Software Systems Seminar
Department of Computer Sciences
University of Salzburg

The K42 Operating System: A Research Context Survey

Thomas Aschauer

July 13, 2007

Abstract

K42 is a research project at IBM Research that explores operating system design by building a complete operating system kernel from the ground up. This survey identifies K42's key concepts and design decisions and gives an overview of related research systems and literature.

1 Introduction

Large-scope research projects in the field of operating systems have been rather rare the past few years. However, there are quite some challenging problems[1] in this field that would best be tackled by comprehensive research efforts [KAR$^+$06, p. 134], [HLTW05].

K42 Goals

The K42 project [KAR$^+$06], [SKW$^+$06] was initiated in 1996 to build a complete research operating system kernel without the burden of legacy requirements and design decisions, and whose design would support several policies and implementations simultaneously, extensibility for new policies and implementations, and good performance. In detail, the primary goals include:

- Considerable performance: K42 should scale well from small to large multiprocessor systems and enable good performance for small as well as for large applications.
- Customizability: Applications should be able to define how the operating system manages their resources. Moreover, automatic adjustment of management policies according to changing workload requirement should be possible.
- Applicability: K42 should be applicable to wide variety of applications and problem domains. Adaptability of the system to meet the requirements of new system architectures should be provided and K42 should support systems of any size.
- Availability: Availability to a large community of researchers and straightforward implementation of specialized components for experimentation purposes also are key factors for K42.

Based on a handful of technology predictions and and the project's goals, the K42 designers formulated a clear technological course to follow.

K42 Status Quo

As stated in [KAR$^+$06], the system now comprises a basic kernel infrastructure and an OS personality emulation layer, which supports the Linux API (*Application Programming Interface*) and the Linux ABI (*Application Binary Interface*). Large scale applications such as web servers, databases

[1] *"The products of forty years of OS research are sitting in everyone's desktop computer, cell phone, car, etc. — and it is not a pretty picture."* [HLTW05, p. 1]

and all kinds of benchmarks can be run on K42, which allows for a direct performance comparison to standard Linux distributions.

2 Key Concepts

The design decisions made by the K42 project team where, of course, driven by the requirements stated in section 1. The subsequent sections 2.1 to 2.4 shortly describe the basic principles they followed to achieve their goal. For every key concept a short introduction is given, then related research projects are listed and finally the application in K42 is described.

2.1 System Structure

Today, many commodity operating systems available still are implemented in a monolithic manner. This approach has the drawbacks of limited modularity, high complexity and high development and maintenance costs.

Micro-Kernel

The micro-kernel approach is a well-known alternative to monolithic implementations [TK95, p. 5], [SGG01, p. 79]. In a micro-kernel structured operating system, the kernel provides just an essential set of services like process management, memory management, IPC(*Interprocess Communication*), and device support. All other operating system functions are implemented in separate services that preferably run in user space. This separation increases the system's flexibility, and has the side effect that it is possible for the kernel to support different operating system personalities, either simultaneously or at different times [Cah96]. Moreover, modification of the servers is simpler that changing parts of a monolithic system, which improves flexibility.

However, this structure can lead to performance penalties through frequent context switches [TK95, p. 5], [DPM02, p. 457]. In addition, another drawback of monolithic operating systems still applies to micro-kernel systems: Abstractions of machine resources and their implementation are fixed.

According to [EKJO95, p. 251], the micro-kernel structure is not appropriate. Three main reasons are mentioned:

- Applications can not provide domain specific optimizations for their specific needs.
- Changing the implementation of existing abstraction is discouraged.
- New abstractions are hard to add and thus the flexibility of application developers is rather limited.

Exokernel

Taking the micro-kernel approach one step further leads to the so-called *exokernel* [DPM02, p. 457], [EKJO95]. In an exokernel operating system all system services run in user space and the small kernel just provides an abstraction of the underlying hardware resources. Untrusted *library operating systems*, running in user space, manage the resources by using the rather low-level kernel interface. Here the challenge is to provide library operating systems as much freedom as possible for the management task, but without undermining the protection from each other. Thus, three basic tasks have to be performed by an exokernel [EKJO95, p. 253]:

- Track the ownership of resources
- Ensure protection for bound resources
- Revoke access to resources

Ideally, the library operating systems allow for very specific optimizations of resource management policies, such that application designers can choose the right one for their particular problem domain.

