GrailOS: A micro-kernel based, multi-server, multi-personality operating system

Ben Leslie

National ICT Australia &
University of NSW, Sydney 2052, Australia
benjl@cse.unsw.edu.au

Abstract

A multi-server, multi-personality operating system has often been seen as the holy grail of the micro-kernel based systems research agenda. Unfortunately to date there have been limited success in achieving this grand vision, and it is no longer a hot-topic in the operating systems community, despite a recent renaissance of micro-kernels in general.

In this paper I argue that new problems exist that motivate the need for such an operating system structure, and that recent advances in technology allow such a project to succeed now, where it had failed in the past. Finally I outline a set of research to be undertaken to attack this problem.

1. Introduction

Workplace OS (WPOS) [FCT98, Raw97] was meant to be the project that implemented IBM's "Grand Unification Theory of Operating Systems". The aim of this ambitious project was to use a microkernel based system design to provide an operating system that could be used to support a variety of operating system personalities, including TalOS [Mye95], OS/2, UNIX and a DOS/Windows 3.1 environment. Unfortunately the product did not live up to expectations in terms of functionality or performance and the project was cancelled, after an estimated expenditure of $US2 billion.

Such a failure has led, on the whole, to a disinterest in similarly structured operating systems, however with 10 years of collective experience and new motivations, I believe it is time once again to consider an operating system structured in such a way.

One of the original motivations of WPOS was to provide a common software base, in the form of a microkernel, across a range of devices from small PDAs to large parallel processing machines. The benefit being a reduction in engineering costs as only one code base need to be maintained over a large set of products. The range of products in which full-featured CPUs are used has increased recently, with products such as mobile phones, music players, and games consoles all moving towards powerful processors, running complex software stacks. The motivation to have a common operating system base for a wide range of products is still strong in the community, and is best seen with the wide range of processors on which Linux runs, and to a lesser extent to variety on which the Windows platform runs. It is clear that such code base reuse is still viewed as important, however it is clear that achieving this does not required a microkernel based design.

A similar motivation of the original work was to be portable across platforms and CPUs. Although many of the world's desktops are now homogenised on the Intel IA-32 processor architecture, there still exist a wide range of different architectures and platforms including ARM, MIPS, Itanium and PowerPC. Most of the major operating systems, including Linux, Mac OS X, and Windows are portable across at least two architectures, and others such NetBSD run on almost every processor and platform available. It is clear that portability is still an important facet for today's operating system, but once again traditional operating system structures have proved adequate in accommodating this requirement.

Allowing the user to run applications designed for different operating systems was one of the most important features of WPOS, and such multi-personality is still seen as important today. Microsoft Windows supports a range of personalities including DOS, native Windows and POSIX [Cus93]. FreeBSD has a Linux emulation feature. The WINE project for running Windows applications on top of Linux is quite popular. A more heavyweight approach which is often used to achieve this is running an alternative operating system through use of a simulator such as VMWare or VirtualPC. While these approaches work they incur high cost in both performance and maintenance required to manage the additional operating system.

While the reasons that motivated the original work on multi-personality, multi-server operating systems still exist today, there are a number of new problems that I believe a new operating system structure can help solve.

Robustness of operating systems is becoming more important. The monolithic approach to system design means that almost all the new features to operating systems are added to the kernel. Any code in the kernel runs without protection and therefore any bugs are able to bring down the whole system. The effects of this can range from a bad user experience to financial loss due to down time to more serious consequences where products are used in safety-critical or life-critical applications. The increase in the number of modules in the kernel also has the effect of increasing unintended coupling of modules, leading to an unmaintainable system [SJW+02]. Perhaps the worst and most thoroughly documented problem here is device drivers [CYC+01, SMLE02], which are often supplied by third parties and are the most likely part of the system to cause robustness problems. There have also been some approaches for isolating device drivers [LCFD+05, SBL03] and appropriately structured isolation has been shown to improve system robustness [SABL04]. Designing a full multi-server operating system provides the opportunity to increase the robustness of not just device drivers but other system components as well, so that the effects of one failing server are contained and applications and other servers are not necessarily affected.

A related area to robustness is the security of systems. Security can mean a lot of different things but one of the highest visibility problems here is the proliferation of viruses and malware which current operating systems fail to adequately protect against. While viruses are perhaps the biggest problems on single-user systems data security is important on multi-user systems, such as shared network servers. There are other less obvious multi-user devices such as mobile phones, where the network provider considers itself the primary user of the device and requires protection from the person using the device. A microkernel based system provides the potential to reduce the trusted computing base to a minimal size. Secure applications can be developed in such a way that they limit themselves to using servers they consider trustworthy.

Apart from the security concerns, resource management is another important aspect of multi-user machines. This problem is seen on shared servers where operating systems have more or less failed to adequately isolate independent users on the system and a virtual machine approach has become popular way of overcoming this problem. The problem also manifests itself on embedded devices where there are certain applications with soft and hard real-time requirements that need to be able to obtain an adequate share of the machine to operate correctly. A microkernel based operating system should allow resources to be allocated among entities at the lowest level.

