Improving the Reliability of Commodity Operating Systems

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

Department of Computer Science and Engineering
University of Washington

SOSP 2003

Speaker: Chin-her Chang
OUTLINE

- INTRODUCTION
- ARCHITECTURE
- IMPLEMENTATION
- RELIABILITY
- PERFORMANCE
- CONCLUSIONS
This paper describes the architecture, implementation, and performance of Nooks.

Nooks executes each extension in a lightweight kernel protection domain.
Nooks differs from earlier efforts in two key ways:

1. we target existing extensions for commodity operating systems rather than propose a new extension architecture

2. we use C, a conventional programming language.
The Nooks architecture is based on two core principles:

1. Design for fault resistance, not fault tolerance.

2. Design for mistakes, not abuse.
### Architecture

<table>
<thead>
<tr>
<th>Approach</th>
<th>Required Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hardware</td>
</tr>
<tr>
<td>Capabilities</td>
<td>yes</td>
</tr>
<tr>
<td>Microkernels</td>
<td>no</td>
</tr>
<tr>
<td>Languages</td>
<td>no</td>
</tr>
<tr>
<td>New Driver Architectures</td>
<td>no</td>
</tr>
<tr>
<td>Transactions</td>
<td>no</td>
</tr>
<tr>
<td>Virtual Machines</td>
<td>no</td>
</tr>
<tr>
<td>Static Analysis</td>
<td>no</td>
</tr>
<tr>
<td>Nooks</td>
<td>no</td>
</tr>
</tbody>
</table>
ARCHITECTURE

- **Goals:**
  1. *Isolation*
  2. *Recovery*
  3. *Backward Compatibility*
Functions

reliability layer:

It is inserted between the extensions and the OS kernel.

A crucial property of this layer is transparency.
Figure 1: The Nooks Isolation Manager, a transparent OS layer inserted between the kernel and kernel extensions.
Isolation

*lightweight* *kernel protection domain*: This domain is an execution context with the same processor privilege as the kernel but with write access to a limited portion of the kernel’s address space.
ARCHITECTURE

- task of the isolation mechanism:
  1. protection-domain management
  2. inter-domain control transfer

  *Extension Procedure Call (XPC)*
Interposition code ensures that:

(1) all extension-to-kernel and kernel-to-extension control flow occurs through the XPC mechanism.

(2) all data transfer between the kernel and extension is viewed and managed by Nooks’ object-tracking code.
ARCHITECTURE

- **wrapper stubs**: Nooks’ stubs provide transparent control and data transfer between the kernel domain and extension domains.
ARCHITECTURE

- **Object Tracking**

  *object-tracking code:*
  
  (1) maintains a list of kernel data structures that are manipulated by an extension.

  (2) controls all modifications to those structures.

  (3) provides object information for cleanup when an extension fails.
ARCHITECTURE

- **Recovery**
  
  **software fault**: extension invokes a kernel service improperly or when an extension consumes too many resources.

  **hardware fault**: when the processor raises an exception during extension execution.
We implemented Nooks inside the Linux 2.4.18 kernel on the Intel x86 architecture.

The interfaces are implemented as C language structures containing a set of function pointers.
## IMPLEMENTATION

<table>
<thead>
<tr>
<th>Source Components</th>
<th># Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Management</td>
<td>1,882</td>
</tr>
<tr>
<td>Object Tracking</td>
<td>1,454</td>
</tr>
<tr>
<td>Extension Procedure Call</td>
<td>770</td>
</tr>
<tr>
<td>Wrappers</td>
<td>14,396</td>
</tr>
<tr>
<td>Recovery</td>
<td>1,136</td>
</tr>
<tr>
<td>Linux Kernel Changes</td>
<td>924</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>2,074</td>
</tr>
<tr>
<td><strong>Total number of lines of code</strong></td>
<td><strong>22,266</strong></td>
</tr>
</tbody>
</table>

Table 2: The number of non-comment lines of source code in Nooks.
Isolation

The isolation components of Nooks consist of two parts:

(1) memory management.

