MODERN SYSTEMS: 
EXTENSIBLE KERNELS AND CONTAINERS

CS6410 Hakim Weatherspoon
Motivation

- Monolithic Kernels just aren't good enough?
  - Conventional virtual memory isn't what userspace programs need (Appel + Li '91)
  - Application-level control of caching gives 45% speedup (Cao et al '94)
  - Application-specific VM increases performance (Krueger '93, Harty + Cheriton '92)
  - Filesystems for databases (Stonebraker '81)
  - And more...
Motivation

- Lots of problems...
Motivation

- Lots of problems... Lots of design opportunities!
Motivation

- Extensibility
- Security
- Performance

Can we have all 3 in a single OS?

From Stefan Savage’s SOSP 95 presentation
Context for these papers

- **1990’s**
  - Researchers (mostly) were doing special purpose OS hacks
  - Commercial market complaining that OS imposed big overheads on them
  - OS research community began to ask what the best way to facilitate customization might be. In the spirit of the Flux OS toolkit...

- **2010’s**
  - containers: single-purpose appliances
  - Unikernels: (“sealable”) single-address space
    - Compile time specialized
1988-1995: lots of innovation in OS development

- Mach 3, the first “true” microkernel
- SPIN, Exokernel, Nemesis, Scout, SPACE, Chorus, Vino,
- Amoeba, etc...
- And even more design papers
Motivation

- Exploring new spaces
  - Distributed computing
  - Secure computing
  - Extensible kernels (exokernel, unikernel)
  - Virtual machines (exokernel)
  - New languages (spin)
  - New memory management (exokernel, unikernel)
Exokernel

- Dawson R. Engler, M. Frans Kaashoek and James O’Toole Jr.
- Engler’s Master’s Thesis.
- Follow-up publications on 1997 and 2002.
- Kaashoek later worked on Corey.

**Exokernel: An Operating System Architecture for Application-Level Resource Management**

Dawson R Engler, M Frans Kaashoek, James O'Toole Jr
Exokernels - Motivation

- Existing Systems offer fixed high-level abstractions which is bad
  - Hurt app performance (generalization – eg: LRU)
  - Hide information (eg: page fault)
  - Limit functionality (infrequent changes – cool ideas don’t make it through)
Motivation (cont.)

- Separate protection from management, mgmt in user space
- Apps should use domain specific knowledge to influence OS services
- Small and simple kernel — adaptable and maintainable
Exokernel

- Kernel only multiplexes hardware resources (Aegis)
- Higher-level abstractions in Library OS (ExOS)
- Secure binding, Visible resource revocation, Abort
- Apps link with the LibOS of their choice
OS Component Layout

Exokernel
Exokernel main ideas

- **Kernel**
  - Resource sharing, not policies

- **Library Operating System**
  - Responsible for the abstractions
    - IPC
    - VM
    - Scheduling
    - Networking
Lib OS and the Exokernel

- Lib OS (untrusted) can implement traditional OS abstractions (compatibility)
- Efficient (Lib OS in user space)
- Apps link with Lib OS of their choice
- Kernel allows LibOS to manage resources, protects LibOss
Exokernel vs Microkernels vs VM

- Exokernel defines only a low-level interface.
- A microkernel also runs almost everything on user-level, but has fixed abstractions.
- A VM emulates the whole machine, doesn’t provide direct access.
Design

- Application-level resource management
- Exports hardware resources
- Multiplexes access between processes
- Separates policy from management
  - avoid resource management!
What problems do we solve?

- High-level abstractions
  - Hurt application performance
  - Hide information
  - Limit functionality

- Existing monolithic kernels
  - Encourage stable (archaic) interfaces
  - Difficult to extend with modern techniques
How do we solve them: Design

- Secure bindings
- Downloading code
- Visible resource revocation
- The abort protocol
How do we solve them: Design

- Secure bindings
- Downloading code
- Visible resource revocation
- The abort protocol
Secure bindings

- Decouples authorization from use
- Authorize once, at “bind time”
- Use transferable “capabilities” to check access
- Cache bindings in-kernel to decrease binding frequency
  - Example: huge software-based TLB
How do we solve them: Design

- Secure bindings
- **Downloading code**
- Visible resource revocation
- The abort protocol
Downloading code

- Userspace application produces kernel space code
- Access checks at download time
- Code is verified before being run, with JIT for speed
How do we solve them: Design

- Secure bindings
- Downloading code
- **Visible resource revocation**
- The abort protocol
Visible resource revocation

- Revocation traditionally invisible (or transparent)
  - Expensive: have to save entire state

- Try visible instead!
  - Save only the state you need
  - Kernel gives you a few microseconds to do it
How do we solve them: Design

- Secure bindings
- Downloading code
- Visible resource revocation
- The abort protocol
Revocation: kernel asks process for resource
- “relinquish page 5 please”
- Process tracks state and returns resource

Abort: kernel demands resource
- “page 5 in 50 microseconds”
- Takes resource “by force”
- Invalidates credentials and bindings.
- Notifies library operating system
Exokernel

- DEC MIPS
- Aegis: actual exokernel
  - Processor
  - Physical memory
  - TLB
  - Exceptions, Interrupts
- ExOS: library operating system
  - Processes, IPC, Virtual Memory, Network protocols
Microbenchmark results

