Shadow Drivers

Mike Swift, Muthukaruppan Annamali
Hank Levy, Brian Bershad

Transparent Recovery for Kernel Extensions

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

Problem

- **Computers crash too often.**
- **We can improve the reliability of computer systems by improving the reliability of the operating system**
  - Most OS crashes due to bugs in extensions
- **OS needs to provide**
  - Isolation for extensions
    - see SOSP'03 paper
  - Transparent recovery after a fault

Example

- **Sound driver crashes while playing music**
- **Today:** machine panics, all work is lost
- **Goal:** OS recovers sound driver while MP3 player continues unaware. Music may drop out for a short period

Goals

- **Transparent**
  - Shouldn't require modifying kernel or drivers
  - Shouldn't require modifying applications or user-mode libraries
- **Shared**
  - Small amount of mechanism should apply to large number of extensions

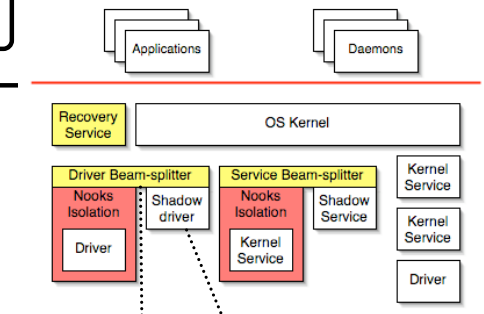
Principles

- **Best effort, but support the rest**
  - Don't try to prevent every fault
  - Don't try to support every extension
- **Design for fault resistance, not fault tolerance**
  - We don't need to prevent every crash to be useful

Architecture

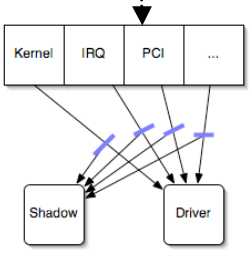
Solution

- Isolate extensions with Nooks lightweight kernel protection domains
- Detect faults using exception handlers and wrappers on extension interfaces
- Recover with **Shadow Drivers**
  - Hot-backup of a kernel extension to take over when the real one crashes
  - Single code base shadows entire class of real extensions - e.g. all sound drivers or all network drivers
  - Shadow driver replays requests to extension to restore its state

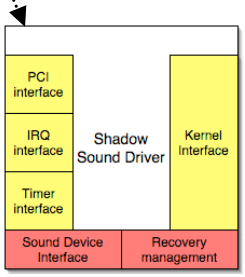


Beam Splitter Details

- Interposed on communication interfaces between driver and kernel at load time and dynamically during driver registration
- Normal Mode operation: Copies calls from driver/kernel to shadow driver
- Failure/Recovery Mode Operation: Redirect calls from driver/kernel to shadow during mask failure



Beam Splitter



Shadow Driver

Shadow Driver Details

- Implements same interfaces as real drivers: e.g. PCI, IRQ, Timer, Network, Character
- Normal mode operation:
  - Record OS resources used
  - Log request data for recovery
- Failure mode operation:
  - Reply kernel/applications with logged information
  - Log data written
- Recovery mode operation:
  - Respond to kernel requests from driver with saved OS resources
  - Plug extension interfaces back into kernel interfaces
  - Replay logged requests to restore driver to pre-crash state

Experience

- **Implementation**
  - Linux 2.4.18 + Nooks for isolation
  - Interpose beam splitter on module load
  - Fault detection with exception handlers
- **Experience**
  - Shadowed several driver classes
    - Network interface device drivers
    - Sound card driver
  - Improved recovery from Nooks
    - Network interface from 12 seconds to 0.5 seconds
    - Sound card from application abort to 0.5 second silence

Related Work

- Recursive & Micro-reboots [Candea & Fox 01]
  - Recover systems by rebooting successively larger portions
- Process-pairs [Bartlett 81]
  - Transfer control to second copy of a process after failure of primary
- Recovery Blocks [Randell 75]
  - Provide second, slim, implementation of function to be called when correctness check fails