

# Application Specific Operating Systems

## MAE Document

Lamia Youseff ([lyouseff@cs.ucsb.edu](mailto:lyouseff@cs.ucsb.edu))

### Abstract

Several application domains have suffered from the general policies dictated by general purpose OS (GPOS), like the UNIX system [1]. Therefore, since the early 1990s, OS community realized the need for customizable OS, where the application performance can be enhanced by customizing the OS for a certain application. However, since then, no one acceptable system was widely used. In this report, we present our research quest of *investigating the feasibility of automatically generating an application specific operating system (ASOS) by customizing GPOS for scientific applications*. Our four focus areas in this study are: customizable OS, scientific applications' characterization, Profiling and optimizing techniques, virtual machine monitors.

### 1. Introduction

Since the mid 1960s, operating system (OS) research has been focused on developing general purpose operating systems (GPOS) that can execute all applications, delivering acceptable performance for each. Moreover, in 1974, Dennis M. Ritchie and Ken Thompson from Bell Laboratories introduced the early UNIX system which afterwards became the basics for widely-used production operating systems. Their main design goal for their new OS was to introduce a "general, multi-user, interactive OS" [1], when they had to trade off some of the system performance to support those features. However, this trade off had hugely harmed the performance of several domains of applications, especially resource intensive ones. Examples of those application domains are database systems, high performance applications and scientific applications. The fundamental performance hinder for those domains was the general policies dictated by the operating system. Thus, in order to enhance their applications' performance, those applications' developers had to reverse-engineer the OS such that to introduce limited changes for limited performance gains. Their other choice was to build the OS from scratch for an enhanced performance. However, these were not the best solutions out there, since reverse-engineering the OS is a very complicated task, and require the developer to know the OS internals, while the other option of building an OS is even more complicated and time-consuming task. Therefore, the OS community realized the need for operating systems which can be customized for special kinds of applications. Since the early 1990s, several contributions in Customizable OS were made from a number of research groups [2-5, 20-25]. Several approaches were considered to develop those flexible OSes including extensible OS, OS frameworks and OS libraries. However, no one customizable OS was widely acceptable and used, due to the several complications involved in developing and deploying such systems, and/or the high overhead induced by the customizations mechanisms on the new system's performance.

Therefore, the need for an application specific OS was evident to many of those application domains' developers. In this study, we are pursuing *the quest of developing an efficient application specific operating system (ASOS)*. We are investigating the feasibility of introducing *an automatically-generated ASOS, by customizing a GPOS like Linux* to obtain an *Application Specific Linux (ASL)*. Efficiency of our ASOS is not only measured in performance gains for the specific application it is developed for, but also in the simplicity of generating the system, the safety of the approach, its portability across different platforms, and its support for legacy applications. We are aiming at avoiding all of the downfalls of the earlier systems, and introducing an ASOS that will be used in production systems as well as in academia.

In order to customize a GPOS to enhance the performance of a specific application, we needed to study different research areas related to our goal. First, we realized the crucial need to understand precisely the application's needs and how it is using the H/W resources. Static and Dynamic profiling of the application gave us a very enlightened insight about the application execution patterns. This insight would guide us through the process of GPOS customization for a certain application. Moreover, to focus our efforts on a specific domain of applications, we chose scientific applications, since they are very resource intensive, and keen to use the computational power on the machinery they are being deployed on. Therefore, automating the development of ASOS for this domain can be used in production systems. Finally, Virtual Machines Monitors (VMM) is another very major research area to investigate, since they allows the sharing of different Oses to the same HW and multiplexes the resources among them. Therefore, we were interested in studying VMM research, and how recent projects were able to eliminate most of the overhead associated with virtualization in VMM.

In the rest of this paper, we will overview the most important work done in the four fundamental areas related to our research quest: Customisable OS, VMM, pattern profiling and optimizations, and scientific application. In section 2, we will be illustrating a brief analysis of four important customisable Oses. In section 3, we will compare between three VMM introduced in the last decade, these VMM are Denali, Xen and ADEOS. We will be illustrating more on why we chose scientific applications as our application domain in section 4, as well as specific characteristics of applications in this domain. Finally, pattern profiling and optimization techniques are inspected and illustrated in brief in section 5. We summarized our paper and conclude it in the last section.

