User-Level Device Drivers

Gernot Heiser
National ICT Australia

October 2003

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
    $\Rightarrow$ 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
- 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
- Time to have another look at user-level drivers

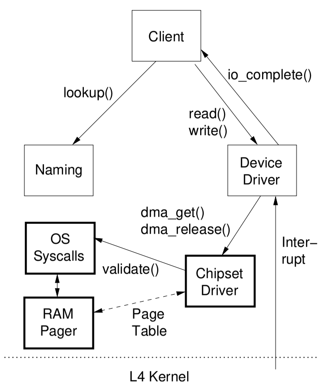
Approach

- 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
- 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

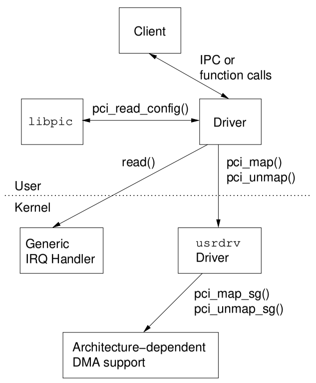
Restricting DMA

- 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
- 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 — Mungi



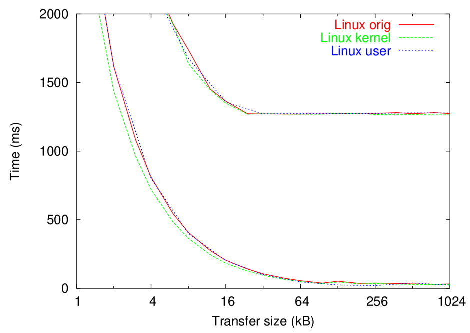
User-Level I/O Architecture — Linux



Implementation

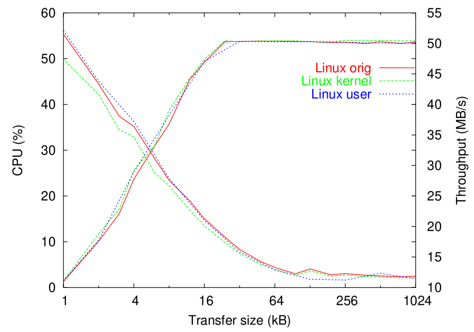
- Driver framework & drivers originally implemented on Mungi
  - ➜ drivers can only execute at user level
- Framework was ported to Linux
  - ➜ Linux drivers can execute in-kernel or at user level
- Initial performance evaluation for IDE disk and Gigabit Ethernet

Performance: Disk Read Latency (Preliminary)



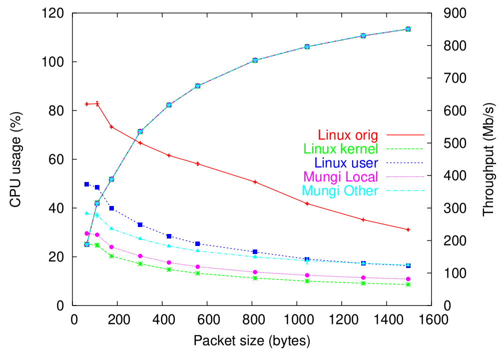
HP i2000 (Itanium 2, zx1 chipset) reading 64MB

Performance: IDE Disk (Preliminary)



HP i2000 (Itanium 2, zx1 chipset) reading 64MB

Performance: Gigabit Ethernet (Preliminary)



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