USER-LEVEL DEVICE DRIVERS

Gernot Heiser
National ICT Australia

October 2003
**WHY USER-LEVEL DEVICE DRIVERS?**

- Ease of development
- Maintainability
- Increased system stability/reliability by
WHY USER-LEVEL DEVICE DRIVERS?

- Ease of development
  - less restrictive programming model, arbitrary languages
  - availability of standard debugging and profiling tools
  - release schedule decoupled from kernel

- Maintainability

- Increased system stability/reliability by
**Why User-Level Device Drivers?**

- **Ease of development**
  - less restrictive programming model, arbitrary languages
  - availability of standard debugging and profiling tools
  - release schedule decoupled from kernel

- **Maintainability**
  - interfaces stable
  - interfaces enforced

- **Increased system stability/reliability by**
WHY USER-LEVEL DEVICE DRIVERS?

- **Ease of development**
  - less restrictive programming model, arbitrary languages
  - availability of standard debugging and profiling tools
  - release schedule decoupled from kernel

- **Maintainability**
  - interfaces stable
  - interfaces enforced

- **Increased system stability/reliability by**
  - running driver in unprivileged mode
  - encapsulating driver into address space
  - may not have to trust drivers?
OTHER MOTIVATIONS

- Microkernel work in general
  - code allowed in microkernel only if it *must* be in kernel
  - drivers need not be in kernel
    - drivers must *not* be in microkernel
**Other Motivations**

- **Microkernel work in general**
  - code allowed in microkernel only if it *must* be in kernel
  - drivers need not be in kernel
    - drivers must *not* be in microkernel

- **Mungi in particular**
  - single-address-space operating system developed at UNSW
  - implemented from scratch (on top of L4 microkernel)
  - requires device drivers
    - attempts at reusing drivers from Linux, BSD, OSkit failed
    - requires too much emulation of original environment
**CHALLENGES**

- Historically serious performance problems with microkernels
- Generally ended up moving code *into* the kernel
  - e.g., Mach, OSF/1, MacOS X
CHALLENGES

• Historically serious performance problems with microkernels

• Generally ended up moving code into the kernel
  ➔ e.g., Mach, OSF/1, MacOS X

• Some spectacular failures
  ➔ IBM Workplace OS: US$2G
## CHALLENGES

- Historically serious performance problems with microkernels
- Generally ended up moving code *into* the kernel
  - e.g., Mach, OSF/1, MacOS X
- Some spectacular failures
  - IBM Workplace OS: US$2G
- Device drivers are performance-critical
  - typically 50% performance loss with user-level drivers
  - results from doubling number of system calls
HOPE

- L4 microkernel (Liedtke, GMD, IBM, Karlsruhe) shown to be fast
  - 20 times performance of Mach
  - runs Linux as user-level server with $\approx 5\%$ performance degradation

- Microkernel technology has come of age
  - e.g., Mungi faster than Linux
HOPE

• L4 microkernel (Liedtke, GMD, IBM, Karlsruhe) shown to be fast
  ➔ 20 times performance of Mach
  ➔ runs Linux as user-level server with \( \approx 5\% \) performance degradation

• Microkernel technology has come of age
  ➔ e.g., Mungi faster than Linux

• Time to have another look at user-level drivers
**APPROACH**

- Clean driver interface based on Mungi component model

- Use domain-specific language
  - platform-independent specification of device registers
  - DSL compiler generates access functions (in C)
  - driver proper implemented in C
  - significant reduction of driver complexity
APPROACH

- Clean driver interface based on Mungi component model

- Use domain-specific language
  - platform-independent specification of device registers
  - DSL compiler generates access functions (in C)
  - driver proper implemented in C
  - significant reduction of driver complexity

- Driver framework not tied to particular OS
  - was ported to Linux as a matter of weeks
  - supports running unmodified Mungi drivers in Linux (kernel & user level)
**Approach**

- Clean driver interface based on Mungi component model

- Use domain-specific language
  - platform-independent specification of device registers
  - DSL compiler generates access functions (in C)
  - driver proper implemented in C
  - significant reduction of driver complexity

- Driver framework not tied to particular OS
  - was ported to Linux as a matter of weeks
  - supports running unmodified Mungi drivers in Linux (kernel & user level)

- Drivers presently available:
  - PCI chipsets, IDE disk, Gigabit Ethernet, video, serial, USB
Restricting DMA