## **2. Operating Systems Customizations**

Many approaches had been taken to obtain a customizable OS, since this research area started in the early 1990s. However, due to space constraints, we will only illustrate on the most important approaches taken in that regard. Nevertheless, the different approaches' main goal was to support the operating system flexibility in order to give it the ability to enhance its applications' performance.

The first approach was introduced by MIT systems group through their Exokernel [2] project. The idea was to give the application developer more flexibility and control over the hardware, by separating HW protection from mechanism implementation. Thus, Exokernel focused on lowering the kernel interface to just above the hardware, in such a way to eliminate most of the abstraction from the kernel. The only kernel's responsibility in Exokernel is to protect the physical resources and multiplex them among competing applications. Furthermore, the OS services were then implemented in user-space OS libraries, which can be manipulated by the application's developers hence after. Therefore, the application's developer can specialize the OS library in the best way that serves his/her application's needs. This approach was widely accepted in Academia. However, it did not gain any commercial interest because of the complication of the exported kernel interface, its lack of portability, as well as the complication involved in specializing the OS library for each application. A similar system to Exokernel was SPIN [20], where the application can specialize the kernel services by downloading extensions into it. The kernel was protected against those un-trusted extensions by requiring them to be written in a type-safe language (Modula-3 [36]). On the other hand, several Micro-kernels allowed kernel services' specialization through downloaded extensions [34, 35], without verifying the safety of the kernel. They either ensured that the downloaded extensions are from trusted sources (ex. loadable kernel modules [37]), or depended on the good-will of the extensions (ex. MS-DOS) [34]. The latter resulted in the kernel's instability and the introduction of several security holes in the kernel due to allowing un-trusted extensions to run in its protected address.

Vino [3, 33, 40] is another OS that also allowed kernel extensions, written in C++ to be downloaded into the kernel. It used software fault isolation (SFI) technique to protect the kernel address space from misbehaving extensions. However, the more interesting aspect in Vino is the self-monitoring, self-adapting OS nature. Vino was designed to allow the kernel to monitor its own operation, analyze the logs of the processes running on it and dynamically decide on deploying and switching between different policies to deliver better application performance. This feature in Vino made it unique among other OS systems. However, Vino designers had only discussed its dynamic nature in their literature, but the system was never implemented [37]. Probably, One of the difficulties that faced the implementation of the project was adapting the system for several applications at the same time. Several applications could have several conflicting HW needs, and thus requires several conflicting OS polices to be adapted at the same time, which was not feasible. However, Vino designers did not reveal the difficulties that faced their system implementation nor did they expose the reasons of their project failure.

Specializing commodity operating systems services was a different approach taken by the Synthetix project [4, 5]. In [5], Synthetix specialized UNIX read system call for HP-UX systems, making assumption about the applications usage pattern of the files system. Those assumptions were validated with synthetix guards, whose runtime responsibility was to decide if the assumption is still valid or not. Upon validation of the assumptions, the application would use the specialized version of the OS services; else the general version was used. As an example, consider specializing the FS to be a one-file-fs, where all of the overhead due to locks and checks of having more than one file opened at a time, is eliminated. Synthetix would direct the application to use one-file-fs when there only one file open in the system, and fall back to the original fs when this assumption is broken. Therefore, the application would benefit from bypassing the overhead encountered in normal fs when executing in the specialized fs. Moreover, they also defined a cost/benefit analysis to identify when using specialized OS services would be beneficial to the system. One fundamental drawback of Synthetix is that the system designer should be the one deriving the system assumptions, and defining where the guards should be placed and checked in the OS. This requires prior knowledge of OS internals, which add a huge overhead to the software development cycle.

