching the kernel (Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J., 2021).

Windows operating systems like Windows 95,98,2000 use monolithic kernel architecture. In a monolithic kernel, both user services and kernel services are implemented in a single address space. If a service crashes, the whole operating system collapses causing the system shut down. Due to the issues, we had with monolithic kernels, to overcome these issues Windows operating system (windows 7) uses a hybrid kernel which means it attempts to combine the features of microkernel and monolithic kernel architectures. The purpose is to utilize both the performance that monolithic kernels offer and the stability that microkernels offer. In a hybrid kernel, some services run in kernel space to reduce the performance overhead of the traditional microkernel, while still running kernel codes as servers in user space. Linux also implements similar architecture as windows by utilizing the monolithic architecture where the kernel handles all hardware and driver operations (Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J., 2021).

Exo kernel is a type of operating system developed by MIT to provide application-level management of hardware resources. To enable application-specific customisation, the exokernel architecture is intended to decouple resource protection from management. Exo kernels are typically small because of their limited operability. Exo kernels support better application control by separating security from management by providing a low-level interface. It also improves the performance of applications by making the most efficient use of resource allocations, making it easier to develop and test new operating systems (Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J., 2021).

Future relevance or importance

The kernel is the operating system's brain, as we all know. Today's technology uses kernels, earlier. Which facilitates our work. Early on, the operating system did not have a kernel, thus the user had to handle everything; however, after the kernel was added, every application procedure became simpler than it had been earlier

New or improved OS Resources in the kernels

Process:

A process decides which memory areas an application can access. A kernel's primary function is to provide program execution and to support it with tools like hardware abstraction. A kernel loads the file containing an application, creates an address space for the program, and then starts the application. A program built once the program is put into memory, and execution eventually starts by branching to a certain place within the program. (Rhoden, B., Klues, K., Zhu, D., & Brewer, E., 2011)

Memory:

The kernel has complete access to the hardware's memory. Currently, processes can use it to safely access this memory as needed. The use of virtual addressing by the kernel allows for the formation of virtual memory partitions in two different places, one designated for the kernel (kernel space) and the other for the applications (user space) (Rhoden, B., Klues, K., Zhu, D., & Brewer, E., 2011).

Kernel I/O:

The memory of the hardware is completely accessible to the kernel. Currently, processes can use it to securely access this memory as needed. The kernel can partition virtual memory into two areas using virtual addressing: one for the kernel (kernel space) and another for the applications (user space). A kernel keeps track of the available devices. To find devices, a device manager initially scans several hardware buses, including PCI and USB (USB). then looks for the suitable drivers for the attached devices. (Roch, B., 2004).

System Calls and Interrupt Handling:

The application uses a system call as an approach. A software to request a service from the operating system. System calls include operations like close, open, read, wait, and write. To use the services, the kernel provides, the necessary kernel functions must be called. Most kernels have a C library or API that enables you to call the necessary kernel functions (Tanenbaum, A. S & Bos, H., 2015).

Scheduling:

In a multitasking system, the kernel will allocate a set amount of time to each program and switch between them quickly enough for the user to think they are all running simultaneously. The kernel chooses which process will run next and how much time it will be allotted using scheduling methods. Which processes come first are decided by the mechanism (Tanenbaum, A. S & Bos, H., 2015).

New or improved version of the kernel

The new kernel, which built using FORTRAN code, offers the same dependability and performance as the old kernel. The new core also permits the simultaneous use of every refining technique now accessible, including single peak configuration tweaks and HKL file fiddling, on a single model. It is now possible for users to create extraordinarily complex and one-of-a-kind fine-tuning models by choosing a different tuning method for each multiphase model component (Degen, Sadki, Bron, Konig, & Nenert, 2014).

User interaction made simpler by the first kernel implementation techniques. The OS and application resources are separated by the kernel, which serves as a middleman for both The kernel also optimizes for the user all OS resources, including memory and I/O. Pattern analysis and data reduction are accomplished using kernel methods. (Tanenbaum, A. S., Herder, J. N., & Bos, H., 2006).

