Volume 5, Issue 6, June 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

The Impact of Architecture on the Performance of Monolithic and Microkernel Operating System

Junaid Jadoon, Faisal Bahadur
IT Department, Hazara University,
Mansehra, Pakistan

Abstract—Since long the field of scientific computing has experienced rapid changes. We have experienced a remarkable advancement of technologies, architectures, vendors, and the use of system. We also expect clock-rates to increase, caches to grow, and multiprocessors to replace Uni-Processors. This paper explores the impact of the above structural trends on operating system (monolithic &microkernel)performance. We find that the benefits of these architectural trends have great impact on majority of the important services provided by the operating system.We find that the Kernel Space also matters a lot in both Monolithic and Microkernel. The paper presents a detailed rectification of all issues relating execution time, space modification,extensibility as well as reduction in performance flaws in important kernel services.

Keywords—Kernel Space,Monolithic kernel,Microkernel, Uniprocessors, Multiprocessor,

I. INTRODUCTION

The kernel is the most important part of an operating system. Roughly, an operating system itself consists of two parts: the kernel space (privileged mode) and the user space (unprivileged mode). Without that, protection between the processes would be impossible. There are two different concepts of kernels: monolithic kernel and microkernel. The older approach is themonolithic kernel, of which Linux, MS-DOS and the early Mac OS are typical representatives.

A. Monolithic Kernel

It runs basic system services like process and memory management, interrupt handling and I/O communication, file system, etc. in kernel space (Figure 1). All basic services in kernel space havethree big discrepancies: the kernel size, lack of extensibility and the bad maintainability. Bug- fixing or the addition of new features means a recompilation of the whole kernel.As the compilation of kernel consumes a lot of time and memory.Every time someone adds a new feature or fixes a bug, it means recompilation or modification of the whole kernel space.

User Space	Applications
	Libraries
Kernel	File System
	Inter Process Communication
	I/O &Device Management
	Fundamental Process Management
Hardware	

Figure 1: Monolithic kernel based operating system

The Monolithic kernel is complex. A kernel with plenty of line of code could be hard and difficult to maintain. A large amount of code means that the operating system could not be feasible for different hardware, especially for embedded systems. Another disadvantage of the monolithic operating system is that it is not reliable; since the kernel's complexity, the possibility of a system crash could be high. A small error in the kernel canlead the whole system to crash.

Monolithic kernel is a single large processes running entirely in a single address space. It is a single static binary file. All kernel services exist and execute in kernel address space. The kernel can call functions directly. The examples of monolithic kernel based OSs are Linux

B. Microkernel

To overcome these limitations of extensibility and maintainability of monolithic kernel, the idea of microkernel's appeared at the end of the 1980's. The concept (Figure 2) was to reduce the kernel to basic process communication and I/O control, and other system services to be residing in user space in form of normal processes that are called as servers). There is an individual server for each process, i.e. aseparate server for managing memory issues, one server for process management, and another one for managing drivers, and so on. As the servers do not run in kernel space anymore, "context switches" are needed, to allow user processes to enter privileged mode and to exit again. In this way microkernel is not the junction or block of system services, but represents just several basic rules and primitives to control the communication between the processes & between a process and the hardware. As communication is not done directly, a new message system is generated, which allows independent communication and favors extensibility[1].

User Space	Applications				
	Libraries				
	File System	Process Server	Pager	Drivers	……
Kernel	Microkernel				
Hardware					

Figure 2: Microkernel based operating system

The servers communicate through IPC and call "services" from each other by sending messages. As the servers are separated it has one advantage that if one server fails it will not harm or effect the working of other server. The example of microkernel based OS are Mac OS X and Windows NT [2].

In this paper we present a detailed characterization of a modern Linux and windows operating system clearly identifying the areas that present challenges to key performance of both Operating systems.We present detailed performancedata of monolithic and microkernel operating system services.

Considering first monolithic results, our data shows that for both present and future systems the storage structure/hierarchy (disk and memory system) is the key for overall system performance. From given technology trends, we find that I/O is the first-order utility for workloads such as program development and transaction processing. Consequently, any changes in the operating system which result in more efficient use of the I/O capacity would be more prominent for performance benefits.