Another class of customizable OS is OS frameworks. OS frameworks provide different components from which a developer can build an OS instance. Scout [32] is an example of this category, where it is a communication oriented operating system, designed to support a certain class of applications; i.e. network applications. A specialized instance of Scout is generated from a collection of modules, which are afterwards optimized based on the application's requirements, in such a way that enhances the applications performance. Other examples of OS frameworks are Flux OSkit [38] and Choices [39]. The main drawback of this category is that all of the OS decisions and configurations must be decided at the design time, initialed by the OS designer. Hence, all of the applications needs have to be well understood and integrated into the OS design phase, as well as being constant and statically-anticipated throughout the application's lifetime. Moreover, OS framework did not simplify any of the complications involved in OS building an OS.

Therefore, although different approach were taken to obtain a customizable OS, all of them (except for Vino) suffered from the same drawback of the complexity associated with customizing OS services for a certain application. It was either the system admin/designer or the application developers' responsibility to customize the OS for a certain application at hand, which add more complication to the already sophisticated software development cycle. We anticipate

that this is the major cause of not having any of those systems in production settings. On the other hand, Vino was the only operating system that was dynamically self adapting to the application's needs. But unfortunately, the system was not implemented.

### 3. Virtual Machines Monitors

Virtual machine monitors (VMM) [31] is a technique used to allow sharing of several OSES to the same HW resources. However, the huge overhead associated with virtualization in VMM discouraged the usage of VMM in production system. Recently, Virtualization overhead had been mostly eliminated due to the introduction of the introduction of para-virtualization [7] technique. Para-virtualization, as defined in [7] is the concept of slightly and strategically modifying the underlying physical hardware architecture while virtualizing it. In this respect, three fundamental projects were of interest to us: Denali [7] (University of Washington), Xen of Cambridge University [8] [9] and ADEOS of Opersys [6].

Denali is an isolation kernel, i.e. a simple thin software layer that runs directly on hardware, and whose function is to subdivide the physical resources among a set of fully isolated protection domain. They use the concept of para-virtualization, where Denali does not precisely emulate the underlying physical hardware. Implementing the concept of para-virtualization in Denali had eliminated most of the overhead normally associated with virtualization. Moreover, Denali provides easy techniques for applications check-pointing, cloning and migration, since it is collecting the memory footprint and virtual device state of the guest OS, running on top of it.

Xen is another VMM whose developers had build their system around the concept of para-virtualization. Their reported virtualization overhead; which was also verified by other groups, ranged between 0% for computational intensive applications (SPEC Int2000 scientific code [30]), and 8% for I/O intensive applications (DB-applications), in comparison with UML [29] and VMWare 5-8% overhead for computational intensive applications and 56-88% overhead for I/O intensive applications. Xen also supports Linux, Windows XP and BSD porting with minimal effort. However, the only drawback with Xen is that it was designed for x86 machines. Xen designers did not mention any plans for porting the system to other machines.

ADEOS stands for Adaptive *Domain Environment for Operating System*. Adeos can be seen as a VMM which enables other un-modified OS to run on top of it, where it just virtualized some hardware commands in order to be able to allow sharing of multiple OSES to the same hardware. Meanwhile, ADEOS can be also considered a nano-kernel, since it provides a miniature hardware management facility that can be used thereafter to build a production OS. Performance measure for ADEOS as a VMM was not reported in any of its publications, which discourages us from using ADEOS as a VMM layer below our ASL.

### 4. Scientific Applications' Characterizations

Scientific Applications are resource-intensive applications domain, which are keen to use the machine's resources in the most efficient method [12]. Therefore, there are several reasons behind our choice of scientific applications as an application domain for experimenting with specializing the Linux kernel for. First, most production scientific applications are deployed on supercomputing infrastructure, where the user is allocated a certain time period to run his/her code. At the application's runtime, the user is the only user utilizing the infrastructure and his/her application is the only code executing. Meanwhile, most of the time, the OS deployed is Linux, which in its design has traded off performance for time-sharing, multi-user support and protection. However, scientific codes are computational intensive, where in this setting, they would not prefer to trade off performance for these other system services. Moreover, scientific

codes execute for long periods of time, which in turn would gain from the dynamic auto-configuration that an ASOS would introduce. Additionally, most scientific application requires check-pointing, and sometimes migration of the computation, which scientific applications' developers have to implement. Implementing those OS-level services in the application is an overhead on the software development cycle. Thus, providing those services through an ASOS would help scientific applications' developers to focus more on the actual scientific code.

