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

Dawson R. Engler,
M. Frans Kaashoek, and
James O'Tool Jr.
M.I.T Laboratory for Computer Science
Cambridge, MA 02139, U.S.A

Presented by Jeffrey J. Weston
Abstractions are Bad

- Problems with monolithic kernel abstractions:
  - Applications are unable to use domain specific optimizations.
  - Changes to the implementation of existing abstractions is discouraged.
  - Restricts flexibility of application builders to add new abstractions since they must be added on top of existing abstractions.
Abstractions are Bad

- Problems with micro-kernel abstractions:
  - Untrusted applications are unable to define specialized IPC primitives because virtual memory and message passing services are implemented by the kernel and trusted servers.
  - Many abstractions, such as page table structures and process abstractions, cannot be modified in micro-kernels.
  - Many hardware resources in micro-kernels are encapsulated in heavy-weight servers that cannot be bypassed or modified.
What is an Exokernel?

- Similar to micro-kernel; limited functionality is built into the kernel.
- User applications have low-level access to hardware resources.
- Kernel provides support for applications to manage their own access to hardware resources.
Exokernel Design

- Exokernel consists of a thin layer that exports and multiplexes physical resources through a set of low-level primitives.
- Applications, using the low-level exokernel interface, implement higher level abstractions that can better meet the performance and functionality needs of the application.
Exokernel Design

- Exokernel uses three techniques to control access to machine resources:
  - Secure Bindings: Applications can securely bind to machine resources.
  - Visible Revocation: Applications can participate in the resource revocation protocol.
  - Abort Protocol: Exokernel can break the secure binding of non-responsive applications by force.
Resource Sharing: Processor

- Application can decide duration and frequency of their time slices.
- When a time slice is over, application event handler is called, allowing application to customize the context switch.
- Time taken during the event handler is borrowed from the next time slice.
- If event handler takes too long, a context switch is forced by the exokernel.
Application-Specific Handlers

- Exokernel provides support to download code to the kernel, called application-specific handlers (ASH).
- This provides two advantages:
  - Eliminates kernel crossings.
  - The execution time of downloaded code is bounded and can thus be run in cases where the application cannot be scheduled.
Application-Specific Handlers

- In the context of networking, ASHs are untrusted user-level message-handlers that are downloaded into the kernel.
- Made safe by:
  - Code Inspection
  - Sandboxing
- Message handlers are executed when a message arrives.
Application-Specific Handlers

- Abilities of ASH message-handlers:
  - Control where messages are copied in memory, eliminating intermediate copies.
  - Integrated layer processing, such as checksums.
  - Message initiation, allowing for low-level message replies.
  - Control initiation.
Aegis, an exokernel, was built using the exokernel architecture.

Aegis exports the following resources:
- Processor
- Physical Memory
- TLB
- Exceptions
- Interrupts
- Network
Exokernel Implementation

- ExOS is a library operating system built to run on top of Aegis.
- ExOS implements the following abstractions:
  - Processes
  - Virtual Memory
  - User-Level Exceptions
  - Various Interprocess Abstractions
  - Several Network Protocols
Performance: ASHs

<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>

Roundtrip latency of a 60-byte packet over Ethernet using ExOS with ASHs, ExOS without ASHs, Ultrix, and FRPC; times are in microseconds.
Performance: ASHs

Average round-trip latency with increasing number of active processes on receiver.
Performance: IPC

<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>

Time for IPC using pipes, shared memory, and LRPC on ExOS and Ultrix; times are in microseconds. Pipe and shared memory are unidirectional, while LRPC is bidirectional.
Performance: 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>

Time to perform virtual memory operations on ExOS and Ultrix; times are in microseconds. The times for appel1 and appel2 are per page.
Conclusion

- Low-level primitives can be implemented very efficiently.
- Low-level multiplexing of hardware resources can be handled efficiently.
- Traditional operating system abstractions can be implemented efficiently at the application level.
- Applications can make special purpose abstractions by merely modifying a library.
Acknowledgement

Tables and figures were adapted from "Exokernel: An Operating System Architecture for Application-Level Resource Management" by Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.