Improving the Reliability of Commodity Operating Systems

Michael M. Swift, Brian N. Bershad, and Henry M. Levy

Published 2003

- CS 5204
- John Smiy

Problem

- Extensions account for over 70% of Linux kernel code
- Programmers often less experienced
- Device drivers remain a significant cause of system failures
  - Windows XP - 85% of reported failures
  - Linux- 7 times more likely than the rest of the kernel

Solution

- Nooks
  - Light weight kernel protection domain
- Targets existing extensions
- Recovers extensions quickly
- Recovered automatically from 99% of faults that caused Linux to crash

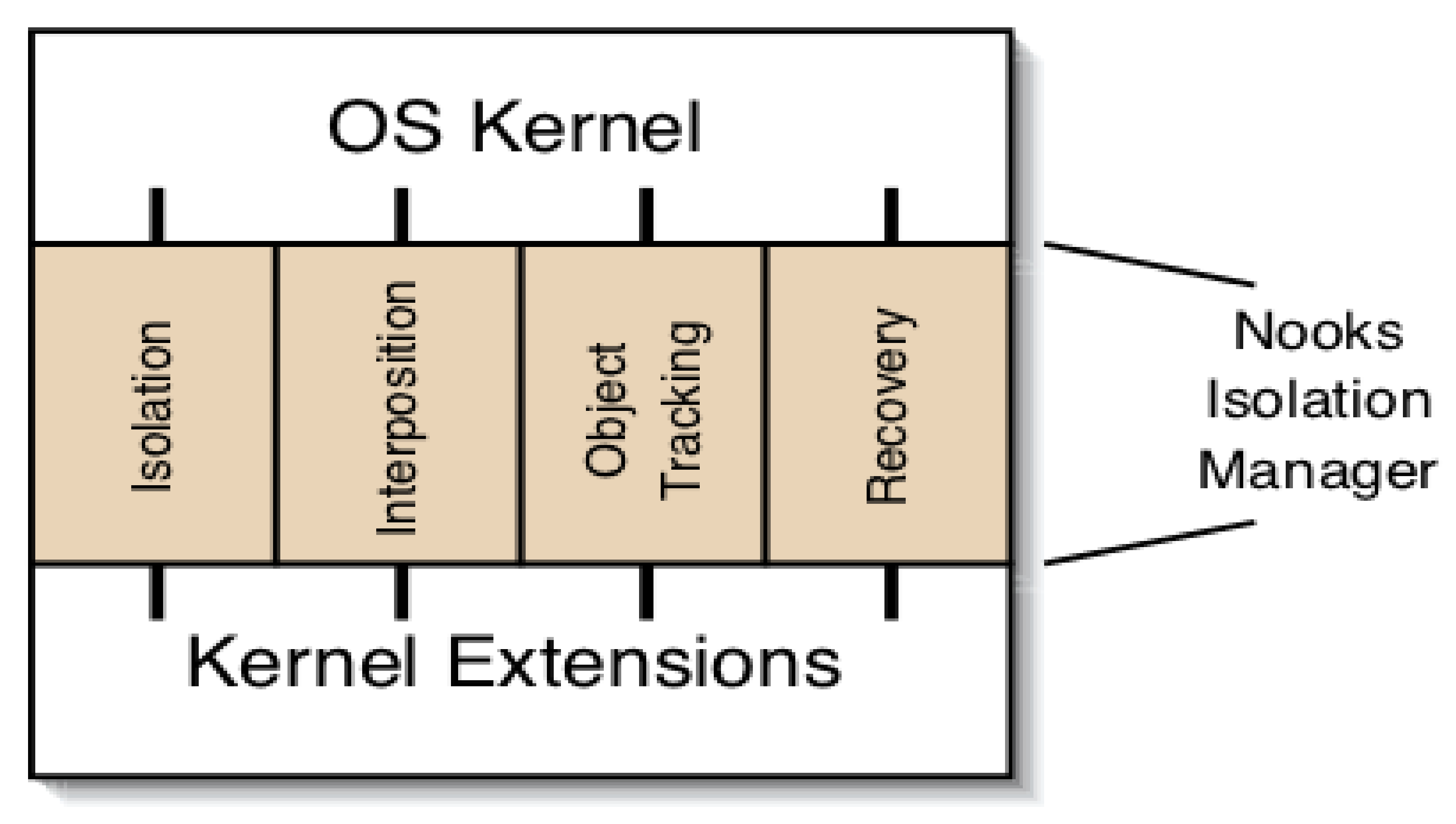
Architecture: Design Principles

- Design for fault resistance, not fault tolerance
  - Malfunctioning driver that does not corrupt kernel data
- Design for mistakes, not abuse
  - Malicious driver that explicily corrupts the system page table