(2) Extension Procedure Call (XPC).
Control transfer in XPC is managed by two functions internal to Nooks:

1. `nooks_driver_call` transfers from the kernel into an extension.
2. `nooks_kernel_call` transfers from extensions into the kernel.
XPC also supports a *deferred call* mechanism.

Two queues exist for each domain: the *extension-domain queue* holds delayed kernel calls, and the *kernel-domain queue* holds delayed extension calls.
changes to the Linux kernel to support isolation:

1. to maintain coherency between the kernel and extension page tables.
2. modified the kernel exception handlers to detect exceptions that occur within Nooks’ protection domains.
IMPLEMENTATION

Figure 3: Protection of the kernel address space.
Interposition

Interposition required two changes to Linux kernel code:

1. modified the standard module loader to bind extensions to wrappers.
2. modified the kernel’s module initialization code.
IMPLEMENTATION

3. all function pointers passed from the extension to the kernel are replaced by wrapper pointers.
4. changed macros and inline functions that directly modify kernel objects into wrapped function calls.
IMPLEMENTATION

- For object modifications:
  1. Not performance critical
     Nooks converts the object access into an XPC into the kernel.
  2. For performance-critical
     create a shadow copy of the kernel object within the extension’s domain.
IMPLEMENTATION

Wrappers

There are two types of wrappers:

- kernel wrappers
- extension wrappers
IMPLEMENTATION

Figure 4: Control flow of extension and kernel wrappers.
Object Tracking

The Nooks object tracker in Nooks performs two tasks:

1. it records the *addresses* of all objects in use by an extension.

2. the object tracker records an association between the kernel and extension of the object.
Recovery

Recovery in Nooks consists of two parts:

1. After a fault occurs, the recovery manager releases resources in use by the extension.

2. The user-mode agent coordinates recovery and determines what course of action to take.
Achieving Transparency

1. Nooks provides wrapper stubs for every function call in the extension-kernel interface.

2. Nooks provides object-tracking code for every object type that passes between the extension and the kernel.
# RELIABILITY

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

![Bar chart showing system crashes](image)

- **System Crashes**
- **Y-axis**: Number of crashes
- **X-axis**: Extension under test (sb, e1000, pcnet32, VFAT, kHTTPd)
- Comparison of crashes between **Native** and **Nooks**

The chart illustrates the number of system crashes for different extensions under test, comparing native and nooks environments.
RELIABILITY

Non-fatal Extension Failures

<table>
<thead>
<tr>
<th>Extension under test</th>
<th>Native</th>
<th>Nooks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sb</td>
<td>150</td>
<td>50</td>
</tr>
<tr>
<td>e1000</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>pcnet32</td>
<td>70</td>
<td>30</td>
</tr>
<tr>
<td>VFAT</td>
<td>250</td>
<td>100</td>
</tr>
<tr>
<td>kHTTPd</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Benchmark</td>
<td>Extension</td>
<td>XPC Rate (per sec)</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Play-mp3</td>
<td>sb</td>
<td>150</td>
</tr>
<tr>
<td>Receive-stream</td>
<td>e1000 (receiver)</td>
<td>8,923</td>
</tr>
<tr>
<td>Send-stream</td>
<td>e1000 (sender)</td>
<td>60,352</td>
</tr>
<tr>
<td>Compile-local</td>
<td>VFAT</td>
<td>22,653</td>
</tr>
<tr>
<td>Serve-simple-web-page</td>
<td>kHTTPd (server)</td>
<td>61,183</td>
</tr>
<tr>
<td>Serve-complex-web-page</td>
<td>e1000 (server)</td>
<td>1,960</td>
</tr>
</tbody>
</table>
This paper described Nooks, a new reliability layer intended to significantly reduce extension-related failures.

Nooks demonstrates that it is possible to realize an extremely high level of operating system reliability with a performance loss ranging from zero to just over 60%.
CONCLUSIONS

(1) implementation of a Nooks layer is achievable with only modest engineering effort, even on a monolithic operating system like Linux.

(2) extensions such as device drivers can be isolated without change to extension Code.

(3) isolation and recovery can dramatically improve the system’s ability to survive extension faults.