After I/O, it is the memory system which has the most significant performance impact on the kernel. In contrast with expectations, we find that future memory systems will not be more bottleneck than they are today. Although there will not have rapid changes in system speed as instruction-processing rates, the use of larger caches and dynamically-scheduled processors will compensate.

We find that on future machines, kernel performance will improve as fast as application program performance resulting in kernel overheads remaining relatively the same in the future

II. COMPARISON OF MONOLITHIC ANDMICROKERNEL IN DIFFERENTASPECTS

A.Memory Management

Monolithic kernels implement everything needed for memory management in kernel space. This includes allocation strategies, virtual memory management, and page replacement algorithms (Figure 3).

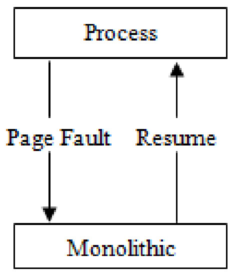

Figure 3: Memory management of monolithic kernels

First Generation Microkernel's delegate the memory management to user space, controlling just the basic access rights (Figure 4). One of the servers is responsible for managing page faults and reserving new memory. Every time a page fault occurs 5, the request has to take the way through the kernel to the pager. The pager must enter the first to privileged mode then to the get back to user mode. Then it sends the result back to the triggering process (again through the kernel). The whole procedure to handle page faults or reserve new memory pages is tedious and time consuming [5].

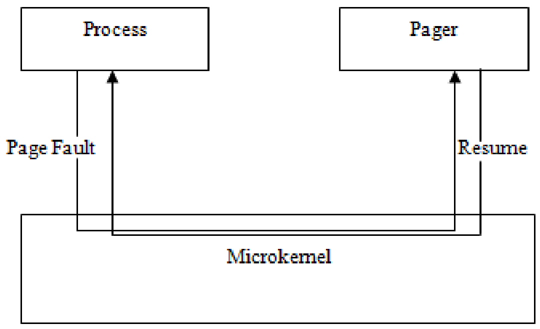

Figure 4: Memory management of 1st-generation microkernel

To rectify performance issue, second generation microkernel's had more characterized strategies of memory management, e.g. L4 (Figure 5). With L4 every process has got three memory management primitives: map, grant and flush. A process only maps memory pages to another process if he wants to share these pages. When a process grants pages to another process, itlosses access on them, and they are under the control of other processes, as long as the granting process does not flushes them. Flushing regains granted and mapped memory pages. This system now works as follows: The microkernel has the whole system memory at startup toone process, the base system process, which rests in user space. If a process needs memory, he will directly ask the base system process. Because every process can only grant/map/flush the memory pages it owned before, memory protection still exists.

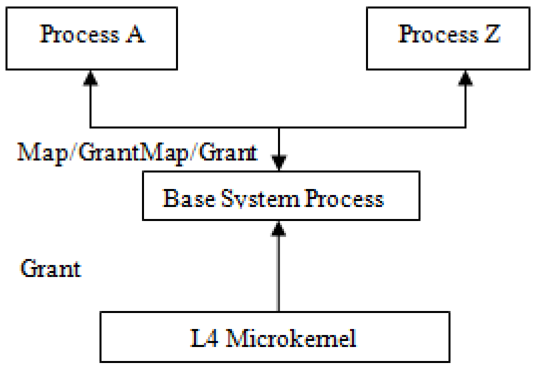

Figure 5: Memory management of the L4

B. *Security and Stability*

The protection of system processes from being iterated or modified by the user or other processes is key feature of the kernel. When multitasking and multithreading are introduced new problems arises, concerning isolation ofmemory and processes. These generic problems include issues like race conditions, memory protection and system security itself. The kernel must be able to grant that in case a process crashes, the system's performance will not be influenced.
Considering only the process that run in the user space is easy. But what happens, if a process crashes inside the kernel? Because of the "hardwiring" of system processes and the resulting dependency/discrepancies of the monolithic approach, in this situation it is assumable that other processes will also crash, resulting in whole system halt. Excluding system processes from kernel space is a way to overcome these problems. Another argument for a true microkernel is its code size. It is easier to ensure the accurateness of a small kernel, than a big one.

