An I/O Architecture for Microkernel-Based Operating Systems

Hermann Haertig Adam Lackorzynski Jork Loeser Frank Mehnert
Martin Pohlack Lars Reuther Alexander Warg

Dresden University of Technology

Motivation It becomes widely accepted that off-the-shelf operating systems, such as Unix-derivatives or versions of Microsoft Windows, are way too complex for guaranteeing real-time or trustworthiness properties. However, applications emerge that require standard operating system's functionality enhanced by the aforementioned properties. Imagine smartphones used for online banking, playing games downloaded from the Internet, and watching video streams. As of today, separation is the only practicable way to handle such complex systems. We show that microkernels provide the appropriate abstractions to enforce this separation efficiently. Microkernel-based separation allows us to run a slightly modified version of Linux concurrently with real-time and trustworthy applications.
Instead of rebuilding such systems from scratch, standard operating system's functionality is provided by L^{4}Linux[2] — a version of Linux running as a user-mode application atop the L4 microkernel. Trustworthy and real-time functionality is added through separate servers running next to L^{4}Linux. Standard applications run unmodified and adaptions are solely necessary to make use of special services.
Microkernel-based separation has several advantages:

- Typically only small parts of an application do require specific properties, and only these parts need to be implemented separately. The large base of Linux legacy applications can be reused without recompilation. Newly-written applications can be kept small.
- Because the specialized systems are small, it is easier to verify the properties of those systems.
- Given a proper encapsulation, the real-time subsystem and the trustworthy applications can reuse both — parts of the running L^{4}Linux and separated components of Linux (e.g. SCSI subsystem).

In our architecture, all components — including device drivers [1] — run as user-mode tasks atop of the L4 microkernel. Obviously, this increases the robustness of the system. Separating components from other parts of the system causes challenges, though. We discuss them in our work, among them the following:

Challenges Encapsulation of untrusted components requires:
Address spaces are used to mutually protect the memory of components. Given this, components can no longer access data of other components directly. Data must be transferred by other means to overcome address-space boundaries, e.g. microkernel IPC or shared memory.
Device register access is restricted by standard virtual memory techniques.

[1] Device drivers are known to be the most error-prone parts of operating systems [1]

Interrupts are managed at a central instance that propagates interrupt events to registered clients. Interrupt sharing enables denial-of-service attacks between drivers. To isolate those drivers, different interrupt lines have to be attached to different devices.
Disabling interrupts — if not controlled — enables malicious drivers to monopolize the system. It is known to be replaceable by appropriate locking mechanisms. Disabling the interrupts to handle devices is no more used in modern drivers. Consequently we can avoid disabling interrupts completely.
Bus-master DMA is a feature of modern busses such as PCI allowing arbitrary physical memory access. To restrict bus-master DMA, we apply the principle of address spaces to the I/O bus. An I/O-MMU implementing a per-device address space would allow for full isolation of each device. Unfortunately, we are not aware of such hardware being implemented. While systems with one I/O-MMU for all devices are available today, they are not capable to isolate device and driver. Instead, we propose to use virtual-machine-monitor techniques as long as I/O-MMUs are not available to emulate the I/O-MMU's functionality in software.

With the above approach, untrusted driver code is reused to share devices between trustworthy and other applications. For ensuring the confidentiality and integrity of trusted applications data, we use tunneling and cryptographic techniques in the secure subsystem. To additionally ensure availability of the data, known techniques such as redundancy can be employed.

Conclusion We presented an architecture to construct trustworthy systems and real-time systems. For this, we applied microkernel-based separation. The resulting systems offer standard operating systems functionality by reusing a modified version of Linux. Additional functionality is added by specific servers. We also discussed some of the challenges resulting from this separation, among them a hardware extension required by any trustworthy system allowing bus-master DMA.
We claim our approach to be efficient both in development time and in execution time.

References

[1] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating systems errors. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 73–88. ACM Press, 2001.

[2] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter. The performance of μ-kernel-based systems. In *16th ACM Symposium on Operating System Principles (SOSP)*, pages 66–77, Saint-Malo, France, October 1997.

[3] H. Härtig, J. Löser, F. Mehnert, L. Reuther, M. Pohlack, and A. Warg. An I/O Architecture for Microkernel-Based Operating Systems. Technical Report TUD-FI03-08-Juli-2003, Dresden University of Technology, Dresden, Germany, July 2003.