Scientific applications furthermore were reported in several researches to have a repeating pattern in using their physical resources. For example, B. Pasquale et al in [10, 11] made a statistical study of the static and dynamic characteristics of several scientific applications running on SDSC, on the span of a month. They have shown that I/O in most scientific application encounter three basic I/O phases. The first is an initial phases, which initializes the application I/O operation (i.e. reading an input file). This phase is characterized with a burst of I/O operations in small period of time. Then, the next phase is a repeating pattern of I/O operations, which takes most of the applications lifetime. Finally, another burst of I/O operation characterizes the final phase, which is for short period of the applications lifetime. They infer from their study that any scientific application would possess some variation of this behavior.

Other researches also studied the characterizations of scientific application, like in [26, 27, 28]. Splash-2 [12, 13] is Stanford scientific application suite, where it characterizes the different characteristics of several scientific codes, and drew possible place for optimization for those codes. Therefore, from the studied made in this area, we can conclude that scientific applications execution patterns can be studied and characterized, and thus it is the best candidate domain for generating ASOS for.

### **5. Optimization and Profiling techniques**

In order to specialize an operating system for a certain application, we need some techniques to understand the application requirements and usage patterns of the hardware resources. Phase profiling [14,15] is a very useful technique in that respect. A program/application is divided into phases, where each phase represents an interval of the program instructions [14]. The goal of phase profiling is to find the similarities between different program's phases, such that it would discover the application behavior patterns. Similarities can be drawn based on different factors, like I/O behavior of the different phases, computational resource usage, or other HW resources usage. Therefore, using this profiling output, we can use them to understand the application behavior and therefore, customize the OS in the best way to serve that application.

Furthermore, Call-graph is another tool that can be used to understand the application behavior, as well as the performance bottlenecks of an ASOS, and optimize the OS based on its output. Lee et al, in [16] used Call graphs to isolate the kernel code used by their application and remove the dead code for smaller kernel footprint for embedded systems. Barton Miller et al [17, 18] also used call-graph to characterize the performance bottlenecks of different applications inside the kernel. They were able to introduce a performance gain of a factor of 2 to their test applications. Paradyn [19] is an automated user-space performance bottleneck finder, where it uses dynamic instrumentation to profile different application functions, and recognizes the performance bottlenecks in each.

In addition, Optimization techniques for ASOS can be applied more efficiently than for GPOS, since we have more insight into the application needs and requirements [33]. Several common optimization techniques can be reapplied to our ASOS to eliminate much of the overhead encountered with the general policies of GPOS and outcome better performance for our system.

Common optimization techniques include functions inlining, constant folding, value propagation, dead code elimination ...etc. Moreover, Inlining the application code into ASOS kernel space is another fine approach to eliminate most of the overhead encountered with domains-crossing. More optimization techniques can be applied in this respect.

## 6. Conclusion

In this paper, we defended the idea of Application specific operating systems (ASOS) and illustrated how it would be useful to certain domains of applications, specifically scientific applications. We had surveyed some efforts done in the area of customizable OS. Given that there is no one acceptable solution in this area till the time of writing this document, we are planning to avoid the down falls of these researches and aim at introducing a system that can be used as a production system as well as in academia. We will design our system for scientific application, which were described by several researches to posses certain patterns of operation. This property makes it feasible to predict scientific applications' hardware needs, and effectively specialize the OS to best serve it. We had also discussed several profiling and optimization techniques. Finally, we described the para-virtualization concept, as well as describing 3 VMM projects that implemented that concept to eliminate virtualization overhead in VMM. Combining the knowledge from these four different areas, ASOS would be a very effective and performance-oriented tool for deploying production scientific codes, as well as other application domains in the future.