Design Goals

- Isolation
  - Kernel isolated from failures in the extension
- Recovery
  - Recover from extensions crashing
- Backwards Compatibility
  - Applies to existing systems

Nooks Isolation Manager (NIM)



NIM: Isolation

- Extension executes within its own lightweight kernel protection domain
- Management of protection-domain
  - Creation, manipulation, and maintenance
- Interdomain control transfer
  - Extension Procedural Call (XPC)

NIM: Interposition

Transparently integrates existing extentions into the Nook environment

- All control flow occurs through XPC
- All data transfers are managed by an object tracker

NIM: Object Tracker

- Oversees all kernel resources used by extensions
- Tasks
  - Maintains a list of kernel data structures used by extensions
  - Controls all modification to those structures
  - Provides object information when an extension fails
- Copies kernel objects into extension domain

NIM: Object Tracking

- Oversees all kernel resources used by extensions
- Tasks
  - Maintains a list of kernel data structures used by extensions
  - Controls all modification to those structures
  - Provides object information when an extension fails
- Copies kernel objects into extension domain

NIM: Recovery

- Detect software fault
  - Extension invokes a kernel service improperly
    - with invalid arguments
  - Extension consumes too many resources
    - Either triggers recovery or return with error
- Detect hardware fault
  - Processor raises exception
  - Always triggers recovery

Implementation

- Linux 2.4.18 kernel on Intel x86 architecture
- Linux may be the worst-case for Nooks targets
- Intercept function calls between the extensions and kernel

Wrappers

- Extension wrappers and Kernel wrappers
- Module loaders bind extensions to wrappers instead of kernel functions
- Performs work in kernel's protection domain

XPC and Control Transfer

- nooks_driver_call
- nooks_kernel_call

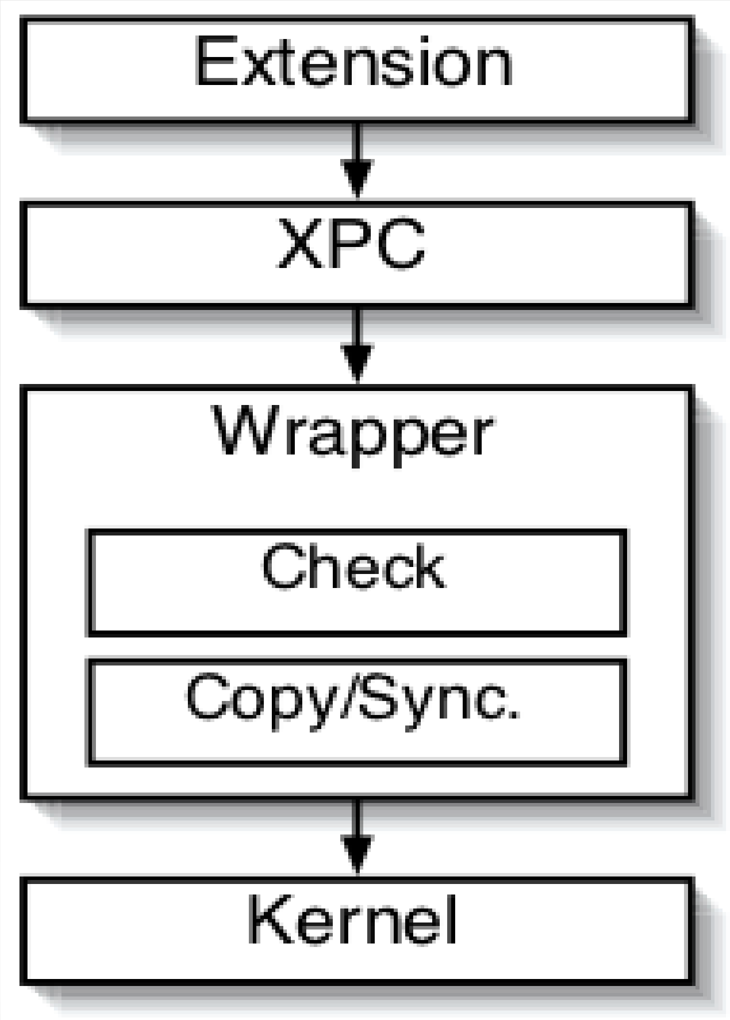
- Take function pointer, argument list, and a protection domain