- Some PCI chipsets contain an I/O MMU
  - maps PCI address space to physical memory
  - designed to support > 4 GB of RAM with 32-bit PCI
  - example: HP zx1, DEC Tsunami chipsets
Some PCI chipsets contain an I/O MMU

- maps PCI address space to physical memory
- designed to support > 4 GB of RAM with 32-bit PCI
- example: HP zx1, DEC Tsunami chipsets

Can use I/O MMU to restrict DMA

- device driver requests DMA mapping for memory region from chipset driver
- chipset driver validates access and sets up PCI mapping
- chipset driver revokes mapping when DMA buffer is released
**User-Level I/O Architecture — Mungu**

```
<table>
<thead>
<tr>
<th>Device Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_complete()</td>
</tr>
<tr>
<td>read()</td>
</tr>
<tr>
<td>write()</td>
</tr>
<tr>
<td>dma_get()</td>
</tr>
<tr>
<td>dma_release()</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>lookup()</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Naming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>OS Syscalls</th>
</tr>
</thead>
<tbody>
<tr>
<td>validate()</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>RAM Pager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Table</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Chipset Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>L4 Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Drivers P7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```
**User-Level I/O Architecture — Linux**

- **Generic IRQ Handler**
  - `pci_read_config()`
  - `read()`
  - `pci_map_sg()`
  - `pci_unmap_sg()`

- **Driver**
  - `pci_map()`
  - `pci_unmap()`
  - `pci_map_sg()`
  - `pci_unmap_sg()`

- **Client**
  - IPC or function calls

- **libpic**
  - `pci_read_config()`

- **Kernel**
  - User

- **usrdrv**
  - Driver

- **Architecture-dependent DMA support**
IMPLEMENTATION

- Driver framework & drivers originally implemented on Mungi
  - drivers can only execute at user level

- Framework was ported to Linux
  - Linux drivers can execute in-kernel or at user level

- Initial performance evaluation for IDE disk and Gigabit Ethernet
PERFORMANCE: DISK READ LATENCY (PRELIMINARY)

HP i2000 (Itanium 2, zx1 chipset) reading 64MB
Performance: IDE Disk (Preliminary)

CPU (%) vs. Throughput (MB/s) for different transfer sizes (kB) ranging from 1 to 1024 kB.

- **Linux orig**
- **Linux kernel**
- **Linux user**

For HP i2000 (Itanium 2, zx1 chipset) reading 64MB.
**Performance: Gigabit Ethernet (Preliminary)**

![Graph showing CPU usage, throughput, and packet size for different systems and conditions.](image)

- **CPU usage (%)** vs. **Packet size (bytes)**
- **Throughput (Mb/s)**
- Key: Linux orig, Linux kernel, Linux user, Mungi Local, Mungi Other

Drivers P12
DISCUSSION

• Disk results encouraging (insignificant overhead)
DISCUSSION

• Disk results encouraging (insignificant overhead)

• Network results presently inconclusive:
  ➔ Performance bug in vanilla Linux 2.5.64 makes fair comparison impossible
  ➔ Linux presently sees factor two performance degradation (for user-level)
DISCUSSION

• Disk results encouraging (insignificant overhead)

• Network results presently inconclusive:
  ➔ Performance bug in vanilla Linux 2.5.64 makes fair comparison impossible
  ➔ Linux presently sees factor two performance degradation (for user-level)
  ➔ Mungi performance degradation is < 20 %
DISCUSSION

- Disk results encouraging (insignificant overhead)

- Network results presently inconclusive:
  - Performance bug in vanilla Linux 2.5.64 makes fair comparison impossible
  - Linux presently sees factor two performance degradation (for user-level)
  - Mungi performance degradation is $< 20\%$

- To be done:
  - loop drivers back into file system
  - include TCP/IP stack
  - evaluate application-visible performance
DISCUSSION

- Disk results encouraging (insignificant overhead)

- Network results presently inconclusive:
  - Performance bug in vanilla Linux 2.5.64 makes fair comparison impossible
  - Linux presently sees factor two performance degradation (for user-level)
  - Mungi performance degradation is $< 20\%$

- To be done:
  - loop drivers back into file system
  - include TCP/IP stack
  - evaluate application-visible performance

- Device drivers are a critical test of microkernel approach
  - acceptable performance *might* be achievable
  - encapsulation of drivers has great potential for reliability