Case Studies In Modern Dynamic Adaptive Operating Systems

Nur Hussein
Computer-Aided Translation Unit
School of Computer Science
Universiti Sains Malaysia
11800 Pulau Pinang
Malaysia
hussein@cs.usm.my

ABSTRACT

Dynamic adaptive operating systems are operating systems that are able to provide a high degree of customisability in a flexible, configurable environment to users and applications. This customization can occur during run-time or boot-time, hence enabling the operating system to adapt to different workloads. This paper is a survey of three different kernels utilizing three different approaches to creating this capability: Linux with loadable kernel modules, Xok which is an exokernel and K42 which is a microkernel. A discussion of the tradeoffs between the three different implementation strategies is discussed.

KEYWORDS

Operating systems, microkernels, exokernels, dynamic adaptive kernels, Linux, K42, Xok

1. Introduction

Operating systems designers have long grappled with the problem of getting their systems to work well under a variety of workloads. Frequently, the algorithms and policies of a general purpose operating system kernel need to work adequately for all of the different applications that will run on it. Unfortunately, algorithms and policies that allow one type of application to run well may hinder the performance of other types of applications. For example, a scheduler policy that favours the throughput of long running CPU intensive processes will cause processes that rely on low latency scheduling to suffer (such as multimedia applications) [4].

On most production operating systems, policies are optimised to accommodate the most common workloads, and thus will perform adequately well (although not at an optimum) for the majority of users. However, such operating systems are known to operate at less-than-optimal performance for specialised applications such as large databases, and also when the workload hits "corner cases". There is an active area of research which aims to enable operating systems to adapt to the changes in workload demands and provide an efficient operating environment for the workloads that it has to handle. This paper is a review of 3 approaches to *dynamic adaptive* operating systems, illustrated by 3 different operating system kernels: Linux, Xok [6] and K42 [5].

2. Taxonomy And Nomenclature

Denys *et al.* [2] define a taxonomy for customizable operating systems that is divided into two orthogonal evaluation criteria:

(i) The initiator of the OS adaptation
- A *human* administrator
- An *application* that the OS runs
- *Automatic adaptation* by the *OS itself*

(ii) The time that adaptation occurs
- *Static adaptation* at design/build/install time
- *Dynamic adaptation* at boot time or run time

In this paper, we are concerned with operating systems that offer *dynamic adaptation*. Dynamic adaptation takes place when parts of the operating system can be selected, used and replaced at will when the operating system is running, or when the system is first booted. All the operating systems discussed here imply that a *human* administrator is required for adaptation, and no automatic updates of the OS or applications takes place.

3. Linux : A Monolithic Kernel

Linux was originally designed as a monolithic kernel, and to the time of writing, still remains so for most of its implementation. The facility of loadable kernel modules was added to the Linux kernel as soon as it became a usable system, and is currently a standard feature on the Linux-based systems.

Loadable kernel modules are compartmentalised blocks of compiled kernel code that can be loaded and unloaded into kernelspace at will by the administrator.

Once a kernel module is loaded into the kernel, the module loader will link the code in dynamically. Then the module initialization functions are called. The entire kernel namespace and codespace will be visible to the module. The result of this is that an error in a module will cause the entire kernel to fail.

Linux employs a relatively simple mechanism to ensure safe loading and unloading of kernel module. All kernel modules need to be accessed while setting a reference counter. A module may only be unloaded when the reference count accessing it is 0.

The Linux implementation of dynamic code loading is relatively straightforward, and is deployed in production systems. The next sections will discuss experimental kernels which use more interesting techniques for runtime OS dynamic adaptation.

4. Xok : An Exokernel

An exokernel [3] is an innovative twist on the idea of microkernels, where instead of implementing operating system services as servers in userspace, they are implemented as a *user library*. The kernel itself is a stripped-down, bare-bones implementation that only provides protected access to the bare hardware. Exokernels provide a mechanism to multiplex the hardware, divide the memory into pages, manage the CPU, translation look-aside buffer (TLB), addressing contexts, interrupts and exceptions. Much of the higher level kernel functions such as networking, memory management and filesystems are implemented in user libraries which can be linked to different applications. These user libraries, called *library operating systems*, set the policies of the system.

