1st EuroSys Doctoral Workshop, Brighton, England, Oct. 2005.

DESIGN AND IMPLEMENTATION OF A RELIABLE OPERATING SYSTEM

Jorrit N. Herder <jnherder@cs.vu.nl>
Andrew S. Tanenbaum <ast@cs.vu.nl>
Herbert Bos <herbertb@cs.vu.nl>

INTRODUCTION

An operating system (OS) forms the software foundation of computers and therefore should be completely reliable. Unfortunately, today's monolithic OSes, such as Windows and Linux, fail to deliver this ideal because they suffer from fundamental design flaws. Their kernel is overloaded with functionality that runs at the highest privilege level, proper fault-isolation is lacking, and foreign code is tolerated within the kernel. This breaches the Principle of Least Authorization (POLA), with all the related risks. A malfunctioning third-party device driver, for example, can easily bring down the entire system.

The goal of this research is to build a reliable OS by exploiting the modularity of a multiserver scheme on top of a tiny microkernel. By splitting the OS into small, independent parts, we can establish 'firewalls' across which errors cannot propagate, thus resulting in a more robust system.

CONTRIBUTIONS

We took a hybrid system with in-kernel device drivers and transformed it into a POSIX-compliant, multiserver OS with a true microkernel. Only a minimal set of abstractions (e.g., passing of fixed-size messages) are now implemented in kernel-mode. The rest of the OS is distributed over user-space servers. With fewer than 4,000 lines of code, the kernel is easy to understand and conceivably might even be proven formally correct eventually.

Each server and each device driver runs as a separate user-mode process. Important servers include the process manager and file system(s). We have also implemented user-space drivers for IDE, floppy and RAM disks, consoles, keyboards, printers, and various Ethernet cards. Extensive measurements have shown that the performance penalty introduced by our changes is 5 to 10%.

In our system, device driver processes are highly restricted in what they can do. Each one can access only those I/O ports and kernel functions for which it has permission, for example. Communication with unrelated processes is disallowed. Furthermore, driver bugs such as bad pointers or infinite loops can affect only that driver. It is possible to kill a malfunctioning driver and restart it without affecting the rest of the system. In this way, the damage that driver bugs can do is greatly limited. In any event, they cannot bring down the entire OS.

RELATED WORK

The L4 microkernel is a well-known contender, but does not focus on the reliability of an entire OS. L4Linux is a single-server OS for L4, but has similar problems as monolithic systems. DROPS is more related to our work, but targeted towards realtime systems. SawMill Linux is another attempt to create a multiserver OS, but, unfortunately, this project has been terminated.

Recently, the idea of running device drivers in a virtual machine within the kernel was proposed. Although this helps to limit the potential damage caused by drivers, it introduces complexity and possibly adds bugs to the kernel. Moreover, the proper mechanism to run untrusted code has been the process model for decades. We strongly believe that this still holds.