Chapter 2
Device Driver Reliability

Michael M. Swift
University of Wisconsin—Madison, USA

ABSTRACT

Despite decades of research in extensible operating system technology, extensions such as device drivers remain a significant cause of system failures. In Windows XP, for example, drivers account for 85% of recently reported failures. This chapter presents Nooks, a layered architecture for tolerating the failure of drivers within existing operating system kernels. The design consists techniques for isolating drivers from the kernel and for recovering from their failure. Nooks isolates drivers from the kernel in a lightweight kernel protection domain, a new protection mechanism. By executing drivers within a domain, the kernel is protected from their failure and cannot be corrupted. Shadow drivers recover from device driver failures. Based on a replica of the driver’s state machine, a shadow driver conceals the driver’s failure from applications and restores the driver’s internal state to a point where it can process requests as if it had never failed. Thus, the entire failure and recovery is transparent to applications.

INTRODUCTION

Improving reliability is one of the greatest challenges for commodity operating systems, such as Windows and Linux. System failures are commonplace and costly across all domains: in the home, in the server room, and in embedded systems, where the existence of the OS itself is invisible. At the low end, failures lead to user frustration and lost sales. At the high end, an hour of downtime from a system failure can lead to losses in the millions.

Computer system reliability remains a crucial but unsolved problem. This problem has been exacerbated by the adoption of commodity operating systems, designed for best-effort operation, in environments that require high availability. While the cost of high-performance computing continues to drop because of commoditization, the cost of failures (e.g., downtime on a stock exchange or e-commerce server, or the manpower required to service a help-
desk request in an office environment) continues to rise as our dependence on computers grows. In addition, the growing sector of “unmanaged” systems, such as digital appliances and consumer devices based on commodity hardware and software, amplifies the need for reliability.

Device drivers are a leading cause of operating system failure. Device drivers and other extensions have become increasingly prevalent in commodity systems such as Linux (where they are called modules) and Windows (where they are called drivers). Extensions are optional components that reside in the kernel address space and typically communicate with the kernel through published interfaces. Drivers now account for over 70% of Linux kernel code, and over 35,000 different drivers with over 112,000 versions exist on Windows XP desktops. Unfortunately, most of the programmers writing drivers work for independent hardware vendors and have significantly less experience in kernel organization and programming than the programmers that build the operating system itself.

In Windows XP, for example, drivers cause 85% of reported failures. In Linux, the frequency of coding errors is up to seven times higher for device drivers than for the rest of the kernel. While the core operating system kernel can reach high levels of reliability because of longevity and repeated testing, the extended operating system cannot be tested completely. With tens of thousands of drivers, operating system vendors cannot even identify them all, let alone test all possible combinations used in the marketplace. In contemporary systems, any fault in a driver can corrupt vital kernel data, causing the system to crash.

This chapter presents Nooks, a driver reliability subsystem that allows existing device drivers to execute safely in commodity kernels (Swift, Bershad & Levy, 2005). Nooks acts as a layer between drivers and the kernel and provides two key services: isolation and recovery. Nooks allows the operating system to tolerate driver failures by isolating the OS from device drivers. With Nooks, a bug in a driver cannot corrupt or otherwise harm the operating system. Nooks contains driver failures with a new isolation mechanism, called a lightweight kernel protection domain, that is a privileged kernel-mode environment with restricted write access to kernel memory.

When a driver failure occurs, Nooks detects the failure with a combination of hardware and software checks and triggers automatic recovery. A new kernel agent, called a shadow driver, conceals a driver’s failure from its clients while recovering from the failure (Swift et al, 2006). During normal operation, the shadow tracks the state of the real driver by monitoring all communication between the kernel and the driver. When a failure occurs, the shadow inserts itself temporarily in place of the failed driver, servicing requests on its behalf. While shielding the kernel and applications from the failure, the shadow driver restarts the failed driver and restores it to a state where it can resume processing requests as if it had never failed.

**DEVICE DRIVER OVERVIEW**

A device driver is a kernel-mode software component that provides an interface between the OS and a hardware device. In most commodity operating systems, device drivers execute in the kernel for two reasons. First, they require privileged access to hardware, such as the ability to handle interrupts, which is only available in the kernel. Second, they require high performance, which is achieved via direct procedure calls into and out of the kernel.

**Driver Software Structure**

A driver converts requests from the kernel into requests to the hardware. Drivers rely on two interfaces: the interface that drivers export to the kernel, which provides access to the device, and the kernel interface that drivers import from the
Related Content

Composition of Optimal Service Workflows with Quality-of-Service Enabled Multi-Criteria Uniform Cost Search Algorithm

Improving the Bluetooth Hopping Sequence for Better Security in IoT Devices
www.igi-global.com/article/improving-the-bluetooth-hopping-sequence-for-better-security-in-iot-devices/210459?camid=4v1a

Teaching Agile Software Development Quality Assurance
www.igi-global.com/chapter/teaching-agile-software-development-quality/29529?camid=4v1a

A Revised EIIO Model and a Simple Software Review Guide
www.igi-global.com/chapter/revised-eiio-model-simple-software/26906?camid=4v1a