A consequence of this design is that different applications can link to different library operating systems based on the requirements of the application. This allows different library operating systems to execute different implementations of OS services at runtime. If a specific application needs to swap implementations, it can be done as easily as re-linking the application with another library OS (see Figure 1, adapted from a diagram in [8]).

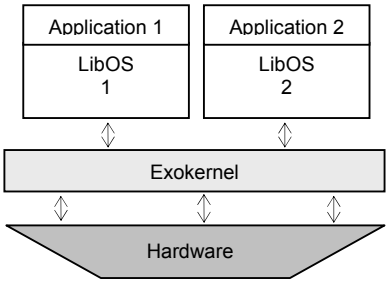

Figure 1: Organisation Of Exokernels, Library Operating Systems And Applications

Xok is an exokernel designed to run on Intel X86 hardware. Xok was used as a test platform by Kaashoek *et al.* to evaluate exokernels as a viable platform for running applications with custom operating system libraries. The default library OS that runs on Xok is called ExOS, which supports a 4.4BSD userspace.

It was found by Kaashoek *et al.* that Xok/ExOS performed as well with FreeBSD and OpenBSD, two production operating systems which also implement a 4.4BSD userspace. Most of the programs that run on FreeBSD and OpenBSD can run unmodified on Xok/ExOS, and thus proves the viability of an exokernel based environment compared to traditional operating systems. However, the more interesting experiments carried out by Kaashoek *et al.* were the customised instances of OS libraries linked to two different applications: XCP, a file copying program that exploits a low-level disk interface and Cheetah, a web server which uses a customised fast I/O library. In both cases, the domain-specific implementations of OS functionality helped the programs achieve better performance than their counterparts on traditional operating systems.

5. K42 : A Microkernel

K42[4] is a research microkernel for 64-bit cache-coherent multiprocessors. It was designed from the ground up to be modular and object-oriented in design. K42 has explicit support online reconfiguration of its software components. This can be used for two things:

(i) Hot-swapping of kernel components, while the system is executing
(ii) Interpositioning of software components, which is providing a software wrapper for existing components, hence extending its functionality (for example, dynamically

inserting a profiler like in Figure 2, adapted from a diagram in [9]).

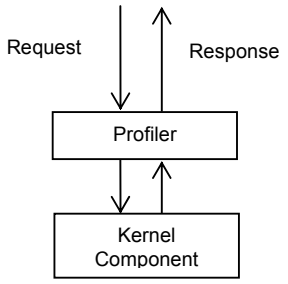

Figure 2: Interpositioning A Profiler Around A Kernel Component

Soules *et al.*[9] describe the four main requirements necessary to support online reconfiguration of a running kernel:

(i) Each software component must be encapsulated within well-defined boundaries and interfaces.
(ii) Software components must be able to achieve *quiescent state* when state transfer takes place. A quiescent state is when the component enters a state where all data access activity to and from the component has stopped and it is safe to swap it out.
(iii) It must be possible to transfer state between the two software components being hot-swapped.
(iv) All external references to the component must be updated.

In the implementation of K42, the choice of programming paradigm and language (it was designed in an object-oriented way and implemented in C++) helped the kernel components define the interfaces between software components. Components designed to be hot-swapped are also created to be compatible enough so that state transfer can take place. Also, all references to objects take place through an intermediate lookup table called the Object Translation Table (OTT). Therefore, whenever components are replaced, only the OTT needs to be updated for that object and all external references are automatically updated.

The K42 kernel determines quiescent states by applying a technique similar to the Read-Copy-Update (RCU) algorithm [7]. All events in K42 occur with the creation of kernel threads, which are short lived and non-blocking. All kernel threads are associated with a "generation" or time period when the thread is created. K42 uses a counter to determine if the total threads generated within a generation have terminated. When this happens, no new threads are created and it enters a quiescent state. RCU-based techniques have also been used in recent Linux kernels to support safe module loading and unloading.