K42

K42's structure design is guided by the exokernel approach. The kernel is structured around a client-server model, where most servers are implemented as user-level libraries. For example, thread scheduling is implemented in a library in user space [SKW^{+}06, p. 35], and also a fast signaling mechanism for interprocess communication is implemented in a user level library [KAR^{+}06, pp. 138-139]. This design allows applications to choose specific services that serve their very specific requirements as good as possible. Furthermore, the mapping to a certain OS personality[2] is done in user space to reduce overhead [KAR+06, p. 134].

2.2 Object-Oriented Design

Operating systems usually are large an inherently complex systems. The paradigm of object-orientation provides a methodology for managing complexity and thus improving understandability and maintainability. The well-known software engineering advantages of object-oriented design include, but are not limited to, encapsulation, reuse, flexibility and portability.

Using object-oriented design methods for building operating systems has become a trend in the early 1990s [TK95, p. 6, sec. 3.4]. However, [Cah96, pp. 7-8] distinguishes between operating systems that use object-oriented techniques for implementation purposes, and operating systems that are designed in an object-oriented fashion, such that the system provides its services in terms of system objects. Such designs result in systems that are highly customizable.

Choices

Choices is a research project exploring the applicability of object-orientation for building operating systems [CIRM93], [Tou05, pp. 7-8]. Similarly to K42, the OO paradigm is employed throughout the whole system, which means that hardware interfaces, application interfaces, system resources, mechanisms and policies are modeled as objects. Moreover, the system architecture comprises a number of frameworks for subsystem design [CIRM93, p. 117-118], [DCC+06, p. 45]. Choices is implemented in C++; some custom OO extensions are provided that the plain language is lacking (i.e. garbage collection, first-class classes[3], dynamic loading and advanced debugging support for classes and objects).

The Choices team reported that object-orientation can indeed have a positive impact on the design of operating systems and increases both, development productivity and the ease of innovative enhancements through code reuse and proper abstractions. The use of framework technology has an additional positive effect through architectural design-reuse [CIRM93, p. 125].

[Cah96, p. 27] lists Choices as probably the best know object-oriented research operating system, and as a highly influential one, since it showed the applicability of the OO paradigm along with its advantages without leading to noticeable performance degradation.

Spring

Spring is an object-oriented, distributed, multi-server microkernel system [HK93], which was developed at Sun Microsystems Laboratories. The major project goals of Spring were to support distributed applications in a highly secure environment.

In Spring, all system resources and operating system services are presented as objects in the OO-sense. A strong focus is on strong interfaces between operating systems components. Thus, these components can be treat as replaceable, substitutable parts. To define the interfaces, the interface description language *Spring IDL* is used. For structuring, the Spring developers chose to follow the microkernel approach; all user-mode services such as file systems, naming, paging, etc. are provided as dynamically loadable modules.

For distributed applications, Spring supports distributed objects by providing a secure and location transparent invocation mechanism [Cah96, pp. 24-27].

K42

One of the key design decisions of K42 was to use object-orientation design throughout the whole system [KAR+06, p. 134]. Two factors serve as rationale: First, structuring operating system data structures according to the object-oriented paradigm can enhance multiprocessor performance. This has already been shown by the tornado system [GKAS99], which is a direct predecessor of K42. Second, a promising research question is to examine the software engineering advantages of the OO-paradigm in the context of a large scale operating system project.

[2]The current implementation just supports the LINUX personality. However, K42 initially was designed to support multiple OS personalities.

[3]*First-class classes* means that classes themselves are represented by special objects at runtime. This is necessary for dynamic loading in an object-oriented language.

The project team reported that the object-oriented design had been a critical success factor for achieving the goals of scalability and customizability, and also is valuable for rapid prototyping and experimentation purposes. The maintainability issue has yet to be shown as a larger community contributes to the system. One potential drawback of the OO-design also has been identified: As all resources and services are modeled as dynamic interconnected objects, the static code paths become non-obvious. This, as a consequence, makes it harder for new developers to get a grip on some important implementation details [KAR+06].

2.3 Customizability

One of the primary goals of an operating system is efficient operation of the computer system [SGG01, p. 6]. In other words, the operating system should provide resources to application programs in an efficient and fair way [DPM02, p. 450]. For that reason, an operating system has to implement a multitude of policies for allocating and sharing resources like memory, processor time and communication facilities. In operating systems with a fixed set of built in policies the designers have to find compromises to support as many applications as possible with a reasonable performance. Naturally, this leads to suboptimal behavior for certain applications that do not match the criteria chosen by the policy designers. For example, traditional file systems and their buffering algorithms are not the optimal way to present storage to certain database applications [EKJO95, p. 252], [BSP+95, p. 267], [DPM02, p. 451]

A solution for this dilemma is to give applications the freedom to adjust the policies for their specific needs. This can be done in quite a wide variety of approaches. The surveys [DPM02] and [Tou05] both develop a set of classification criteria for extensibility in operating systems. The most important ones are the following:

- *Granularity and depth* specify the size of the units of customization and to which level the operating system can be adapted to specific needs.
- *Time* describes whether the customization can be performed at build-time, installation-time or run-time.
- *Integrity* deals with the determination of a validity of a customization task.
- *Initiation* tells whether the adaption is performed manually, autonomically by the application or automatically by the operating system.

