COMP30640: Assignment 1 - Research Reflection - Exokernel

Adam Ryan - 14395076

December 21, 2020

1 Abstract

In this report, the author reflects on the paper of Engler et al [1] in which they describe an operating system (OS) kernel architecture in which, contrary to traditional kernel systems, applications can interface more directly with hardware. The author provides a high-level relfection on this paper, examining it from the perspective of the categories outlined in the COMP30640 Research Reflection Requirement document.

Contents

2 Research Motivation

As we have seen within COMP30640, traditional kernel architectures involves a layer of abstraction in which applications must interface with hardware by acting through the kernel via system calls. In a monolithic kernel architecture, all aspects of the operating system are contained in the kernel, in a microkernel a small core OS is operating in the kernel which is kept small and features are minimised with some traditional kernel services kept as trusted user-level modules whereby interactions between these modules occur via message-passing through the kernel, and modular kernels in which you have a core kernel with a variety of dynamically loadable modules. Regardless of the kernel type, Engler et al. describe how the abstraction provided by traditional kernel architectures introduces a number of generalised problems, which their exokernel architecture is designed to address.

A key motivating factor in the development of exokernels is that Engler et al. outline how traditional OS-architecture attempts to satisfy the needs of all applications. As not all applications require each of the features an operating system may provide, and as the OS abstraction must be generalised (rather than class-specific) to account for a variety of applications, there is a performance hit to applications, there may be an increase in the application complexity by needing to operate through these generalised abstractions, and there may be a reduction in an application's functionality.

They outline in their paper how some classes of applications, such as databases and garbage collector processes, have predictable data access patterns and generalised page replacement strategies, rather than application-dictated strategies, can significantly degrade performance, and it has been shown that application-controlled caching can increase performance by as much as 45%. As information gets hidden from an application via the abstraction an OS provides, such as the hiding of page faults, there is an increased difficulty in designing applications with suitable performance as the application cannot be designed to take advantage of this information. Finally, as all applications share the same abstractions to access hardware resources it is rare that the abstractions change which means that recent features researched rarely make their way into operating systems, limiting application functionality as these abstractions are not regularly expanded upon due to the risk this may present.

3 Proposed Solution

In this paper, the authors attempt to address the challenges outlined in 2; they design a kernel architecture in which the traditional kernel abstraction happens entirely at an application level, so that traditional kernel features can be expanded upon at an application level, and interaction with the hardware occurs through low-level primitives within the kernel which allows for applications to leverage custom abstractions specifically designed for that application. Engler et al. detail a broad overview of the design principles of an exokernel, and then describe their implementation of an exokernel and OS following these principles - Aegis and ExOS, which serve as a prototype exokernel and operating system, and demonstrate the advantages this system may provide via benchmarking.

At a high level, the first component of the design of an exokernel is allowing a library operating system (LOS) to access the hardware components as directly as possible without abstracting the resources provided by the underlying hardware, and should not manage resources outside of what's required for safety such as resource ownership or revocation. The LOS should manage all page allocation and this should be exposed to applications with the underlying structures directly accessible to allow developers to tailor allocation requests. The exokernel needs to handle how resources are allocated and revoked (similar to a traditional kernel even though the LOS manages the resources) and protects resources. To do this, 'secure bindings' are used which involve simple (for the OS) protection checks and allow authorisation at access time. The authors of the paper implement secure bindings using a combination of hardware, caching techniques, and downloading code into the kernel. The implementation of secure bindings and the guiding principles of the exokernel are what drive the design of Aegis and ExOS.

4 Methodology

Using the principles described in 3, the authors designed a prototype exokernel and library operating system called Aegis and ExOS respectively.

Using these implementations, they compare the functionality to that of Ultrix, a monolithic system which performs strongly compared to other OS on identical hardware to evaluate the benefit of an OS with an exokernal implementation. One caveat the authors include is that their OS does not offer functionality equivalent to Ultrix, but conclude the additional overhead of this functionality is not anticipated to add significant overhead (the author of this paper is skeptical). The authors provide detail on the implementation of Aegis and ExOS (e.g. that Aegis does scheduling via round robin through time slices) and then detail benchmarking performances. In particular, the authors compare how Ultrix performs in procedure cals and system calls, and how exception dispatches are performed with Ultrix performing significantly lower in base operations compared to Aegis. The authors attempt to compare the performance of transfer protocols with equivalent operations on L3, but scale the results according to the hardware in an attempt for a comparison (although Aegis performs over 6 times faster in this comparison it is not a direct like-for-like comparison between the two and to this author seems like a tenuous claim)