XPC and Control Transfer

Save the callers content to stack
Find a stack for the calling domain
Change page tables to target domain
Call the function

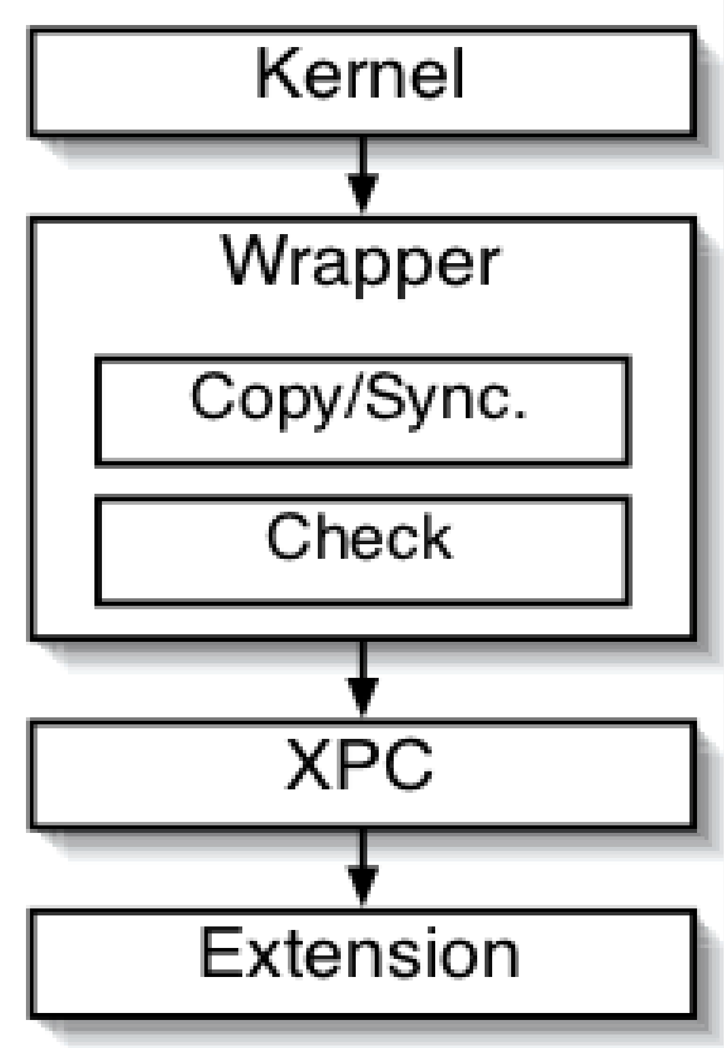
Kernel Wrappers

- Calls XPC so wrapper can execute in kernel's domain
- Calls kernel function directly



Extension Wrapper

- Executes wrapper in kernel's domain
- Performs XPC to tranfer to function in extension