## References:

- [1] Dennis Ritchie, Ken Thompson, The UNIX Time-Sharing System, ACM Communications. ACM 17(7): 365-375 (1974).
- [2] Kaashoek, M.F., D.R. Engler, G.R. Ganger, H.M. Briceno, R. Hunt, D. Mazieres, T. Pinckney, R. Grimm, J. Jannotti and K. Mackenzie, Application Performance and Flexibility on Exokernel Systems, Proceedings of the 16th Symposium on Operating Systems Principles, (Oct. 1997), pp. 52-65
- [3] M. I. Seltzer and C. Small, Self-monitoring and self-adapting operating systems. In Proceedings of the Sixth workshop on Hot Topics in Operating Systems, 1997
- [4] Crispin Cowan, Tito Autrey, Charles Krasic, Calton Pu, and Jonathan Walpole, Fast Concurrent Dynamic Linking for an Adaptive Operating System, In International Conference on Configurable Distributed Systems (ICCDs'96), Annapolis, MD, May 1996.
- [5] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang, Optimistic Incremental Specialization: Streamlining a Commercial Operating System, In Proceedings of the 1995 ACM Symposium on Operating Systems Principles, Copper Mountain Resort, CO, USA, pages 314--324, December 1995
- [6] Karim Yaghmour, "Adaptive Domain Environment for Operating Systems", Opersys Inc., June 2002
- [7] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble, Denali: A Scalable Isolation Kernel, Proceedings of the Tenth ACM SIGOPS European Workshop, Saint-Emilion, France, September 2002.
- [8] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebar, Ian Pratt and Andrew Warfield, Xen and the art of virtualization, In the Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), October 2003
- [9] Paul R Barham, Boris Dragovic, Keir A Fraser, Steven M Hand, Timothy L Harris, Alex C Ho, Evangelos Kotsovinos, Anil V S Madhavapeddy, Rolf Neugebauer, Ian A Pratt and Andrew K Warfield, Xen 2002, Technical Report UCAM-CL-TR-553, January 2003
- [10] Barbara K. Pasquale and George C. Polyzos, "Dynamic I/O Characterization of I/O Intensive Scientific Applications", Proceedings of Supercomputing '94, Washington, D.C., November 14-18
- [11] Barbara K. Pasquale and George C. Polyzos, "Dynamic I/O Characterization of I/O Intensive Scientific Applications", Proceedings of Supercomputing '94, Washington, D.C., November 14-18, 1994
- [12] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, The SPLASH-2 Programs: Characterization and Methodological Considerations, In Proceedings of the 22nd International Symposium on Computer

## Application Specific Operating Systems (ASOS)

---

Architecture, pages 24-36, Santa Margherita Ligure, Italy, June 1995.

[13] HomePage of Stanford Parallel Applications for Shared Memory (SPLASH), <http://www-flash.stanford.edu/apps/SPLASH/>

[14] Priya Nagpurkar and Chandra Krintz, Visualization and Analysis of Phased Behavior in Java Programs, ACM International Conference on the Principles and Practice of Programming in Java (PPPJ) June 16-18, 2004, Las Vegas, NV.

[15] D. W. Wall, Predicting program behavior using real or estimated profiles, In Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation, Toronto, Canada, 1991

[16] Chi-Tai Lee, Jim-Min Lin, Zeng-Wei Hong and Wei-Tson Lee, An Application-Oriented Linux Kernel Customization for embedded Systems, Journal of Information Science and Engineering, Vol. 20 No. 6, pp. 1093-1107 (November 2004)

[17] Alexander V. Mirgorodskiy and Barton P. Miller, CrossWalk: A Tool for Performance Profiling Across the User-Kernel Boundary, in International Conference on Parallel Computing (ParCo), Dresden, Germany, September 2003.

[18] Andrew R. Bernat and Barton P. Miller, Incremental Call-Path Profiling, Technical report (University of Wisconsin-Madison, February 2004)

[19] Harold W. Cain, Barton P. Miller, and Brian J.N. Wylie, A Callgraph-Based Search Strategy for Automated Performance Diagnosis, in Euro-Par 2000 (Munich, Germany, August 2000).

[20] Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gun Sirer, Marc E. Fiuczynski, David Becker, and Susan Chambers, Craig an Eggers, Extensibility, safety and performance in the spin operating system, In Proc. of the 15th ACM Symposium on Operating Systems Principles (SOSP '95), pages 267--284, December 1995.