An important design issue for customizable systems is performance: While application specific policies potentially lead to a better performance for certain applications, the customizability mechanisms are not free of charge. The overhead they may introduce can lead to performance penalties that even outweigh the original performance gain [DPM02, p.451]. However, customizability in operating systems has been subject to research investigations for quite some time, and it still is a hot research topic [HLTW05].

Exokernel

In the exokernel system, customizability is achieved by exchangeable user-level libraries. Due to the fact that the kernel exposes the hardware abstractions at a very low level to the library operating systems, an application can choose an implementation that best suits its needs. However, customization still is limited to build-time [EKJO95],[Tou05, pp. 9-10].

Choices

The Choices system models all operating system concepts as objects, thus they can be specialized and it allows applications to dynamically load new services into the kernel. Thus, an application can customize the system at runtime, but with the restriction that possible service requests have to be known in advance at compile-time [CIRM93], [DPM02, p. 456] ,[Tou05, pp. 7-8].

SPIN

SPIN is a customizable operating system written in Modula-3. Applications can provide so-called *extensions* to the the kernel that are invoked on certain events. A primary goal of SPIN is to provide a safe extension mechanism. The mechanism's implementation is tight bound to Modula-3, as language features such as type-safety, interface boundaries and enforced modularity are employed for safety checks. This approach, as the authors of [BSP+95] argue, is promising because it provides a reasonable fast extension mechanism without compromising performance. However, the safety policy is defined by the semantics of the programming language, and the extension model does not define how extension can be removed [DPM02, p. 463] [Tou05, p. 17].

K42

As mentioned in the introduction, customizability was a primary goal of the K42 design. In K42, the key design idea is to represent every single resource instance by a different set of one or more object instances. As these objects manage the given resource, replacing them by specialized object instances allows applications to perform very fine grained customizations. This granularity makes it possible, for example, that the same application sets up two different page replacement policies for two distinct files. Moreover, as different usage patterns for these two files may apply, the page replacement policies may even be adapted on the fly to reflect changing workload demands. The mechanism for dynamically replacing an object instance by another one is referred to as *hot swapping* in K42 terminology. The question of how to determine whether a given configuration is safe or not is not subject to investigation in this system; safety is defined by interface signature compliance.

Reported performance measurements show that the approach can lead to performance gains for several standard benchmark suites [SAH+03, p.2], [BHA+05].

2.4 Object Distribution

Using object-oriented techniques for developing distributed applications is faced with the problem that the objects may be located at different computation nodes. The technique of remote method invocation deals with these aspects. The Spring system [HK93], for example, provides a secure mechanism for location transparent invocation.

Another approach is to design objects such that they are partitioned or replicated to reside locally on each of the nodes they are to be invoked on. The idea is that logically there exists just one object in the system. Its implementation (or parts of it) physically are distributed on several nodes. The representative objects on these nodes represent themselves as if they were just this one object. Thus, the designers of the representatives can take advantage of the awareness of distribution to provide efficient implementations that, for example, limit the amount of communication needed, while the distribution is transparent to the clients.

Fragmented Objects

In [MGNS94] the authors propose a uniform concept for designing distributed object-oriented application called *Fragmented Objects*. The mechanism for distributing objects is used for reasons of performance, availability, protection and load balancing.

K42

K42 introduces the term *Clustered objects* for a similar mechanism [KAR+06, p. 135, pp. 137-138], [GKAS99] It is presented as a way to scalable implementations of concurrently accessed objects in a distributed environment using distribution, replication and partitioning.

Clustered objects are used heavily throughout the whole operating system. This enables a system service of be distributed among different processors of a system, which in the context of NUMA (*non-uniform memory access*) systems can maximize locality. The project team reports that this approach can lead to significantly improved system performance [AAS+03]. Moreover, clustered object are key building blocks for supporting customizability.

3 Conclusion

This survey presents an overview of the research context of K42. Key construction principles of the system are identified. They are described briefly and references to other research operating systems employing similar principles are given.