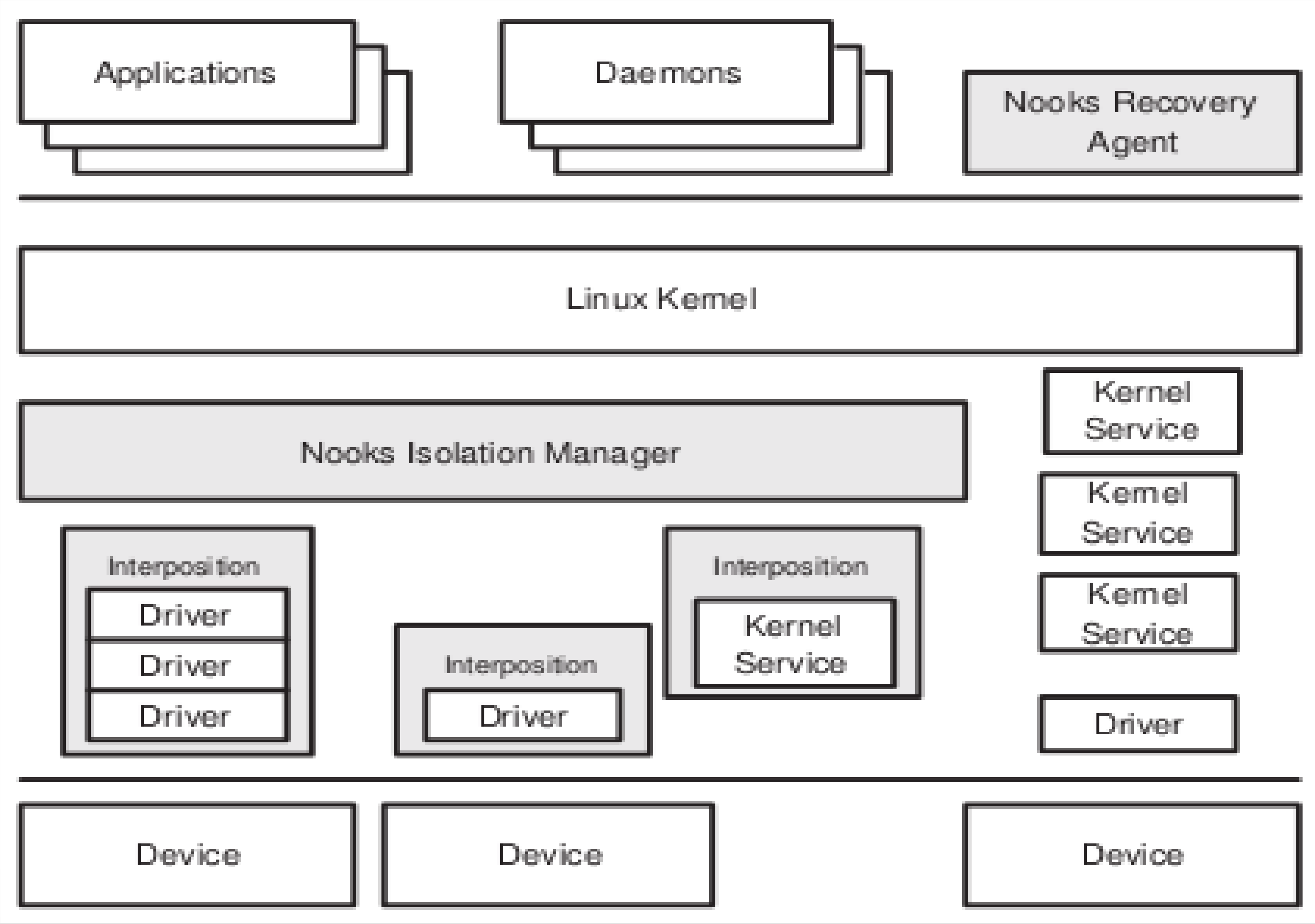
Wrapper Tasks

- Check parameter pointers are valid
  - Object tracker and memory manager
- Creates a copy of kernel objects within extention's protection domain
- Perform XPC into either the kernel or extention to execute desired function

Handling of Kernel Objects in Wrappers

- Linked directly for read only
- Non-performance critical writes to kernel objects are converted into XPC calls.
- Performance Critical writes
  - Shadow copy in extension's domain
  - Synchronized before and after XPC's into the extension