Issues with kernels

At first, the computer did not support multitasking, and many users' faces were hidden by shadows. Computers were originally a one-man show. "Bare metal," a business device. Programs for the mobile interface have total control over the memory, hardware, and remote printers. This has led to a greater need for multi-user and multi-tasking (OS). All software operations are served by the operating system (OS) packages, which effectively use the laptop's resources. The kernel is an operating system's brain. This is the initial action. It loads for the first time at the beginning of the consultation and is thereafter constantly utilized (Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J., 2021).

Most suitable environments for kernels

The kernel may be dynamically enlarged by include functions from any functional class. If the process were running in user mode, it could customize the kernel using the sys config subroutine with the required privileges. By utilizing kernel services to load, unload, and terminate dynamically loaded kernels, kernel extensions can further modify the kernel. When called by a user process in kernel mode, a kernel extension executes in the process environment.. Similarly, a kernel extension executed in an interrupt environment when invoked as part of an interrupt handler (Qian, Z., Xia, R., Sun, G., Xing, X., & Xia, K., 2022).

Kernel extension can determine which environment it called by running the getpid or thread self-kernel service. Linux is the best environment for building kernels as it has all the most advanced resources. Along with this knowledge of C and assembly language required to build a kernel from scratch. There are different Linux platforms out there that you can use to build the necessary kernels depending on the use. But to build a kernel you would need to have extensive knowledge in the respective domain. Right now, we also have a lot of evolutions in kernels with the most recent being cloud environments (Qian, Z., Xia, R., Sun, G., Xing, X., & Xia, K., 2022).

Evolution of kernels

There is a lot of process and debate going on between people from various industries about whether monolithic kernels or microkernels have a superior architecture. Despite the widespread disagreement, research is continuing at both the industry and academic levels, in tandem with new enhancements. It is currently not a matter of which kernel is superior; rather, it is a matter of code modularity. Various companies are working on various projects to develop new operating systems and kernels to support those platforms. From Google's Fuchsia project to BPF in Linux, there is a lot of evolution going on in the kernel space. Google Fuchsia is built on a new message-passing kernel called Zircon. Its code base is derived from the Little Kernel (LK) for embedded devices, as well as for low resource uses on a wide range of devices (Kuo, H. C., Chen, J., Mohan, S., & Xu, T., 2020).

Zircon is mostly written in C++, with some parts in assembly language. Zircon has a small set of user services, drivers, and libraries that are required for the system to communicate with hardware and load user requirements. Its current features include handling multiple threads, virtual memory, and process intercommunications. Zircon is heavily inspired by Unix kernels but differs significantly. Like Google Fuchsia there are other projects that are happening in parallel like Redox which is still following the microkernel approach. There is also some experimental research OS called Manticore, written in Rust, aiming to explore the world and concepts of Para kernels. When getting more into the advancement of kernels a lot of research is happening into what kind of language to use and the kind of kernels to be built to support cross-platform usage by improving performances and reducing the usage of resources (Kuo, H. C., Chen, J., Mohan, S., & Xu, T., 2020).

Evolution of Operating systems

Nowadays, technologies are changing at the rate of polyphyodonty. Many global enterprises are researching and developing future products to enhance performance and compete in the large technology market. According to research, users' use of mobile devices and the internet is increasing, so the market is expanding with new features and technologies to demonstrate how future technology will evolve. To provide a better user experience, we would need a better software platform to support all these new features. The operating system is personally responsible for all the products. There are many operating systems that are already very mature, such as Android by Google and iOS by Apple. However, the Internet of Things will be the operating system of future (IoT) devices. To support all these different streams of business is looking to build the best-operating systems to meet the demand (Jang, Baek, & Park, 2022).