<table>
<thead>
<tr>
<th>Machine</th>
<th>OS</th>
<th>pipe</th>
<th>pipe’</th>
<th>shm</th>
<th>lrpc</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC2100</td>
<td>Ultrix</td>
<td>326.0</td>
<td>n/a</td>
<td>187.0</td>
<td>n/a</td>
</tr>
<tr>
<td>DEC2100</td>
<td>ExOS</td>
<td>30.9</td>
<td>24.8</td>
<td>12.4</td>
<td>13.9</td>
</tr>
<tr>
<td>DEC3100</td>
<td>Ultrix</td>
<td>243.0</td>
<td>n/a</td>
<td>139.0</td>
<td>n/a</td>
</tr>
<tr>
<td>DEC3100</td>
<td>ExOS</td>
<td>22.6</td>
<td>18.6</td>
<td>9.3</td>
<td>10.4</td>
</tr>
<tr>
<td>DEC5000</td>
<td>Ultrix</td>
<td>199.0</td>
<td>n/a</td>
<td>118.0</td>
<td>n/a</td>
</tr>
<tr>
<td>DEC5000</td>
<td>ExOS</td>
<td>14.2</td>
<td>10.7</td>
<td>5.7</td>
<td>6.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Machine</th>
<th>OS</th>
<th>Roundtrip latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC5000/125</td>
<td>ExOS/ASH</td>
<td>259</td>
</tr>
<tr>
<td>DEC5000/125</td>
<td>ExOS</td>
<td>320</td>
</tr>
<tr>
<td>DEC5000/125</td>
<td>Ultrix</td>
<td>3400</td>
</tr>
<tr>
<td>DEC5000/200</td>
<td>Ultrix/FRPC</td>
<td>340</td>
</tr>
</tbody>
</table>
ExOS Virtual Memory

<table>
<thead>
<tr>
<th>Machine</th>
<th>OS</th>
<th>dirty</th>
<th>prot1</th>
<th>prot100</th>
<th>unprot100</th>
<th>trap</th>
<th>appel1</th>
<th>appel2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC2100</td>
<td>Ultrix</td>
<td>n/a</td>
<td>51.6</td>
<td>175.0</td>
<td>175.0</td>
<td>240.0</td>
<td>383.0</td>
<td>335.0</td>
</tr>
<tr>
<td>DEC2100</td>
<td>ExOS</td>
<td>17.5</td>
<td>32.5</td>
<td>213.0</td>
<td>275.0</td>
<td>13.9</td>
<td>74.4</td>
<td>45.9</td>
</tr>
<tr>
<td>DEC3100</td>
<td>Ultrix</td>
<td>n/a</td>
<td>39.0</td>
<td>133.0</td>
<td>133.0</td>
<td>185.0</td>
<td>302.0</td>
<td>267.0</td>
</tr>
<tr>
<td>DEC3100</td>
<td>ExOS</td>
<td>13.1</td>
<td>24.4</td>
<td>156.0</td>
<td>206.0</td>
<td>10.1</td>
<td>55.0</td>
<td>34.0</td>
</tr>
<tr>
<td>DEC5000</td>
<td>Ultrix</td>
<td>n/a</td>
<td>32.0</td>
<td>102.0</td>
<td>102.0</td>
<td>161.0</td>
<td>262.0</td>
<td>232.0</td>
</tr>
<tr>
<td>DEC5000</td>
<td>ExOS</td>
<td>9.8</td>
<td>16.9</td>
<td>109.0</td>
<td>143.0</td>
<td>4.8</td>
<td>34.0</td>
<td>22.0</td>
</tr>
</tbody>
</table>

+ Fast Sys call.
- Half the time in look-up (vector).

Repeated access to Aegis STL and ExOS PageTable
Perspective

- Extensible kernels are actually fast.
- End-to-end arguments.
- Efficient implementations.
- Extensibility without loss of security or performance
  - Exokernels
    - Safely export machine resources
    - Decouple protection from management
Containers

- Grouping of processes
- Provide isolation between groups
- Containers cannot customize operating systems
- Isn’t this similar to the problem exokernels tried to solve?

Diagram:

- Containers
  - MySQL
  - Web Server

- Hypervisor
  - OS
Unikernel: Library Operating Systems for the Cloud

Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Ralraj Singh, Thomas Gazagneaire, Steven Smith, Steven Hand, and Jon Crowcroft

University of Cambridge, University of Nottingham, Citrix Systems Ltd, OCamlPro SAS

In Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems pg. 461–472.

Unikernel slides from Shannon Joyner
Unikernel = EXOKERNEL + CONTAINERs

- Run one application per virtual machine
- One process per application
- Everything compiled into a VM image
- Do not compile unused code
Unikernel

- Run directly on top of standard hypervisor
- Can run multiple unikernels on the same hypervisor
Mirage

- Produces unikernels
- Compiles OCaml code to Xen VM image
- 4 main components
  - Text + Data segment
  - Foreign Grants
  - Minor Heap
  - Major Heap
Text and Data

- OCaml Runtime
- PVBoot
  - Initializes VM
HEAP

- Minor Heap
  - Short lived values in VM
  - Fast
- Major Heap
  - Long lived values
Foreign Grants

- Used for VM communication
- Write data to a grant table
- Exchange table between

Unikernel, Figure 2
ApACHE BENCHMARK

- Mirage unikernel improvements result in better performance than having multiple cores

![Graph showing throughput comparison between different configurations]

Unikernel, Figure 2
Exokernel versus Unikernel

- Exokernel
  - All applications on same system
  - Poor isolation

- Unikernel
  - Single application per system
  - Better isolation
Next Time

- Read and write review:
  - *Xen and the Art of Virtualization*, Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield. *19th ACM symposium on Operating systems principles (SOSP)*, October 2003, page 164--177.
Next Time

- MP1 part 2 due Friday
- Project Survey Paper proposals due next week
- Presentation schedule
- Check website for updated schedule