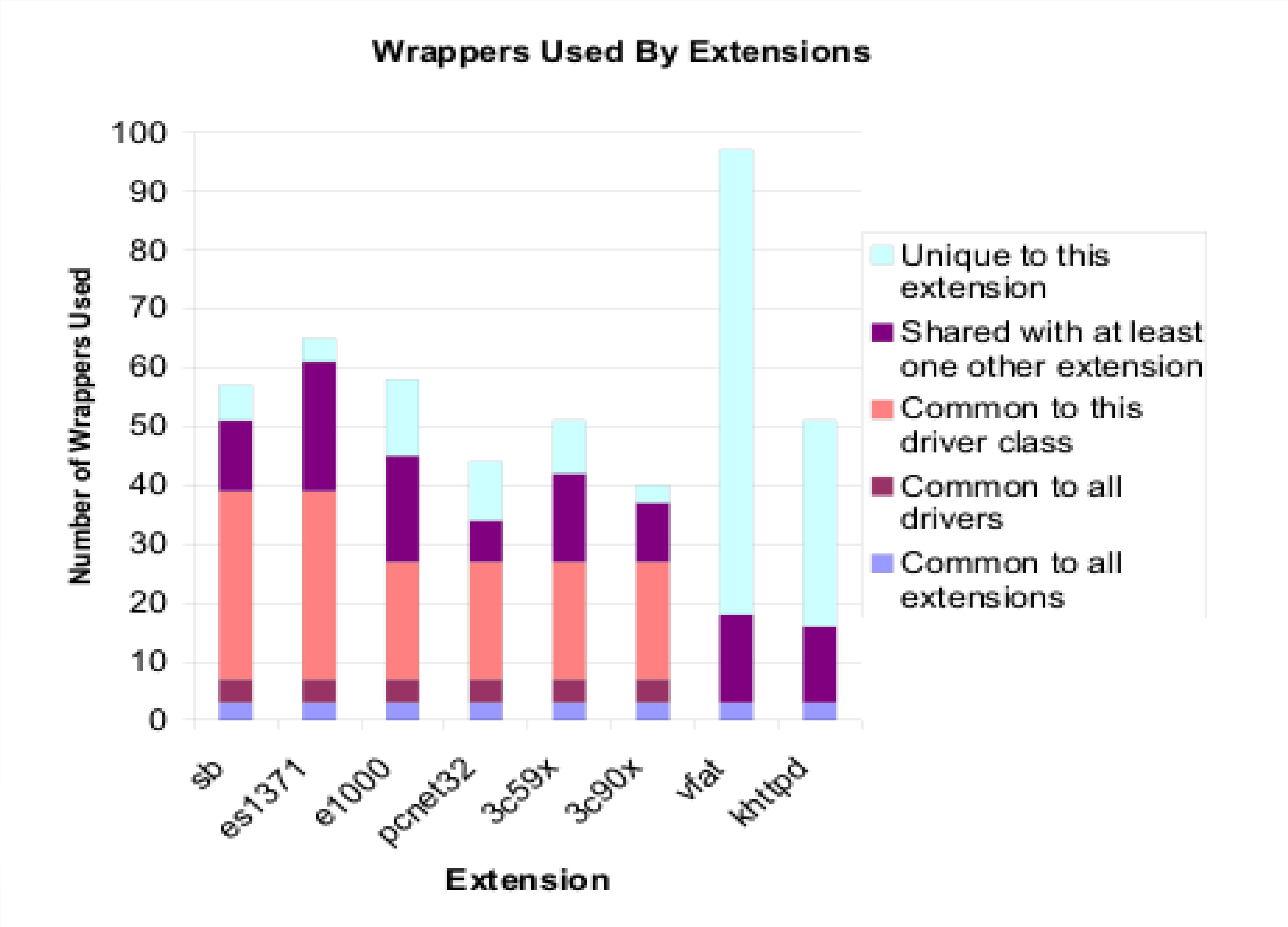
Nook Layer Inside Linux OS



Wrapper Coding

- Main wrapper function written by hand
  - Once per OS
- Automatic generation of wrapper entry code and skeleton of wapper body
  - Based on Linux kernel header files
- Often shared among multiple drivers

Wrapper Code Sharing



Kernel Object

Kernel data structure accessed through a pointer

- All kernel objects are recorded by the object tracker
- Every object that passed through interfaces between the kernel and supported extensions

