NOOKS

Improving the reliability of commodity operating systems

Mike Swift, Steve Martin, Doug Buxton, Leo Shum, Mirco Stern
Hank Levy, Brian Bershad

http://www.cs.washington.edu/homes/mikesw/nooks

Problem

- **Reliability is the critical problem for commodity operating systems**
  - Linux, Windows XP ubiquitous in data center,home, office, and appliances.
- **Existing reliability solutions have not transferred**
  - Require rewrite of OS kernel and all extensions

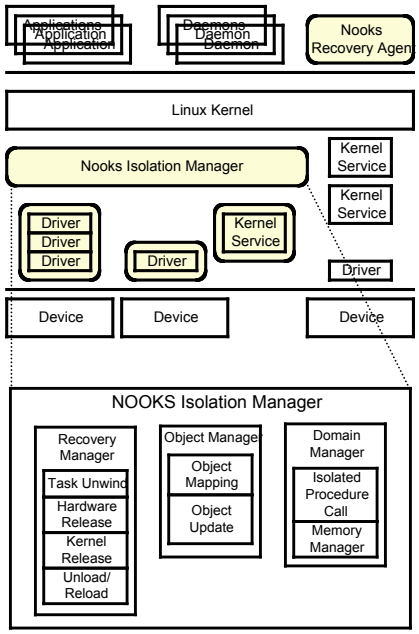
Principles

- **Best effort, but support the rest**
  - Don't try to prevent every fault
  - Don't try to support every extension
- **Design for fault resistence, not fault tolerance**
  - We are interested in reliability, not security.

Goals

- **Isolation**
  - Prevent extensions from causing the operating system to crash
- **Recovery**
  - Restart crashed extensions automatically
- **Efficiency**
  - Impose a minimum performance penalty
- **Backward Compatibility**
  - Support existing extensions with no code changes
  - Integrate into existing operating systems with few changes

Architecture



High Level Architecture

- Isolate device extensions with in a virtual memory protection domain
- Use interposition to add parameter checks and protection domain change to kernel-extension interface
- Fault model
  - Crashing faults: causes OS to stop functioning
  - Functional faults: extension doesn't perform correctly
  - Goal: prevent or recover from a large percentage of crashing faults

Architecture Details

- **Wrappers**
  - Interposed functions between kernel and extension
  - Responsible for validating parameters to kernel and data transfer between protection domains
- **Domain Manager**
  - Manages memory isolation with separate page table per protection domain
  - Transfers control between domains by changing processor page table and swapping stack
- **Resource Manager**
  - Maintains table of kernel objects in use by extensions
  - Maintains shadow copies of writeable objects for extensions
  - Maintains table of extension functions callable from kernel
- **Error handling**
  - Errors from extension occur at:
    - Memory instructions: triggers restart of extension (can't continue)
    - Calls to/from kernel: reflected as error codes returned to extension or kernel
- **Recovery Manager**
  - Unwinds executing tasks
  - Releases kernel resources (from resource manager)
  - Unregisters extension functions from kernel
  - Reloads extension
  - Releases physical resources

Experience

- **Implementation**
  - Linux 2.4.10
  - Interposition through module load
  - Memory isolation with page tables
  - Fault detection with exception handlers
- **Experience**
  - Isolated several kernel components
    - Network interface device drivers
    - VFAT File system
    - KHTTP Web server
  - Found bugs in extensions during development
    - 3c90x driver overwrites memory after freeing
    - KHTTPD web server double-release kernel socket

Code Statistics

Kernel Functions Wrapped	257
Extension functions wrapped	123
Kernel source files changed	.h 36 .c 22

Recovery	328
Domain Management	1052
Resource Management	811
Wrappers	5240
Miscellaneous	840
Total	8271

- **Lessons learned**
  - What makes isolation easier?
    - Enforce data hiding
    - Enforce regular calling conventions
    - Procedural, not macro, interfaces
    - Kernel allocated objects
    - No parameter shadowing
  - What extensions are easiest/cheapest to isolate?
    - Device drivers: simplest parameters