Virtual reality technology would create a major disruption in the video game industry followed by healthcare and engineering. Similarly, users are trying to connect different devices seamlessly. Due to the difference in platforms the devices are not synchronized as expected. This is one of the areas where major tech giants must build an OS that would support cross-functionality between operating systems. Right now, Google is working on fuchsia (an upcoming operating system). It is an open-source and microkernel-based operating system for desktops, mobile devices, and other embedded systems. The core concept of the operating system is to build a powerful ecosystem for a better user experience. Users can synchronize desktops mobile phones, and wearable devices. It may be a replacement for Android OS. Similarly, Harmony OS is another open-source operating system that is in development since 2012. This is also based on micro-kernel. They are also working on making Harmony OS more usable for various devices like desktops, mobiles, etc (Jang, Baek, & Park, 2022).

Similarly, Apple and Microsoft are also enhancing their existing OS to provide a better user experience by reducing downtime and improving performance. Right now, a lot of research is happening with new companies trying to build better-operating systems and existing tech giants are making their products more sustainable and occupying space in the global emerging market (Jang, Baek, & Park, 2022).

Conclusion

In this paper, we discuss the kernel's detailed description. Additionally, there are other kinds of kernels, each with a different architecture, benefits and drawbacks, and comparisons between microkernel, monolithic, and exokernel. the kernel's stability and security. The kernel works in the Windows, Linux, and Mac operating systems and serves to secure the OS. The kernel also implemented in this operating system. whose task will be in the future and how important the kernel is.

We understand that authoring this paper kernel is the operating system's brain, as we all know. Early on, the operating system did not have a kernel, thus the user had to handle everything; however, after the kernel was added, every application procedure became simpler than it had been earlier. When a user process calls a kernel extension in kernel mode, the extension executes in the process environment. Similarly, a kernel extension is executed in an interrupt environment when invoked as part of an interrupt handler. Linux is the best environment for building kernels as it has all the most advanced resources. Various companies are working on various projects to develop new operating systems and kernels to support those platforms. From Google's Fuchsia project to BPF in Linux, there is a lot of evolution going on in the kernel space. Google Fuchsia is built on a new message-passing kernel called Zircon

Linux kernel underlines the world in ways that most people do not truly consider. Even game systems, except for Xbox, employ software that is like Linux, including your internet router, any Android phone, and any smart TV. The mobile phone market is heavily biased toward Android, which is built on the Linux kernel.

References

Adekotujo, A., Odumabo, A., Adedokun, A., & Aiyeniko, O. (2020). A Comparative Study of Operating Systems: Case of Windows, UNIX, Linux, Mac, Android and iOS. *Interrnational Journal of Computer Application*, 16-23. Obtenido de 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-UNI

Chen, J. B., & Bershad, B. N. (1993). The impact of operating system structure on memory system performance. *In Proceedings of the fourteenth ACM symposium on Operating systems principles, 14*, 120-133. Obtenido de https://dl.acm.org/doi/pdf/10.1145/168619.168629

Degen, T., Sadki, M., Bron, E., Konig, U., & Nenert, G. (31 de August de 2014). *The HighScore suite*. Obtenido de https://www.cambridge.org/core/services/aop-cambridge-core/content/view/C5F7C0DD3DA96EA5E0CFF1A0C9C87DF3/S0885715614000840a.pdf/the-highscore-suite.pdf

Engler , D. R & Kasshoek, F. M & O'Toole Jr, J. (1995). Exokernel: an operating system architecture for application-level resource management. *ACM SIGOPS Operating Systems Review*. Obtenido de https://dl.acm.org/doi/pdf/10.1145/224057.224076

Frans Kaashoek, M., Engler, D. R., Ganger, G. R., Briceno, H. M., Hunt, R., Mazières, D., ... & Mackenzie, K. (1997). Application performance and flexibility on exokernel systems. Obtenido de In the Proceedings of the 16th ACM Symposium on Operating Systems

Principles (SOSP 1997), Saint-Mal, France: https://dl.acm.org/doi/abs/10.1145/268998.266644

Gien, M. (1990). Micro-kernel Architecture Key to Modern Operating Systems Design. *Unix Review*, 10. Obtenido de http://www.leonard.nom.fr/publications/chorus/CS-TR-90-42.pdf

Golub, D. B., Julin, D. P., Rashid, R. F., Draves, R. P., Dean, R. W., Forin, A., ... & Bohman, D. (1992). Microkernel Operating System Architecture and Mach. *In Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, 20. Obtenido de https://courses.cs.washington.edu/courses/cse451/15wi/lectures/extra/Black92.pdf

Jang, E. T., Baek, s., & Park, K. (2022). Aging Aging analysis framework of windows-based systems through differential-analysis of system snapshots. *CMC-computers materials & continua*, 5091-5102.

