Improving the Reliability of Commodity Operating Systems

(SOSP'03)

Presented by Brevan Chun

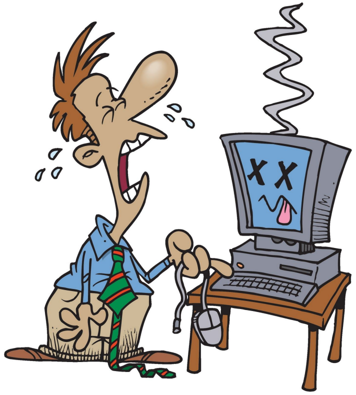
Outline

- ✓ Motivation
- ✓ Design
- ✓ Implementation
- ✓ Testing & Evaluation
- ✓ Key Points
- ✓ Discussion

Motivation

I. Computer reliability needs to improve
   - As the cost of computing drops, the cost of failures increases
   - Unmanaged systems must be reliable

II. OS extensions…
   - increasingly prevalent
     - 70% of Linux code
     - 35,000+ Windows XP drivers
   - account for a large portion of system failures
     - 7x more likely to have code errors in Linux
     - 85% of Windows XP failures



Approach	Required Modifications		
	Hardware	**OS**	**Extension**
Capabilities	yes	yes	yes
Microkernels	no	yes	yes
Languages	no	yes	yes
New Driver Architectures	no	yes	yes
Transactions	no	no	yes
Virtual Machines	no	no	no
Static Analysis	no	no	no
Nooks	no	no	no

Table 1

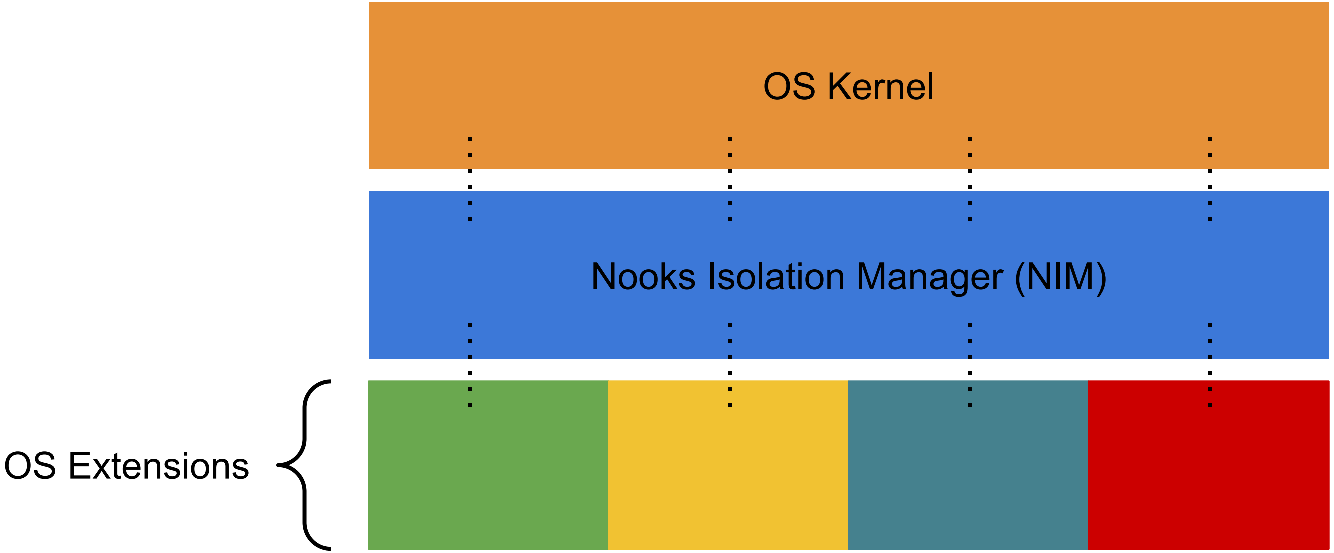
Nooks

- “rather than guaranteeing complete fault tolerance through a new (and incompatible) OS or driver architecture, our goal is to prevent the *vast majority* of driver-caused crashes with *little or no change* to existing driver and system code”

- Design for fault resistance (not fault tolerance)
- Design for mistakes (not abuse)

- Improve OS reliability with better fault resistance
  - ↳ Isolation
  - ↳ Recovery
  - ↳ Backwards compatibility