C. *I/O Communication*

I/O communication works through interrupts, generated by or sent to the hardware.Monolithic kernels run device drivers inside the kernel space. Kernel processes directly handle hardware interrupts. If hardware provide any feature to be changed or added, then all layers in monolithic kernel will b changed in any case.This concepts was used to achieve individualization and separation from kernel space. Onemodule represents (parts of) a driver and is (un)loadableduring runtime. That way, drivers which are not neededby the system are not loaded and memory is preserved.

The microkernel approach doesn't handle I/O communication directly. It only ensures the communication. Requests from or to the hardware are redirected as messages by the microkernel to the servers in user space. If the hardware triggers an interrupt, the microkernel sends a message to the device driver server, and has nothing more to do about it. The device driver server takes the message and sends it to the right device driver. That way it is possible to add new drivers, exchange the driver manager without exchanging drivers or even exchange the whole driver management system without changing any other part of the system shows that it is not feasible to put the drivers into kernel space, to get acceptable performance[3]. In this way the size grows and the kernel cannot be fully held in the processor's cache memory.

D. *Extensibility and Portability*

Extensibility is the most prominent fact for microkernels. It is due to its size, one of the biggest contrasts to monolithic kernels. Adding new features to a monolithic system means modification or recompilation of the whole kernel, often including the whole driver infrastructure. If you want to apply a new memory management routine into a monolithic architecture, then you have to modify other parts as well. In case of a microkernel the services are interlinkedto each other through the message system. It is enough to re-implement the newmemory manager. Microkernel's also show their flexibility in removing features

III. RELATED WORKS AND MOTIVATIONS

The first microkernel conceptwas in Carnegie-Mellon University, the microkernel Mach operating system. Mach usually minimizes the kernel into a very small module. The Mach kernel only provides process management; thread management, IPC and I/O service. Due to more IPC overhead scientists began to redesign the structure of the kernel. Prof Dr. JochenLiedtke invented his first microkernel with low overhead in passing messages, L3. The L3 kernel directly passes the message between processes leaving the process security and authentication to user space servers. This design method greatly reduced massive IPC overhead which occurred in Mach system. Results were so feasible that on the same system where MachRequired 114 microseconds for even the most smallermessages, L3 send the same message for less than 10sec.[4]

Many people considered Mach operating system as poor performer because of the overheads in IPC. There are other several microkernel systems that claimed to boast better performance such as L4ka and Minix3. Asmost current OS are using monolithic kernel that's why the project uses monolithic operating systems for comparison. Linux is a typical monolithic operating system to be used.

IV. CONCLUSION

L4 has proved that speed is no more an issue against microkernels anymore. Their additional modification predestines them for different applications reaching from embedded to desktop systems. Their counterparts are less maintainable as compared to them. Due the independency of the different parts due to message passing, microkernel systems can be easily modified. IPC can be madefaster by clever communication algorithms. To achieve this performance, microkernel must be directly optimized for the processor, where they are assumed to run on. That way, the full energy of a processor can be used directly.

REFERENCES

[1] William Stallings. *Operating systems.Internals anddesign principles*.3rd (international) edition.1998.
[2] Wang Chengjun, "*The Analyses of Operating System Structure*", Knowledge Acquisition and Modeling. Second International Symposium Volume 2, pp.354-357, 2009.
[3] Messer, A., Wilkinson, T., "*Components forOperating System Design*", Object-Orientation in Operating Systems, 1996., Proceedings of the FifthInternational Workshop, pp. 128-132, 1996.
[4] Zhou Qingguo, Ding Ying, McGuire, N., Li Canyu, Cheng Guanghui, and Hu Bin, "*A Case Study of Microkernel for Education*", IT in Medicine &Education, 2009.ITIME '09. IEEEInternational Symposium Volume 1, pp. 133-136, 2000.
[5] Dandamudi, S.P,*Guide To Assembly Language Programming In Linux*", Ottawa, Canada, 2005.