Nooks
Improving the Reliability of Commodity Operating Systems
Swift, Bershad, and Levy
SOSP’03

1. What was the motivation for Nooks?

- Reliability is important - cost of failures is high
- Extensions (modules / device drivers) account for 70% of Linux
  - written by less experienced programmers
  - yet reside in kernel space
- Extensions cause most OS failures (85% WinXP)

→ Treat extensions differently than rest of Kernel
2. What were the design principles and goals of Nooks?

**Principles:**

1) Fault resistance, not fault tolerance (don't handle all)

2) Mistakes, not abuse

→ Better performance + better reliability

**Goals:**

1) Isolation: (detect extension problem before infects rest of kernel)

2) Recovery: automatic recovery to permit applications to continue

3) Backward Compatibility: Existing, with minimal changes

Reviewer: Does Eval show they met their goals?
3. What are the components of the Nooks Isolation Manager (NIM)?

Layer code between OS kernel + Extensions:

1) Isolation
2) Interposition
3) Object Tracking
4) Recovery

Should be transparent
4. Why is isolation needed? At a high level, how is it provided?

- Prevent damage to kernel or other extensions

- Lightweight protection domain:
  - run at same processor privilege, but reduced memory access

- XPC extension procedure call
  to jmp before &
5. Why is Interposition needed? At a high level, how is it provided?

- Catch all control + data transfers
  \[ \text{XP} \]  \text{Object Tracker}  

How? Wrappers (stubs)
  of kernel's extension API
  + extension entry points
6. Why is Object Tracking needed? At a high level, how is it provided?

- Maintain list of kernel structures manipulated by extension
- Why? Control mods to kernel structures
- Help w/ cleanup on recovery
7. Why is Recovery needed? At a high level, how is it provided?

- Need to be able to detect problems, restart extension

- S/W fault:
  - call routine with wrong args
  - too many resources

- H/W fault:
  - read/write pages o/o permission
8. How much work was it to implement Nooks in Linux 2.4.18?

Significant!
700 kernel functions, 650 extension-entry functions
18 developer months
22,266 lines of code
14,396 wrapper lines!
924 Linux kernel changes
9. For memory management, what memory rights does an extension have? What memory rights does the main Linux kernel have? How is this protection provided?

Extension: r/w own domain
r of kernel

Kernel: r/w of all

Each extension has own page tables
(changing protection domains → changing page tables)
10. Why is a synchronized copy of the kernel page table needed for each domain? Are there any implications of this? Why does the Nooks design prevent bugs but not malicious extensions? What is the performance cost of switching between lightweight protection domains?

Each extension
- Needs to know about pages in kernel address space

- Required changes to Linux when modifying kernel page changes
  (Is costly w/ many extensions??)

- Nothing to prevent extension from modifying hw page table base register

- TLB flush on every switch between domains
11. How is control between an extension and kernel domain handled with XPC? What is the purpose of a deferred call?

nooks::driver::call: function ptr, args, protection domain

- kernel -

- save context, find stack

exchange page tables to target domain

call function

(wrappers around XPC provide transparency and do the checking of parameters)

Deferred: Used for batching multiple XPC calls; expensive to switch between domains frequently
12. Did the Linux kernel need to be modified to support isolation?

Yes

1) Maintain coherency of kernel read ptr.

2) Handle exceptions in Nooks domains

3) Global variable for task ptr
13. In Linux, extensions sometimes directly access global data structures. How is this handled? When is XPC used? When is it not? How should one determine which approach to use?

- If just read, okay
- Write: Replace macros + inline functions w/ wrapped calls
- If direct in extension—have to find all of these!

XPC: Not perf critical

Shadow copy of kernel object; perf critical in extension domain
- sync before + after multiple calls

Work, work, work...
14. What does a wrapper need to do? What is the difference between call-by-value-result and call-by-reference?

- **call-by-value-result**
  - arg V → copy to L during call
  - copy back to V when done

- **call-by-reference**
  - single-threaded
  - same semantics as call by ref
  - can do checks on values/results when copy in/out

*checking of parameters*
- very context specific!
- interact with object tracker + mem manager

(kernel ➔ extension)

(extension ➔ kernel)

(wrapper ➔ XPC)

(copy/sync)
15. How much work is it to write a wrapper?

- Lots?
- By hand
- Requires knowledge of parameter use
- Reusable by all extensions using same interface
16. What must the Object Tracker track?

- all kernel objects manipulated by extensions
- 43 different types inspected every interface to find set

1) Records addresses of objects used by extension
   - record in single XPC call table or per-protection-domain hash table if long-lived

2) For objects modified, track association between extension + kernel version

  Must know lifetime of objects
17. What does the Nooks recovery manager do?

1) Detects failure

2) Unwind executing tasks
   
   Unload extension
   Release all resources (including refs to obj)
   Reload + restart
   
   Does not ensure that apps can continue to use driver! (Swift's next paper)
18. What are some limitations of Nooks?

- can't prevent priv. instr.
- can't prevent infinite loops
- check parameters, but not completely
- just kill + restart
19. Does Nooks meet its goals of Isolation, Recovery, and Backward Compatibility?

- Isolation: Prevent errors from crashing system?
  
  Fig 6: Positive
  
  Fig 7: Not so good w/ non-fatal errors
  - can't detect problem

- Recovery?: can an app run??
  - If data is damaged before detection, problem
  - See Table 3 of Shadow Driver paper

- Backward Compatibility
  - Some Linux changes
  - Would need more extensions to know if have all wrappers...
20. Is the performance of Nooks acceptable?

Table 4.

ok if low IPC rate or low CPU util
not good for complex extension
(kHTTPd) or VFAT
21. Conclusions?

- Good motivation for driver reliability important
- Different structure needed

- Lightweight protection domain

- Lots of work by hand
- Performance impact at crossings
- Doesn't automatically recover for applications
  (shadow drivers)