Nooks



Nooks

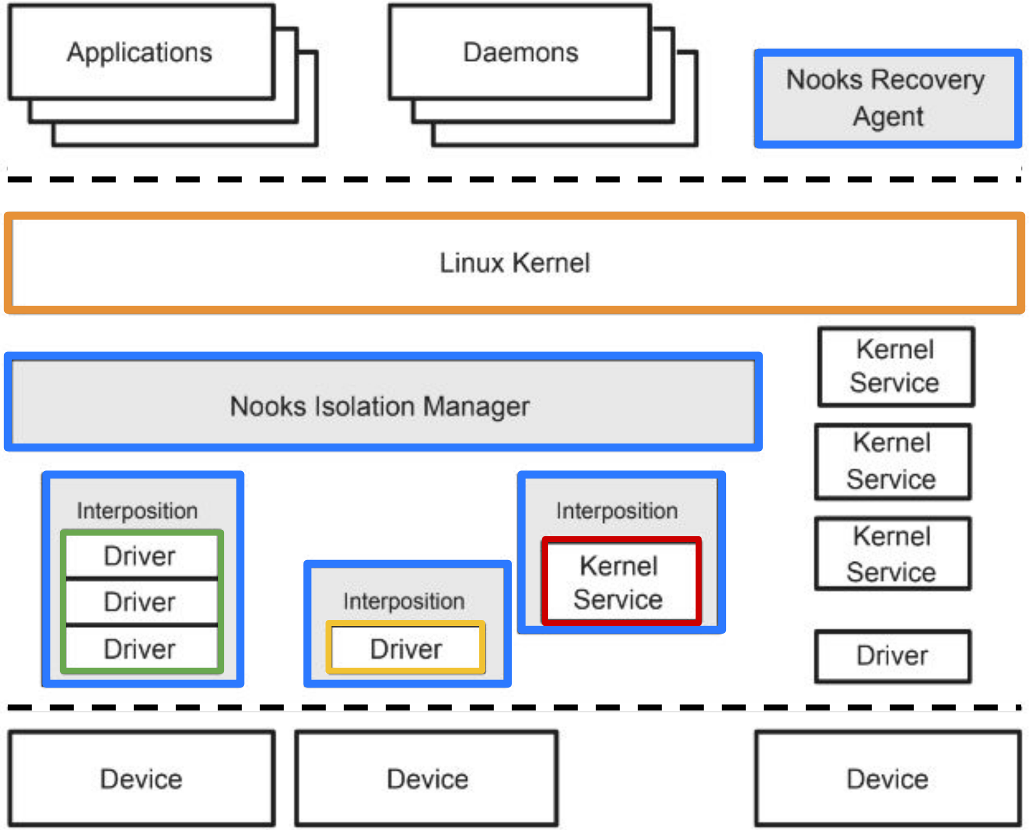


Figure 2

Implementation

A. Isolation - protect kernel from extension failures
B. Recovery - automatic recovery
C. Backward compatibility - applicable to existing systems

1. Isolation
2. Interposition
3. Object tracking
4. Recovery

Source Components	**# Lines**
Memory Management	1,882
Object Tracking	1,454
Extension Procedure Call	770
Wrappers	14,396
Recovery	1,136
Linux Kernel Changes	924
Miscellaneous	2,074
Total number of lines of code	22,266

Table 2

Isolation

- Prevent extension errors from damaging the kernel

- Lightweight kernel protection domains
  - ↳ Kernel privilege
  - ↳ Limited write access
  - ↳ NIM maintains a synchronized copy of the kernel page table (for each ext.)

- Extension Procedure Call (XPC)
  - ↳ Resembles LRPCs but instead assumes a trusted domain & asymmetry
  - ↳ `nooks_driver_call` & `nooks_kernel_call`
  - ↳ Deferred calls

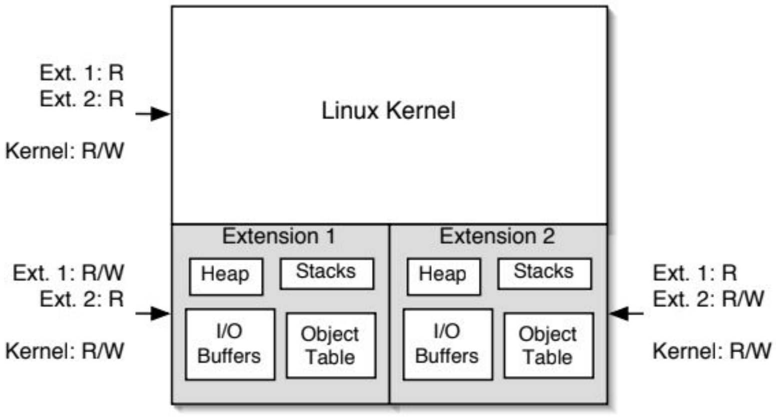


Figure 3: Protection of the kernel address space.