The hot-swap operation in K42 is described by Baumann *et al.* as taking place in six steps [1]:

(i) Before the update, all access to an object's methods takes place through the OTT.
(ii) Using interpositioning, a *mediator* object is used to track incoming method invocations to an object that is targeted as a hot-swap candidate. It forwards requests for calls to the object but also checks the thread generation and waits for the number of threads for the previous generation to reach zero.
(iii) Once the previous generations' threads have finished running, the mediator object will block new thread creation requests until all threads from the current generation have also terminated. Recursive calls however are not blocked to avoid deadlock.
(iv) Once all the calls that the mediator has forwarded has completed, the object reaches quiescent state and can be safely swapped out. The new object must be able to receive the state information compatibly from the old one, when the state transfer takes place.
(v) The OTT is updated by the mediator to point to the new object, and the method invocation calls currently blocked are forwarded to the new object.
(vi) The mediator and the old object are destroyed (de-allocated).

6. Discussion On Design Approaches

Trade-offs exists between each of the design approaches used by Linux, Xok and K42. Monolithic kernels such as Linux perform very well, but at the expense of modularity and inter-component isolation. A failure in any part of Linux will cause the entire system to fail, causing a kernel panic. Hence, inserting buggy or unstable code into a running kernel can have disastrous consequences.

Microkernels have long since been praised for their modularity. K42's hot-swapping mechanism provides a level of flexibility that monolithic kernels cannot match. However, this flexibility comes at a price. When swapping components, the system has no way to verify that the functionality provided by the new component safely replaces the functionality in the older component. As such, this problem cannot be solved automatically, thus programmer diligence

is required to ensure safe component-swapping occurs.

Finally, exokernels provide an interesting look at an alternative design for operating systems. We also get a remarkable degree of flexibility in customising operating systems services to tailor to different applications. However, creating custom library operating systems is a more complex endeavour than simply tuning existing production kernels to adapt to specific application behaviour. As of writing, no exokernel based system has been deployed in production, and it remains to be seen if the concept will gain traction amongst users.

7. Conclusion

While three different kernels have been discussed in this paper, this is by no means an exhaustive list. However, the most promising or interesting methods in modern research and production kernels that employ dynamic adaptive behaviour have been covered in this paper. It is hoped that this survey is a good starting point for those interested in recent operating systems developments to gather more information on the subject.

8. Acknowledgements

Thank you to Seth Arnold, Bert Hubert, Zwane Mwaikambo, William Lee Irwin III, C. Donour Sizemore, and the rest of my friends who have given me input on the material in this paper.

9. References

[1] Baumann, A., Kerr, J., Appavoo, J., Da Silva, D., Krieger, O., and Wisniewski, R.W., "Module Hot-Swapping for Dynamic Update and Reconfiguration in K42", *Proceedings of the 6th Linux.Conf.Au*, Apr 2005.

[2] Denys, G., Piessens, F., and Matthijs, F., "A Survey of Customizability in Operating Systems Research", *ACM Computing Surveys*, 34(4), Dec 2002, pp. 450—468.

[3] Engler, D.R., Kaashoek, M.F., and O'Toole, J. Jr., "Exokernel : An Operating System Architecture for Application-Specific Resource Management", Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, Dec 1995, pages 251—266.

[4] Hussein, N., Kolivas, C., Haron, F., and Chan, H.Y., "Extending the Linux Operating System for Grid Computing", *Proceedings of the APAN Network Research Workshop*, 2004.

[5] IBM Research, "The K42 Operating System", http://www.research.ibm.com/k42, 2006

[6] Kaashoek, M.F., Engler, D.R., Ganger, G.R., Briceño, H.M., Hunt, R., Mazières, D., Pinckney, D., Grimm, R., Jannotti, J., and Mackenzie, K., "Application Performance and Flexibility on Exokernel Systems", *Proceedings of the Sixteenth ACM symposium on Operating Systems Principles*, 1997, pp. 52—65.

[7] McKenney, P., Sarma, D., Arcangeli, A., Kleen, A., Krieger, O., and Russell, R., "Read Copy Update", in *Proceedings of the Ottawa Linux Symposium*, 2002.

[8] MIT Parallel and Distributed Operating Systems Group, "MIT Exokernel Operating System", http://pdos.csail.mit.edu/exo.html, 2006.

[9] Soules, C.A.N., Appavoo, J., Hui, K., Da Silva, D., Ganger, G.R., Krieger, O., Stumm, M., Wisniewski, R.W., Auslander, M., Ostrowski, M., Rosenburg, B., and Xenidis, J., "System Support for Online Reconfiguration", *Proceedings of the USENIX Annual Technical Conference*, Jun 2003.