Improving the Reliability of Commodity Operating Systems

Swift, M. M., Bershad, B. N., & Levy, H. M., *Proc. of 19th ACM Symposium on Operating Systems Principles*, 2003.

(Also appeared in ACM Transactions on Computer Systems, 22(4), 2004).

Presented by Hari K. Pyla

Outline

- Introduction
- Previous work
- Motivation
- Nooks
- Architecture
- Implementation
- Performance
- Analysis and conclusions

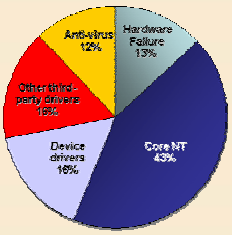
2

Introduction

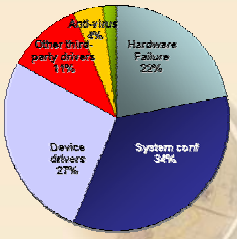
- What issues qualify a good Operating System?
  - Performance
  - Functionality
  - Scalability
  - Reliability ⇦

3

Introduction: Analysis of Crashes



Microsoft NT

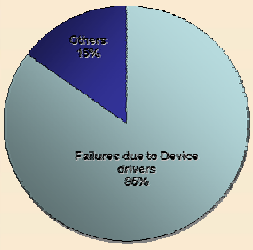


Microsoft 2000

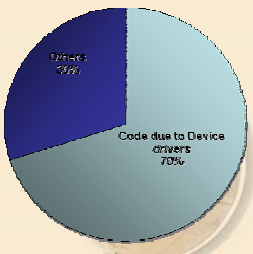
4

Data obtained from http://nooks.cs.washington.edu/retreat.ppt

Introduction: Analysis of Crashes



Microsoft XP



Linux

5

Previous Approaches

Name	Description	Used In
Kernel wrapping	Verify all parameters on calls between the kernel and device drivers	Microsoft Driver Verifier
Hardware memory protection	Prevent device drivers from writing to kernel memory	Palladium, Shinagawa
Privilege level change	Prevent device drivers from executing privileged instructions and/or emulate privileged instructions	Exokernel
Software fault isolation	Inject code into device drivers to ensure that addresses and instructions are safe	Vino
Safe languages	Rely on the compiler/virtual machine to allow only safe (non-faulting drivers to be loaded	SPIN

6

Comparison of Driver Safety Approaches

Parameter	Kernel wrapping	Hardware memory protection	Privilege level change	Software fault isolation	Safe languages	Nooks
Requires rewriting driver	**No**	**No**	**No**	**Maybe**	**Yes**	**No**
Easily supports recovery	**No**	**Yes**	**Yes**	**No**	**No**	**Yes**
High performance for small data vol.	**Yes**	**No**	**No**	**Yes**	**Yes**	**Yes**
High performance for large data vol.	**Yes**	**Yes**	**Yes**	**No**	**No**	**Yes**
Isolates memory corruption	**No**	**Yes**	**Yes**	**Maybe**	**Yes**	**Yes**

7

Motivation

- Address the ever increasing system crashes due to new OS extensions
- Bridge gap between OS kernel and Device Drivers creators
- Differentiate crashes due to malicious intent and programming errors
- Design for fault resistance not fault tolerance
- Interested in reliability, not security
- Retroactive solution
  - With low overheads
  - Backward compatibility, "Patch" style approach

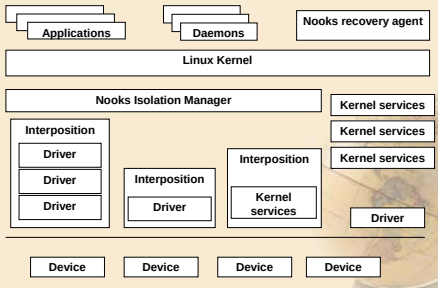
8

Nooks: Overview

- Middleware that isolates OS and device drivers
  - Reliability subsystem to prevent driver failures
  - Recovers quickly with no lost application state
  - Requires only minimal change to the kernel
  - Requires no source changes for most device drivers