A final motivation is the flexibility of the system. Existing monolithic systems provide a fixed interface and provide abstractions which are not always the most appropriate for a given workload. Databases are often given as an example of where the operating system gets in the way of what the program really wants to do. Another example in the use of virtual machines, where the underlying operating system abstraction don't map well to the virtual machine abstraction, and performance is often lost during the impedance mismatch. A multi-personality operating system would allow different applications to run on a different personality. For example you could have a dedicated JVM personality used for running Java applications, running side by side with a Linux personality for running normal applications. A multi-server based system also provides the potential for users to write and use their own extensions without the need to escalate privileges and install the extension in the trusted kernel. Appropriate flexibility can help contribute to overall system security.

Improving the flexibility, robustness, security and resource management of operating systems while also providing multiple user APIs and portability provide the motivation for revisiting a microkernel based, multi-server, multi-personality operating system.

2. Design

In attempting to explore whether a microkernel based, multi-server, multi-personality operating system is feasible and useful I plan to design and construct **GrailOS**, an operating system built with this structure.

GrailOS will support four different personalities:

- A native personality which will provide direct access to servers through a high performance API.
- A Linux personality will provide binary compatibility for Linux applications.
- A Windows personality for running Windows applications.
- A Java personality for running Java applications.

GrailOS will support three different architectures to cover a range of different hardware environments:

- ARM will be used as a representative of the embedded style computer.
- EM64T will be used as a representative of desktop level computers.
- Itanium will be used as a representative of high performance server level computers.

Figure 1 shows a very high level overview of the basic structure of the system. At the base is a microkernel which will be the only code to operate in privileged mode. The microkernel used for this work will be either

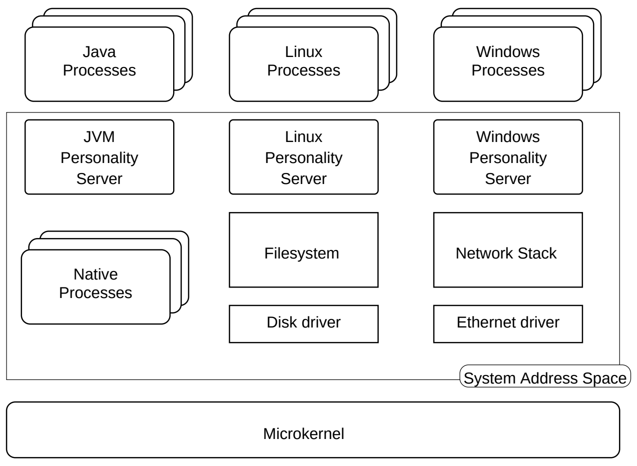

Figure 1. Overview of system structure

NICTA::Pistachio-embedded [NIC05] L4 implementation, or the new implementation based on the seL4 [Elp04] API. The microkernel is used to provide low level mechanisms that deal directly with the hardware, including virtual memory, processes and threads.

The system address space contains the servers that implement the multi-server operating system. These include device drivers such as ethernet and disk. It also includes important shared services such as file systems, and network stacks. Also in the system address space are the personality servers which provide the functionality needed for each of the different personalities supported. The processes for each personality run on top of their respective personality server.

From such a high level the project appears relatively straight forward, unfortunately there are difficulties when the detail is examined. In the following sections some of the design problems that need to be overcome are listed. This is by no means an exhaustive list, but is intended to give the reader a better idea of some of the problems faced in this endeavour.

2.1 Binary compatibility

One of the important things required when providing a multi-personality OS is how to run existing binaries. For this project we do not consider running binaries of differing instruction sets. The main problem is how to trap any system calls the process might make and then emulate that functionality. This one small problem opens a wide range of possible implementation techniques, such as system call redirection, system call trampolines, and binary rewriting.

2.2 Inter-server interfaces

Another important design decision to be made is how to communicate and transfer data between the servers and the processes themselves. Previous attempts at creating multi-server operating systems have shown that the overhead from extra context-switching has to potential to cause serious overall performance problems. Previous work looking at user-level device drivers showed that careful construction of interfaces to avoid excessive context-switching and data copying are imperative in achieving good performance. Finding appropriate interfaces for communicating between different servers such as file system, disk, and buffer-cache are seen as being extremely important for the success of the project.

2.3 Reuse of existing components

Although the overall system is being designed from scratch, to have a useful and practical system, reusing existing components from existing operating systems is seen as an important part of the project. Although most existing operating system are monolithic in nature they usually posses some form of modularity in the form of loadable device drivers and other modules such as file systems. There is a goal of designing system so that some of these modules can be reused, increasing the range of features available.