Interposition

- Provide transparency to extensions
- Wrappers
  ↳ Preserve kernel/driver interfaces while enabling protection

  1. Check parameters for validity (w/ object tracker)
  2. Call-by-value-result (copy kernel objects)
  3. Use XPC to execute function

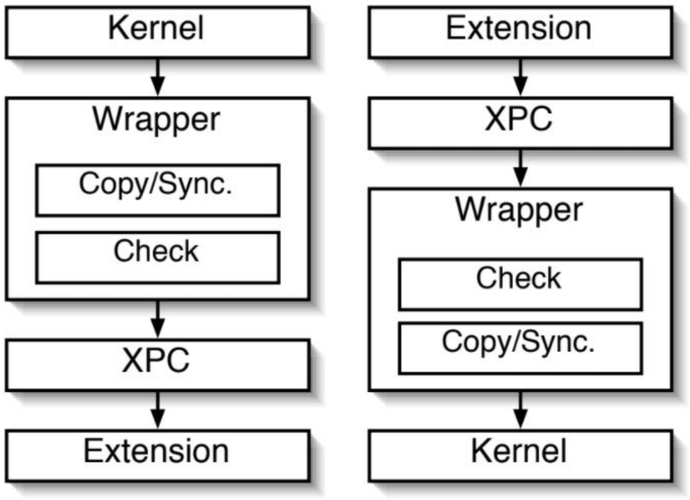


Figure 4

Object Tracking

- Manage the manipulation of kernel objects
- Record all kernel objects in use by extensions
    - ↳ Studied every object that supported extensions used
    - ↳ Record the address and association
- Perform garbage collection
    - ↳ Protection-domain hash table

Recovery

- Detect and recover from faults
  - ↳ Detection through software checks, exceptions, signals
  - ↳ Flexible recovery policy
  - ↳ Release resources
- Hardware faults must trigger recovery
  - ↳ Software faults can return error code or do recovery
- User/program can explicitly trigger recovery

Tests

- Linux 2.4.18
- 8 extensions
  - 2 sound drivers
  - 4 ethernet drivers
  - VFAT file system
  - kHTTPd kernel web server
- Driver stress tests
  - Play MP3 file
  - ICMP-ping
  - TCP streaming
  - Untar and compilation
  - Web load generator

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

Table 3: The extensions isolated and the function that each performs. Measurements are reported for extensions shown in bold.

Tests

1. Synthetic fault injection
   - Nooks eliminated 99% of crashes
   - System deadlock in remaining cases

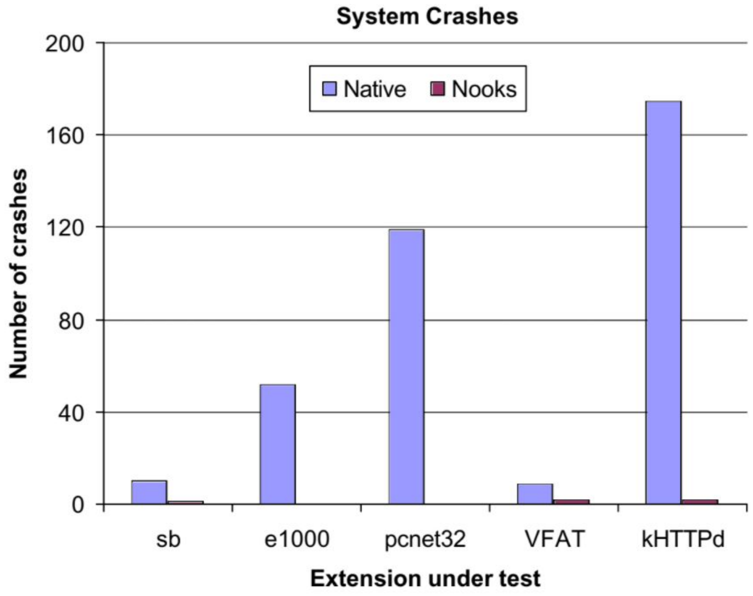


Figure 6: The reduction in system crashes in 2000 fault-injection trials (400 for each extension) observed using Nooks. In total, there were 317 system crashes in the native configuration and only four system crashes with Nooks.