References

[AAS+03] J. Appavoo, M. Auslander, D. Silva, O. Krieger, M. Ostrowski, B. Rosenburg, R. Wisniewski, J. Xenidis, M. Stumm, B. Gamsa, R. Azimi,

R. Fingas, A. Tam, and D. Tam. Enabling scalable performance for general purpose workloads on shared memory multiprocessors. Technical Report IBM Research Report RC22863, IBM Research, 2003.

[BHA+05] Andrew Baumann, Gernot Heiser, Jonathan Appavoo, Dilma Da Silva, Orran Krieger, Robert W. Wisniewski, and Jeremy Kerr. Providing dynamic update in an operating system. In *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference*, pages 32–32, Berkeley, CA, USA, 2005. USENIX Association.

[BSP+95] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility safety and performance in the spin operating system. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 267–283, New York, NY, USA, 1995. ACM Press.

[Cah96] Vinny Cahill. Flexibility in object-oriented operating systems: A review. Technical Report TCD-CS-96-05, University of Bologna, 1996.

[CIRM93] Roy H. Campbell, Nayeem Islam, David Raila, and Peter Madany. Designing and implementing choices: an object-oriented system in c++. *Communications of the ACM*, 36(9):117–126, 1993.

[DCC+06] Francis M. David, Jeffrey C. Carlyle, Ellick M. Chan, David K. Raila, and Roy H. Campbell. Exception Handling in the Choices Operating System. In C. Dony, J. L. Knudsen, A. Romanovsky, and A. Tripathi, editors, *Advanced Topics in Exception Handling Techniques*, volume 4119 of *Lecture Notes in Computer Science*, pages 42–61. Springer-Verlag Inc., New York, NY, USA, 2006.

[DPM02] G. Denys, F. Piessens, and F. Matthijs. A survey of customizability in operating systems research. *ACM Computing Surveys*, 34(4):450–468, 2002.

[EKJO95] D. R. Engler, M. F. Kaashoek, and Jr. J. O'Toole. Exokernel: an operating system architecture for application-level resource management. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 251–266, New York, NY, USA, 1995. ACM Press.

[GKAS99] Ben Gamsa, Orran Krieger, Jonathan Appavoo, and Michael Stumm. Tornado: maximizing locality and concurrency in a shared memory multiprocessor operating system. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 87–100, Berkeley, CA, USA, 1999. USENIX Association.

[HK93] G. Hamilton and P. Kougiouris. The spring nucleus: A microkernel for objects. In *Proc. of the Summer 1993 USENIX Conference*, pages 147–159, Cincinnati, OH, 1993.

[HLTW05] Galen C. Hunt, James R. Larus, David Tarditi, and Ted Wobber. Broad new os research: challenges and opportunities. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, Berkeley, CA, USA, 2005. USENIX Association.

[KAR+06] Orran Krieger, Marc A. Auslander, Bryan S. Rosenburg, Robert W. Wisniewski, Jimi Xenidis, Dilma Da Silva, Michal Ostrowski, Jonathan Appavoo, Maria A. Butrico, Mark F. Mergen, Amos Waterland, and Volkmar Uhlig. K42: building a complete operating system. In *Proceedings of the 2006 EuroSys Conference, Leuven, Belgium*, pages 133–145, 2006.

[MGNS94] Mesaac Makpangou, Yvon Gourhant, Jean-Pierre Le Narzul, and Marc

Shapiro. Fragmented objects for distributed abstractions. In T. L. Casavant and Singhal M., editors, *Readings in distributed computing systems*, pages 170–186. IEEE Computer Society Press, 1994.

[SAH$^+$03] Craig A. N. Soules, Jonathan Appavoo, Kevin Hui, Robert W. Wisniewski, Dilma Da Silva, Gregory R. Ganger, Orran Krieger, Michael Stumm, Marc A. Auslander, Michal Ostrowski, Bryan S. Rosenburg, and Jimi Xenidis. System support for online reconfiguration. In *USENIX Annual Technical Conference, General Track*, pages 141–154, 2003.

[SGG01] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[SKW$^+$06] Dilma Da Silva, Orran Krieger, Robert W. Wisniewski, Amos Waterland, David Tam, and Andrew Baumann. K42: an infrastructure for operating system research. *SIGOPS Operating Systems Review*, 40(2):34–42, 2006.

[TK95] Anand R. Tripathi and Neeran M. Karnik. Trends in multiprocessor and distributed operating systems designs. *The Journal of Supercomputing*, 9(1-2):23–49, 1995.

[Tou05] Jean-Charles Tournier. A survey of configurable operating systems. Technical Report TR-CS-2005-43, Department of Computer Science, University of New Mexico, November 2005.