Object Tracker Tasks

- Records the addresses of all objects in use by extensions
- An association is made between the kernel and extension version of objects
  - For objects written by extensions,
  - Used to pass parameters between protection domains

Recovery

- Triggered through:
  - Software Checks
  - Processor Exceptions
  - Explicit Signals
- Suspends the extension
- Notifies recovery manager

Recovery Manager

Goal is to return the system to a clean state

- Disables interupts from devices using the extension
  - Prevents livelock
- Unwind current tasks
- Releases resources in use by the extension
- Starts user-mode recovery agent

User-Mode Recovery Agent

- Flexible recovery via extension configuration files
- Performs extension specific recovery
- Capable of:
  - Changing configuration parameters
  - Replacing the extension
  - Disable recovery if extention fails frequently

Releasing Kernel Resources

- Walks through object tracker freeing, releasing, or unregistering objects no longer used by devices
- Associates each object type with recovery function
- Releases object to the kernel
- Removes references from the kernel into the extension.

Known Limitations of Implementation

- Does not provide complete isolation or fault tolerance
- Extensions run in kernel mode
  - Cannot prevent deliberate corruption of system state
- Recovery is limited to extensions that can be killed and restarted safely

Testing

- Synthetic fault injection rapidly inserts faults in Linux kernel
- Changes a single instruction in extension code
- Emulates common errors such as:
  - Uninitialized local variables
  - Bad parameters
  - Inverted test conditions

Extensions Isolated

- Device Drivers
- Optional Kernel Subsystem (VFAT)
- Application specific kernel extension (kHTTPd)

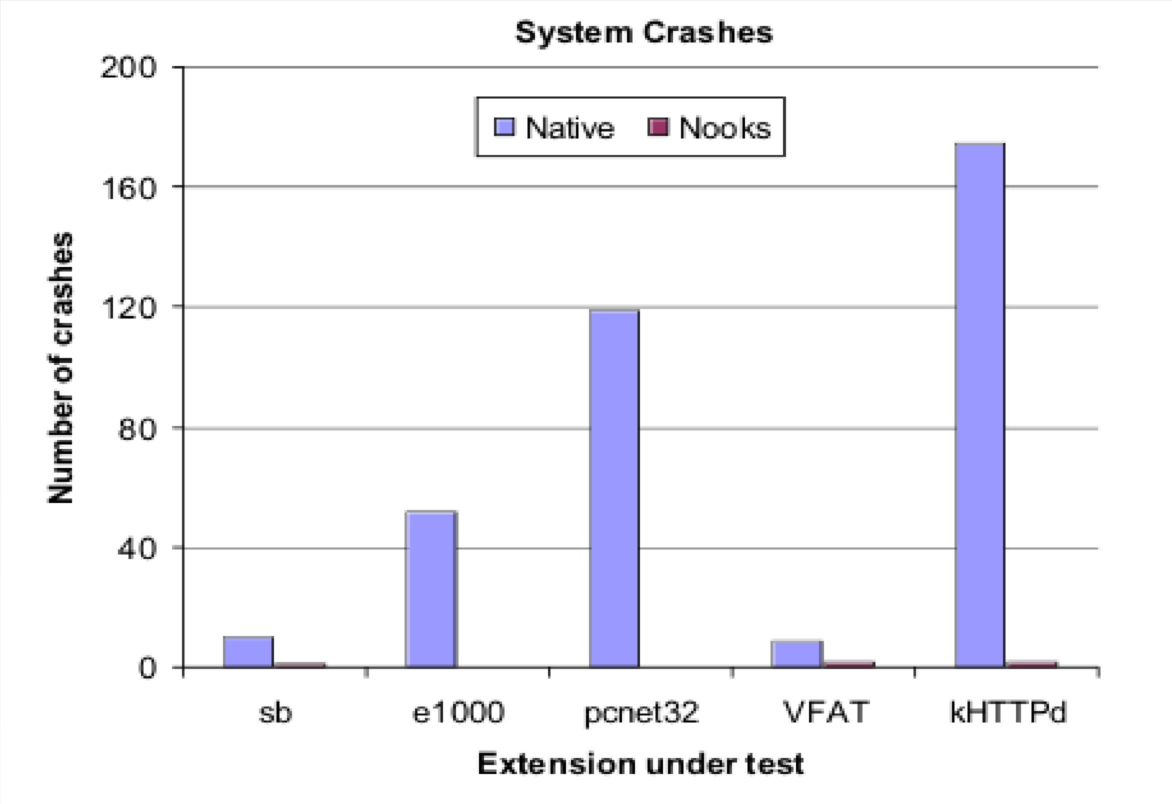
Extension	Purpose
sb	SoundBlaster 16 driver
es1371	Ensoniq sound driver
e1000	Intel Pro/1000 Gigabit Ethernet driver
pcnet32	AMD PCnet32 10/100 Ethernet driver
3c59x	3COM 3c59x series 10/100 Ethernet driver
3c90x	3COM 3c90x series 10/100 Ethernet driver
VFAT	Win95 compatible file system
kHTTPd	In-kernel Web server