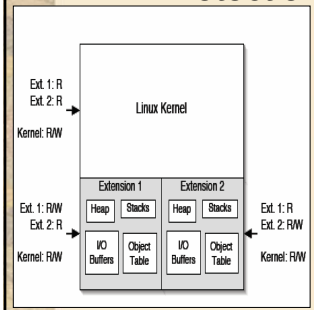
9

Nooks Layer Inside Linux OS



10

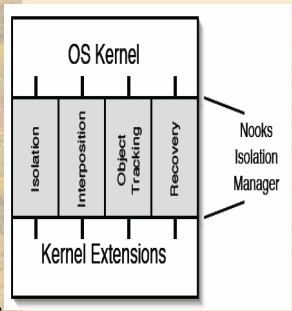
Isolation: Lightweight Kernel Protection Domains



- Executes in kernel mode
- Is logically part of the kernel
- Has read access to kernel data
- Has *restricted* write access to kernel data

11

Architecture: Functions of NIM



- Isolation
- Interposition
- Object Tracking
- Recovery

12

Architecture: "Isolation" in NIM

- Functionality Provided
  - Prevent extension errors from damaging kernel or other extensions
  - create, manipulate and maintain domains (Protection Domain Management)
  - Inter domain control transfer
- Internals
  - Memory management (create stacks, heaps, sockets etc.)
  - Extension Procedure Call (XPC)
    e.g. nooks_driver_call() & nooks_kernel_call()

13

Architecture: "Interposition"

- Functionality provided
  - Transparently integrate extensions into Nooks
  - Ensure kernel-extension control flow thru XPC
  - Ensure data transfer between kernel and extension is monitored by *object-tracking* code
- Internals
  - wrapper stubs
  - Modified module loader
  - Modified kernel module initialization code
  - Function pointers from extension to kernel replaced by wrapper pointers

14

Architecture: "Object Tracking"

- Functionality provided
  - Maintain list of kernel data structures manipulated by an extension
  - Control modifications to structures
  - Provide information for cleanup when extensions fail
- Internals
  - Record addresses of kernel objects in use by an extension
  - Monitor lifetime of objects and perform garbage collection
  - Maintain per protection domain hash table, current task structure etc.

15

Architecture: "Recovery"

- Functionality provided
  - Determination of whether to trigger recovery or return error code to invoking extension
  - Detection and recovery of various extension faults (h/w and s/w)
- Internals
  - Disable interrupt processing
  - Start user mode recovery agent
  - release resources in use by extension
  - change configuration
  - Replace, reload and restart extensions

16

Working of Nooks: An Example

- New USB device connected: driver gets loaded
- Loader invokes Nooks wrapper stubs at Nooks kernel runtime interface
- Wrapper intercepts the call invoking object tracking code
  - manage parameters passed between the kernel and driver or vice versa
  - Wrapper transfers control from caller's to callee's domain and vice versa using XPC
- Neither driver nor kernel is aware of existence of Nooks layer!

17

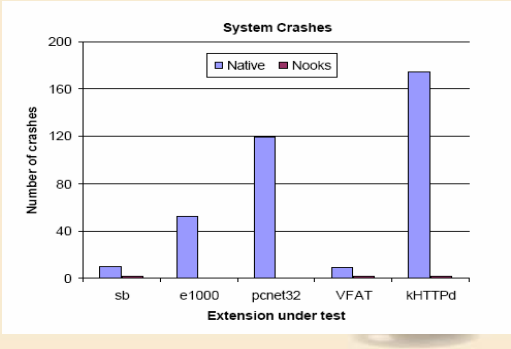
Implementation

- OS: Linux
- Kernel: 2.4.18
- Hardware: Intel x86
- Time: 18 months
- Language used: C

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

18

Performance and Reliability



19

Performance and Reliability

Overhead Benchmarks

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

20

Nooks: Limitations

- Does not provide complete fault tolerance
- Cannot prevent extensions from deliberately executing privileged instructions
- Does not prevent infinite loops inside extensions
- Can perform only a static check in terms of parameters passed
- Recovery is limited to drivers that can be killed and restarted safely

21

Analysis and Conclusions

- Drivers limit OS reliability, are major source of failures
- OS should remove dependence on driver safety
- Existing OS can be extended to run existing driver code safely
- Nooks philosophy is practical and can be easily incorporated
- Nooks lightweight kernel protection domains support reliable driver execution by
  - Preventing kernel corruption
  - Supporting existing driver API

22

Thank you

23