Following the comparisons of Ultrix and Aegis, the authors compare Ultrix with ExOS. In particular, it focuses on how OS abstractions can be implemented at an application level with a particular focus on IPS, virtual memory, and communication. To compare IPC, the authors examined the latency of sending messages between processes, looked at the time for switching processes using one counter, and finally measuring an LRPC into a different address space. When looking at IPS, Ultrix does not, in all cases, have a direct comparison with the ExOS implementations. however in those which are comparable ExOS displays a significant benefit. When examining virutal memory, the authors use the multiplication of $A, B \in M_{150}(\mathrm{Z})$ which does not leverage the specific benefits of ExOS, and indeed ExOS performs comparable to Ultrix or slightly worse in these specific tests; the authors comment that this suggests application-level virtual memory does not add noticeable overheap to operations involving a VM footprint but caution that the experiment is limited. When comparing Ultrix to ExOS on other VM benchmarks, the results are similar with Ultrix out-performing ExOS or ExOS largely perfoming comparably (the benchmarking measures denoted prot100 and unprot100 are notable exceptions). The authors note that more sophisticated developments rather than the authors' prototype implementation could result in further optimisations

As a final point of comparison, the authors examine Application-specific Safe Handlers (ASH) to examine how and why ExOS downloads code into the kernel and the impact of this operation on performance. ASH are untrusted application handlers which are downloaded into the kernel. Results of ExOS, Ultrix, and FRPS are compared by examining the roundtrip latency over ethernet on certain hardware configurations. Using round-robin for process scheduling, without ASH performance grew linearly while with ASH, performance remained relatively stable as the number of active processes increased, while with Ultrix the latency ranged much more depending on how many active processes were running which the authors use to conclude that decoupling actions resulted in a significant performance benefit.

As a final demonstration of the design of their exokernel prototype, the authors demonstrate how the implementation of special purpose implementations of processes to meet the requirements of specific applications can provide ExOS with various dramatic benchmark improvements while simultaneously resting on the same machine.

5 Results

Following on from the benchmarking and general overview of results described in 4, the authors ultimately conclude that the primitives in an exokernel can be implemented far more efficiently than traditional primitive implementation (in partiuclar focusing on Ultrix since that was the authors' point of comparison), multiplexing can be efficiently implemented, that the design of an exokernel and LOS is valid with the ability to efficiently implement traditional OS operations at application level, and that applications implement specialised abstractions by modifying libraries to achieve substantial improvements in both functionality and performance.

Ultimately, based on the results and tests conducted, the author's conclude that a exokernel is a viable OS structure with the potential for significant performance increases versus traditional OS architecture.

6 Author Reflection

The structure of an exokernel is one which is interesting to the author, and is why this paper was ultimately chosen. The author particularly liked how a variety of components of the design of an exokernel were tested, and how benchmarking performances were included within the paper. Engler et al. were clear in the paper's structure and gave detail on why certain design decisions were made which were then directly tested within their methodology. The performance was given in context with the comparison to Ultrix which grounded many of the claims made regarding performance increases and demonstrating the potential viability of a production-level exokernel and how this could provide significant performance increases.

One aspect which the author is slightly skeptical on is two claims which were made in the paper. The first of these is that Engler et al. highlight that Ultrix has significantly more functionality than their prototype ExOS but that they don't expect additional features to add much overhead. The author is slightly skeptical as to this claim, both in terms of performance and significantly in terms of the development overhead which would be added (although this is technically beyond the scope of the paper, one aspect which stands out to the author is how much additional development may be required in application development to make use of some of the performance increases which might be gained by an exokernel; this also seems like an aspect which could ultimately result in significant bloat). Even though the author's focus on the performance benefits, if these abstractions need to ultimately be created at an application level not only will they ultimately still become present, but it will add (as touched upon) a greater level of complexity in application development.

The second of these is that while the performance gains are focused on significantly, other aspects of operating systems are not firmly addressed in the paper. In particular, the authors focus heavily on how a low level of abstraction between hardware and software can bring significant performance benefits to applications, however while this may be true the low-level design and highly customised implementation would have a significant effect on scalability and the ability to port software to alternative (and potentially more powerful) hardware systems. While the authors demonstrate how Aegis and ExOS outperform stronger systems, if the software cannot easily be ported or transferred to new hardware these performance benefits may eventually be overtaken and leave this design unsuitable for a production OS.

It isn't clear to the author how either of these two aspects are addressed by Engler et al. Although an exokernel may be a viable kernel architecture, these two aspects seem to be significant barriers to the author on why such a kernel architecture, even with the performance benefits Engler et al. detail, may not be as superior as the benchmarking results appear to portray. The paper heavily focuses on the theoretical design of such an OS and the prototyping of the viability of this design rather than the practical implementation, and while the design is highly interesting and could potentially be useful for highly bespoke systems where performance is crucial, the issues of portability and the various application development problems (e.g. bloat of the LOS over time via multiple specific implementations of what is largely the same process, application development complexity, etc.) seem like they would need to be addressed before concluding this is a viable commercial OS architecture rather than one design for highly bespoke or niche systems where development time and hardware scalability isn't as key a concern.

References

[1] D. R. Engler, M. F. Kaashoek, and J. O'Toole. Exokernel: An operating system architecture for application-level resource management. *SIGOPS Oper. Syst. Rev.*, 29(5):251–266, December 1995.