Environment

- All except e1000 tests were ran in VMware
- ”Native” test ran
  - Nooks was present but not used
- Each extension ran for 400 trials
- 5 random errors were injected during each trial
- The same 400 trials with the same 5 errors were then run with Nooks enabled.

System crashes

- 317 system crashes reduced to 4 system crashes
- In these 4 crashes the system deadlocked

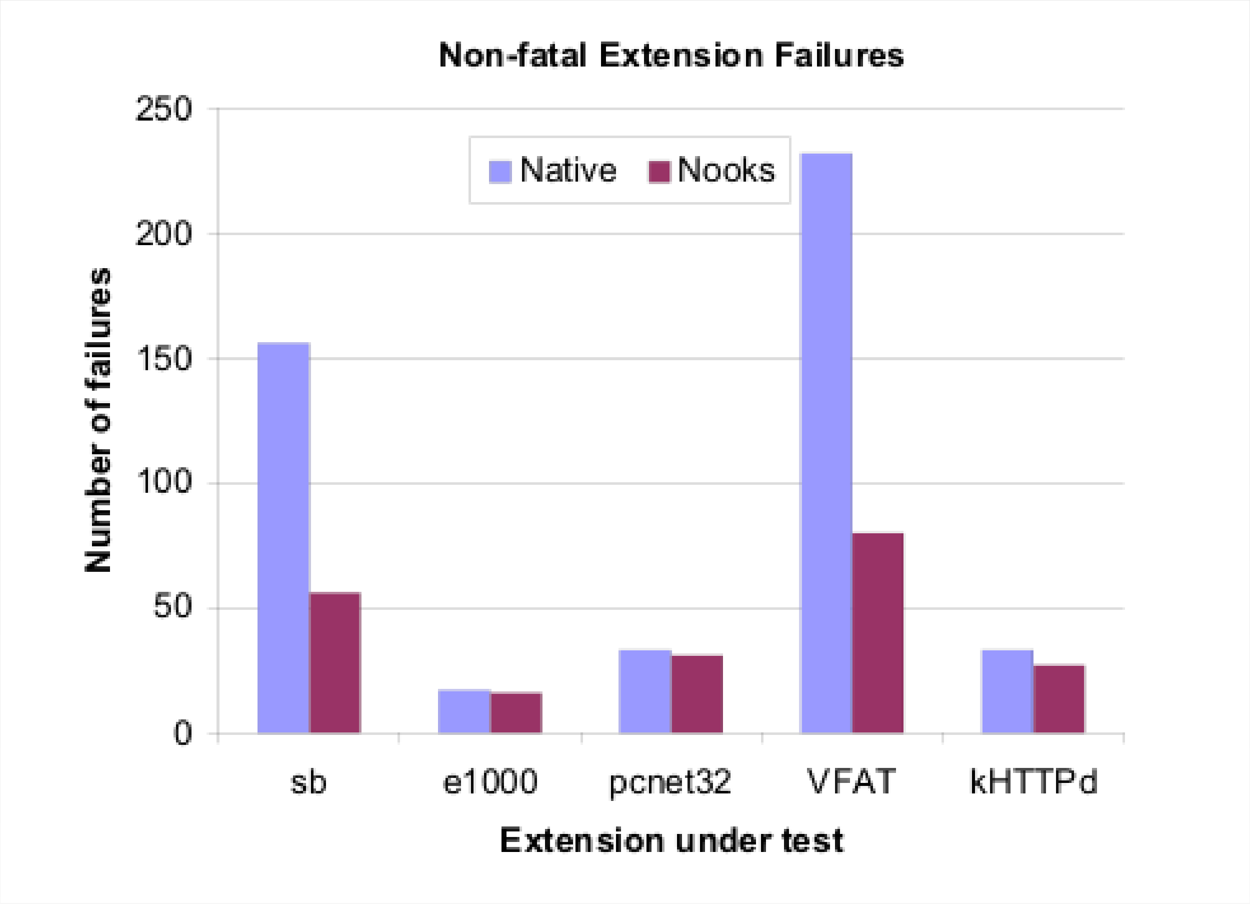


Interupt vs Process Oriented

- Linux treats exceptions in process oriented code as non-fatal
- Process Oriented
  - VFAT and sb
- Interrupt Oriented
  - e1000 and pcnet32
- kHTTPd is process oriented but can corrupt interupt-level data structures

Non-Fatal Errors

- Not designed to detect non-fatal errors
- Processor exceptions



Extension Reliability

- Extension is unloaded, reloaded, and restarted
  - Default
- Directly improved reliaility for network, sb, and kHTTPd extensions.

VFAT Reliability

- VFAT deals with persistent data storage on disk
  - 90% of cases resulted in disk corruption
- Proposed Solution:
  - Synchronize with in memory disk cache before releasing resources
  - Reduced corruption to 10%

Manually Injected Bugs

- Inserted a small number of bugs by hand
- Used most common faults
  - Removed checks for NULL pointers
  - Failure to properaly initialize stack and heap variables
  - Dereferencing a user level pointer
  - Freeing resources multiple times
- Nooks recovered from all these failures

Performance Testing Environment

- Dell 1.7 GHz Pentium 4 PC
- Linux 2.4.18
- 890 MB of RAM
- SoundBlaster 16 soundcard