Kaashoek, M. F., Engler, D. R., Ganger, G. R., Briceno, H. M., Hunt, R., Mazieres, D., ... & Mackenzie, K. (1994). Application performance and flexibility on exokernel systems. Obtenido de In Proceedings of the sixteenth ACM symposium on Operating systems principles: https://dl.acm.org/doi/pdf/10.1145/268998.266644

Kapoor, C., Kathuria, K., & Adlakha, A. (2014). Kernel Operating System. *International Journal of Research in Information Technology*. https://708e8bd3-a-bb6ad6f3-s-sites.googlegroups.com/a/ijrit.com/papers/october/V2I1019.pdf?attachauth=ANoY7cq2fGG8yWxgY2zs7pak_eXv7c_SNAVVIPPHnt7Plv8zZvTXwFWsSbiXJtb07VwXZrMhyq1sdnrxS-DerYoussqV9J-rQB9nQohR5iknvuwGS3wZK-iZqlc2xLTY6PKaG7X64vQAjLekWrretQ5U

Kuo, H. C., Chen, J., Mohan, S., & Xu, T. (2020). Set the configuration for the heart of the os: On the practicality of operating system kernel debloating. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1-27. Obtenido de https://dl.acm.org/doi/pdf/10.1145/3379469

Malallah, H., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Ageed, Z. S., Ibrahim, I. M., ... & Merceedi, K. J. (2021). A comprehensive study of kernel (issues and concepts) in different operating systems. *Asian Journal of Research in Computer Science*, 8(3), 16-31. Obtenido de https://www.researchgate.net/profile/Karwan-Jameel-2/publication/351424523_A_Comprehensive_Study_of_Kernel_Issues_and_Concepts_in_Different_Operating_Systems/links/60a4203e92851ccc66b61fda/A-Comprehensive-Study-of-Kernel-Issues-and-Concepts-in-Different-O

Mutia, R. I. (2014). Inter-Process Communication Mechanism in Monolithic Kernel and Microkernel. *Department of Electrical and Information Technology Lund University*, 7. Obtenido de https://www.eit.lth.se/fileadmin/eit/project/142/IPC_Report.pdf

Qian, Z., Xia, R., Sun, G., Xing, X., & Xia, K. (2022). A measurable refinement method of design and verification for micro-kernel operating systems in communication network. *Digital Communication and networks*.

Rhoden, B., Klues, K., Zhu, D., & Brewer, E. (2011). Improving Per-Node Efficiency in the Datacenter with New OS Abstractions. *In Proceedings of the 2nd ACM Symposium on Cloud Computing*. Obtenido de https://dl.acm.org/doi/pdf/10.1145/2038916.2038941

Roch, B. (2004). Monolithic kernel vs. Microkernel. *Tu Wein*. Obtenido de http://web.cs.wpi.edu/~cs3013/c12/Papers/Roch_Microkernels.pdf

Tanenbaum, A. S & Bos, H. (2015). *Modern Operating System.* New Jersey: Pearson.

Tanenbaum, A. S., Herder, J. N., & Bos, H. (2006). Can we make operating systems reliable and secure? *Computer*. Obtenido de https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1631939

Ulmanu, C. (2019). Types of Operating System Kernels. *In Conferinţa tehnico-ştiinţifică a studenţilor, masteranzilor şi doctoranzilor*, 4. Obtenido de https://ibn.idsi.md/sites/default/files/imag_file/597-600_0.pdf