[21] Lance Shuler, Rolf Riesen, Chu Jong, David van Dresser, Arthur B. Maccabe, Lee Ann Fisk, and T. Mack Stallcup. The Puma operating system for massively parallel computers. In Proceedings of the Intel Supercomputer Users' Group. 1995 Annual North America Users' Conference, June 1995.

[22] A. Veitch & N. Hutchinson: Kea - A Dynamically Extensible and Configurable Operating System Kernel, In Proceedings of the Third International Conference on Configurable Distributed Systems, IEEE Press, pp 236-242, May 1996.

[23] Doug Ghormley, David Petrou, Steven Rodrigues, and Thomas Anderson. SLIC: An Extensibility System for Commodity Operating Systems, USENIX Technical Conference 1998, June 1998.

[24] Antonio Augusto Frohlich and Wolfgang Schroder-Preikschat, High Performance Application-Oriented Operating Systems -- the EPOS Approach, In: Proceedings of the 11th Symposium on Computer Architecture and High Performance Computing, pages 3-9, Natal, Brazil, 1999

[25] Hulse,D. and Dearle,A, Trends in Operating Systems Design: Towards a Customizable Persistent Micro-kernel, Report Pastel RT1R4. University of Stirling 1998.

[26] Distinctive characteristics of scientific applications, the International Federation for Information Processing Report, Ottawa, Ontario, Canada, October 2000.

[27] Jeffrey S. Vetter, Frank Mueller: Communication characteristics of large-scale scientific applications for contemporary cluster architectures, J. Parallel Distrib. Comput. 63(9): 853-865 (2003)

[28] L. Carrington, A. Snaveley, N. Wolter, X. Gao, A Performance Prediction Framework for Scientific Applications, in Proc. Workshop on Performance Modeling and Analysis, ICCS, Melbourne, Australia, June 2003.

[29] J. Dike. A User-mode Port of the Linux Kernel. In Proceedings of the 4th Annual Linux Showcase & Conference, 2000.

[30] SPEC: Standard Performance Evaluation cooperation, <http://www.spec.org>.

[31] Rune Johan Andresen, Virtual Machine Monitors, CERN OpenLab for Data grid application, August 2004. Summer Reports.

[32] A. B. Montz, D. Mosberger, S. W. O'Malley, L. L. Peterson, T. A. Proebsting. Scout: A Communications-Oriented Operating System, Hot OS conference, May 1995

[33] Margo Seltzer, Christopher Small, Michael D. Smith, Symbiotic Systems Software, Workshop on Compiler Support for Systems Software (WCSSS '96)

[34] Hulse,D. and Dearle,A. Trends in Operating Systems Design: Towards a Customizable Persistent Micro-kernel, Report Pastel RT1R4. University of Stirling 1998.

[35] Seltzer, M., Endo, Y., Small, C., Smith, K., Issues in Extensible Operating Systems, Harvard University

Computer Science Technical Report TR-18-97,  
November 1997

[36] G. Nelson, System Programming in Modula-3,  
Prentice Hall, 1991.

[37] G. Denys, G. Piessens, and F. Matthijis, A Survey  
of Customizability in Operating Systems Research,  
ACM Computing Surveys (CSUR), Vol. 34, Issue 4,  
Pages 450 - 468, Dec., 2002.

[38] Ford, B., Back, G., Benson, G., Lepreau, J., Lin,  
A., and Shivers, O. 1997. The flux OSKit: A substrate  
for kernel and language research. In Proceedings of  
the 16th ACM Symposium on Operating Systems  
Principles. 38–51.

[39] Campbell, R., Islam, N., Raila, D., And Madany,  
P. 1993. Designing and implementing Choices: an  
object-oriented system in C++. Commun. ACM 36, 9,  
117–126.

[40] Christopher Small and Margo Seltzer, Structuring  
the Kernel as a Toolkit of Extensible, Reusable  
Components, Proceedings of the 1995 International  
Workshop on Object Orientation in Operating Systems  
(IWOOS '95).