- Intel Pro /1000 Gigabit Ethernet adapter
- 7200 RPM, 41 GB IDE HDD
- Tests ran on the bare machine

Performance Results

Benchmark	Extension	*XPC Rate (per sec)*	*Nooks Relative Performance*	*Native CPU Util. (%)*	*Nooks CPU Util. (%)*
Play-mp3	sb	150	1	4.8	4.6
Receive-stream	e1000 (receiver)	8,923	0.92	15.2	15.5
Send-stream	e1000 (sender)	60,352	0.91	21.4	39.3
Compile-local	VFAT	22,653	0.78	97.5	96.8
Serve-simple-web-page	kHTTPd (server)	61,183	0.44	96.6	96.8
Serve-complex-web-page	e1000 (server)	1,960	0.97	90.5	92.6

- All of the drivers had less than a 10% penalty
- kHTTPd was nearly 60%
- The number of XPC proposes a burden on the TLB

Performance Results

Benchmark	Extension	*XPC Rate (per sec)*	*Nooks Relative Performance*	*Native CPU Util. (%)*	*Nooks CPU Util. (%)*
Play-mp3	sb	150	1	4.8	4.6
Receive-stream	e1000 (receiver)	8,923	0.92	15.2	15.5
Send-stream	e1000 (sender)	60,352	0.91	21.4	39.3
Compile-local	VFAT	22,653	0.78	97.5	96.8
Serve-simple-web-page	kHTTPd (server)	61,183	0.44	96.6	96.8
Serve-complex-web-page	e1000 (server)	1,960	0.97	90.5	92.6

- The e1000 driver batches incoming messages together
- It does not batch out going messages together
- More XPCs

Nooks Positives

- Prevented 99% of system crashes
- Less than 10% performance overhead for drivers
- Directly improved reliability for network drivers, sb, and VFAT
- Recovers extensions quickly
- Works with existing extensions

Nooks Negatives

- Does not provide complete fault tolerance
- Does not protect against malicious extensions
- Too high of an overhead for some extensions

Questions

?