2.4 Resource management on shared services

Although the microkernel provides a perfect base for performing resource isolation of various subsystems, the use of shared servers complicates things considerably. In a naive implementation .

This section has provided an overview of the design of GrailOS and provided details of some of the design difficulties that will need to be overcome to produce a successful system. The next section examines how the research will be evaluated.

3. Evaluation

To determine whether this project is a success or not, a number of experiments are proposed to test the various aspects of system and compare against existing systems. Experiments are grouped by which aspects of the system is being tested.

3.1 Performance

Although people are usually willing to give up some performance in return for new features, they generally don't want to pay too much. The aim of this work is to be able to have a less than 10% overhead on application performance. Obviously there are a wide range of possible workloads, so some of the macro level benchmarks to run will be: running a kernel compile on Linux, running SpecJBB2000 on the Java personality and running SpecWEB on Windows.

To get a better understanding of the performance characteristics other micro-benchmarks experiments will also be run including, bonny, iozone, lmbench, ipbench.

3.2 Resource Management

We want to show that we are able to provide the isolation guarantees that we speculate that we can provide. To do this we will run the macro-benchmarks as in the performance evaluation, however rather than simply running one copy we will run multiple copies each with a specified percentage of the system allocated to them. Success will be defined as achieving performance within 5% of the allocated resources.

3.3 Robustness

To check robustness we will use a fault injection tool which perturbs components in the operating system. The operating system should be able to recover from these faults and allow user applications to continue operating.

3.4 Security

Security is a difficult thing to systematically benchmark. One approach is a honeypot approach where it is exposed to people attempting to break the security.

Other approaches involve intentionally creating security holes in various components inside the operating system and showing that the violation is restricted to a given component or subsystem.

The amount of code in the trusted computer base will also be used as a metric.

3.5 Flexibility

If the system is sufficiently general and flexible it should be possible to implement and use a newly created system personality without needing changes to the system and being able to reuse the existing services. To test this plan to have a third party, likely an undergraduate student, attempt to write a new personality for GrailOS. The system will be evaluated on the number of serious changes required to the design and implementation to accommodate the new personality.

4. Research Contribution

Many of the techniques I currently plan on using have been reported by researchers before, however these techniques are yet to be used together to produce a high-performance, multi-server operating system. Showing that building such a thing is possible i a significant contribution in itself. However it is also expected, that while pursuing this research new insights and techniques of system construction will be found which will prove valuable to the operating systems community as a whole.

References

[Cus93] Helen Custer. *Inside Windows-NT*. Microsoft, 1993.

[CYC+01] Andy Chou, Jun-Feng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating systems errors. In *18th SOSP*, pages 73–88, Lake Louise, Alta, Canada, Oct 2001.

[Elp04] Kevin Elphinstone. Future directions in the evolution of the L4 microkernel. In Gerwin Klein, editor, *Proceedings of the NICTA workshop on OS verification 2004, Technical Report 0401005T-1*, Sydney, Australia, Oct 2004. National ICT Australia.

[FCT98] Brett D. Fleisch, Mark Allan A. Co, and Chao Tan. Workplace microkernel and OS: A case study. *Softw.: Pract. & Exp.*, 28:569–591, 1998.

[LCFD+05] Ben Leslie, Peter Chubb, Nicholas Fitzroy-Dale, Stefan Götz, Charles Gray, Luke Macpherson, Daniel Potts, Yueting Shen, Kevin Elphinstone, and Gernot Heiser. User-level device drivers: Achieved performance. *J. Comput. Sci. & Technol.*, 20(5), Sep 2005.

[Mye95] W. Myers. Taligent's CommonPoint: the promise of objects. *IEEE Comp.*, 28(3):78–83, Mar 1995.

[NIC05] National ICT Australia. *NICTA L4-embedded Kernel Reference Manual Version N1*, Oct 2005. http://ertos.nicta.com.au/Software/systems/kenge/pistachio/refman.pdf.

[Raw97] Freeman L. Rawson III. Experience with the development of a microkernel-based, multiserver operating system. In *6th HotOS*, Mar 1997.

[SABL04] Michael M. Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy. Recovering device drivers. In *6th OSDI*, San Francisco, CA, USA, Dec 2004.

[SBL03] Michael M. Swift, Brian N. Bershad, and Henry M. Levy. Improving the reliability of commodity operating systems. In *19th SOSP*, Bolton Landing (Lake George), New York, USA, Oct 2003.

[SJW+02] Stephen R. Schach, Bo Jin, David R. Wright, Gillian Z. Heller, and A. Jefferson Offutt. Maintainability of the Linux kernel. *IEE Proc.: Softw.*, 149:18–23, 2002.

[SMLE02] Michael M. Swift, Steven Marting, Henry M. Levy, and Susan G. Eggers. Nooks: An architecture for reliable device drivers. In *10th SIGOPS Eur. WS*, pages 101–107, St Emilion, France, Sep 2002.