Reliable operating systems
Reliable operating systems

*Can we make operating systems reliable and secure?*

[Andy Tanenbaum, Jorrit Herder, Herbert Bos, 2006]
What makes operating systems unreliable?

• They are huge

• They have poor fault isolation
What makes operating systems unreliable?

• They are huge
  • millions of lines of code
What makes operating systems unreliable?

- They are huge
  - millions of lines of code
  - between 6 and 16 bugs per 1000 lines of code
What makes operating systems unreliable?

- They have poor fault isolation
- Thousands of procedures linked together as a single binary program
- Can overwrite key kernel data-structures
- If a virus infects even just one procedure, it can spread quickly to the whole kernel
What can we do?

• Improve on legacy operating systems
  • device drivers are the core of the problem
    • isolate them \([\textit{Nooks}, \text{SOSP'03}]\)
    • synthesize them \([\textit{Termite}, \text{SOSP'09}]\)
  • Re-design the OS
    • microkernel
Nooks

• conservative approach, maintains monolithic kernel design

• protects the kernel from buggy device drivers

*Nooks: An architecture for reliable device drivers*

[Mike Swift, Hank Levy, et. all, SOSP’03]
Nooks architecture

Each driver is wrapped in a layer of protective software that monitors all interactions between the driver and the kernel.
Nooks

• Goals:
  • protect the kernel against driver failures
  • recover automatically from driver failures
  • as few changes as possible to the drivers and kernel
Nooks techniques

- Isolation
- Interposition
- Recovery
Nooks techniques

• Isolation

  • *lightweight kernel protection domain* is a module that
  • executes in kernel mode
  • is logically part of the kernel
  • has read access to kernel structures
  • has *restricted* write access to kernel structures
Nooks techniques

- Interposition

  - each driver class exports an interface
  
  - wrappers for both exported and imported functions
    
    - some automatically generated
    
    - 455 wrappers: 329 for the functions exported by the kernel

- when a driver attempts to write a kernel object:

  - first, copy object to driver’s protection domain
Nooks techniques

- Recovery
  - user-mode recovery agent (consults configuration database)
  - in many cases enough just to release the resources held and restart the driver
  - shadow drivers are used to allow applications to continue after the crash
Nooks limitations

• can catch 99% of fatal driver errors and 55% of the non fatal ones
• drivers can execute privileged instructions
• wrappers themselves can contain bugs
• drivers can re-enable access to all memory
Microkernels

- directly attack the core of the problem: having the entire OS running as a huge binary in kernel mode
Minix architecture

A tiny kernel runs in kernel mode with the rest of the OS running as a collection of fully-isolated user-mode server and driver processes.
Another conservative approach

- Termite-1, today’s talk
- Termite-1 generates bug-free drivers
  - push-button synthesis
- Termite-2 [OSDI’14]
  - user-guided synthesis
- “the first tool to combine the power of automation with the flexibility of conventional development” - what about SKETCH? [ASPLOS’06]