Tests

2. Non-fatal failures
    - Nooks catching exceptions and recovering the extensions
    - A “nanny” process or manual invocation to recover undetected failures

3. Recovery errors
    - VFAT FS is corrupted upon recovery 90% of the time
    - Nook extension could improve reliability

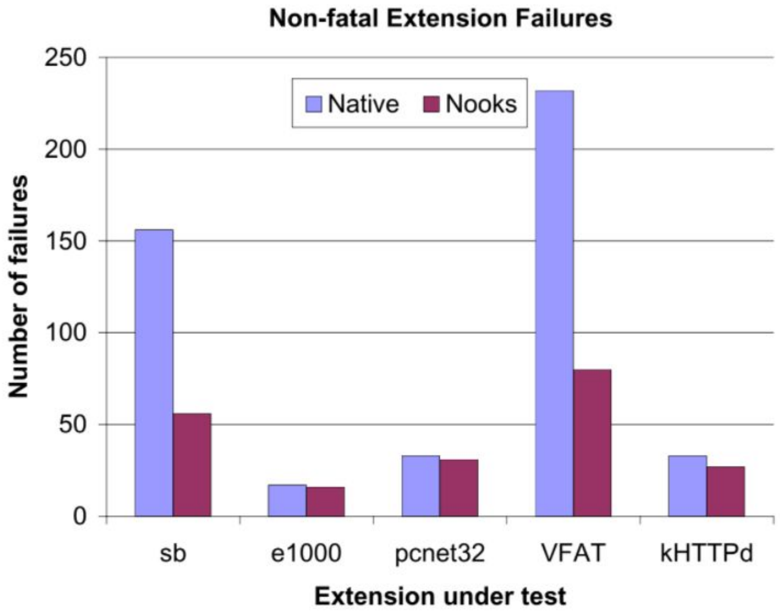


Figure 7: The reduction in non-fatal extension failures observed using Nooks. In total, there were 512 such failures in the native configuration and 212 with Nooks.

Tests

4. Manually injected errors
    - Manually modified extensions are detected and recovered by Nooks

5. Latent bugs
    - Found several bugs in OS extensions under test

Performance

- Performance is closely related to XPC frequency
    - ↳ Low performance comes from high CPU utilization
    - ↳ TLB misses when changing protection domains
    - ↳ Object tracking is slow
- Speedup is possible

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

Table 4: The relative performance of Nooks compared to native Linux for six benchmark tests. CPU utilization is accurate to only a few percent. Relative performance is determined either by comparing latency (Play-mp3, Compile-local) or throughput (Send-stream, Receive-stream, Serve-simple-web-page, Serve-complex-web-page). The data reflects the average of three trials with a standard deviation of less than 2%.

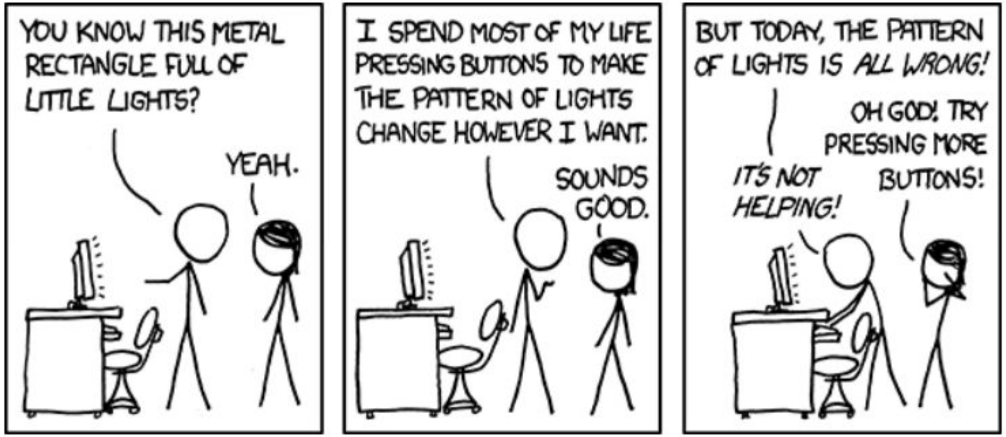
Key Points

1. Nooks increases system reliability by protecting the OS from driver failures
    a. Modest effort to implement in Linux
    b. Nooks does not modify extensions
    c. Isolating extensions can improve system reliability
2. Nooks makes compromises to maintain compatibility
    a. Backward compatibility **>** fault tolerance
3. Nooks is effective in tests and efficiency is workload dependent

Michael M. Swift, Brian N. Bershad, and Henry M. Levy. 2003. Improving the reliability of commodity operating systems. In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03). Association for Computing Machinery, New York, NY, USA, 207–222. DOI:https://doi.org/10.1145/945445.945466

Discussion

1. Is backward compatibility worth the compromises?
2. Is Nooks trying to be too general? Should it just focus on drivers?
    a. Non-driver performance is poor
3. What else could be encapsulated by Nooks to provide fault resistance?



xkcd.com/722