Exokernel: An Operating System Architecture for Application Level Resource Management.

Aaryan Bhagat

Summary

The Paper rests on an idea that a kernel designed to do minimal work as possible is safe, reliable, efficient. It calls this kernel as exokernel and proposes that it will only act as a gateway for applications to have direct secure access to the hardware. Upon searching this paper I found the publishing date 1995 and that does make a lot of sense because as of today where abstractions are the most dominant form of writing code nobody will like this idea of a bare minimum kernel (maybe few exceptions).
For this they plan to expose the hardware resource through a secure connection, let the application participate in the resource allocation and then have a protocol for a secure release of that connection. The authors want to send multiple types of data through these connections hence they have implemented a hardware multiplexer. We can easily see that the efficiency is increased, a lot, the experiment section is a testament to that. Application level code here will have a lot of freedom and can optimize themselves accordingly. In addition to this, this kernel has the facility to directly run a new code (downloaded code) in it and has been recommended to create packet filters. Experimentation is done based upon their microkernel (aegis) and an application side OS used (ExOS, library operating system). Comparison is done with Ultrix.

Pros

- Efficient (fast) as minimum support the kernel has to provide, the philosophy they embibe is “no/minimal management of resources only providing resources”.
- Reliable code as functionalities are minimum so you can write simple code.
- This much handling at application level means no kernel crossings so no wasting time in context switchings.
- Applications can write handlers which are application specific and will run directly in the kernel.
- Lot of flexibility is given to the application side as they have to write every kind of resource management, all other kinds of abstractions like IPC, RPC by themselves, hence they can optimize their code from quite low level.

Cons

- Lot of expose to 3rd party applications, no one expected to write this much code, if let's say you want to install some software which analyzes packets (wireshark for e.g), you will have to find another software which has done all the necessary abstractions (IPC, gRPC, packet filter), install it first then write code which will be compatible to that software. Will involve a lot of maintenance on your part. Things can break easily.
- Although the kernel has provided basic facilities to safeguard memory resources it is still not enough because every form of abstraction is left to the application side, for e.g in case of Application Safe Handlers, (or any other download code) which is an untrusted 3rd party running directly in kernel mode. What safeguards they have adopted is not clear, they have only cited 2 methods, code checking and sandboxing but I am not sure at all how sandboxing will be implemented in such a primitive kernel. So all in all, this remains unclear.

Other Comments

I understand the POV of this paper of that time, they wanted to extensively work on the efficiency and did not realize what level of software building will happen, ideas like this cannot be implemented today where we are forced to use layers and layers of abstractions for any minute software we write and cannot build everything from scratch and also where attacks on systems have gotten so complex that giving application this level of control to the hardware will be alarming. Only in some cases where the overall goal is very specific can be worked out. For e.g if someone wanted to make an alternative to QubesOS (where every application is a container, an OS extensively build for security) he can definitely start with this exokernel as a base.