# Device Drivers

COMP9242
2011/S2 Week 5

NICTA Copyright 2010 From imagination to impact

## Some statistics

- 70% of OS code is in device drivers
  - 3,448,000 out of 4,997,000 loc in Linux 2.6.27
- A typical Linux laptop runs ~240,000 lines of kernel code, including ~72,000 loc in 36 different device drivers
- Drivers contain 3—7 times more bugs per loc than the rest of the kernel
- 70% of OS failures are caused by driver bugs

NICTA Copyright 2010 From imagination to impact

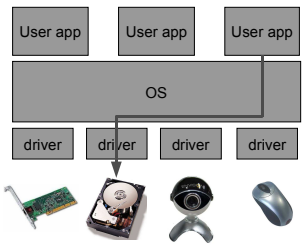
## Lecture outline

- Part 1: Introduction to device drivers
- Part 2: Overview of research on device driver reliability
- Part 3: Device drivers research at ERTOS

NICTA Copyright 2010 From imagination to impact

Part 1: Introduction to device drivers

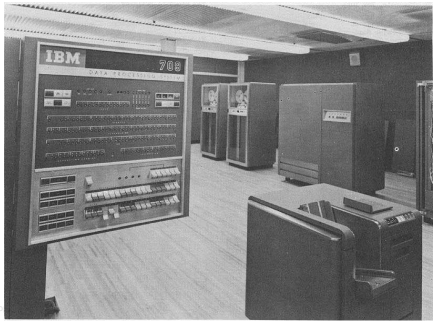
NICTA Copyright 2010 From imagination to impact




NICTA Copyright 2010 From imagination to impact

## OS archeology

The first (?) device drivers: I/O libraries for the IBM 709 batch processing system [1958]

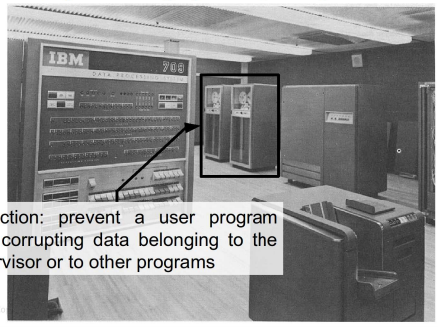



NICTA Co

## OS archeology

NICTA

The first (?) device drivers: I/O libraries for the IBM 709 batch processing system [1958]



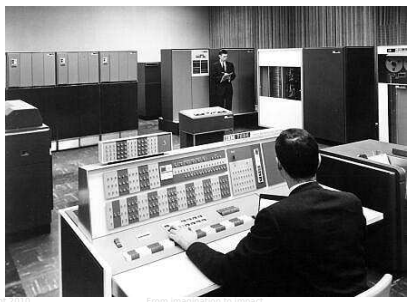

Protection: prevent a user program from corrupting data belonging to the supervisor or to other programs

NICTA Copyright 2010 From imagination to impact

## OS archeology

NICTA

IBM 7094 [1962] supported a wide range of peripherals: tapes, disks, teletypes, flexowriters, etc.

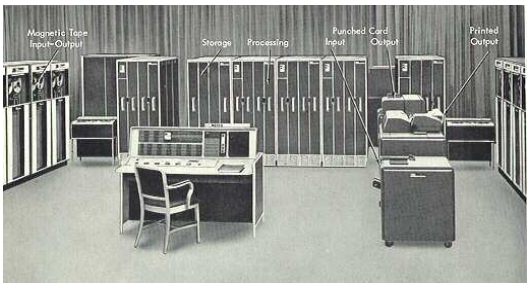


NICTA Copyright 2010 From imagination to impact

## OS archeology

NICTA

IBM 7090 [1959] introduced I/O channels, which allowed I/O and computation to overlap

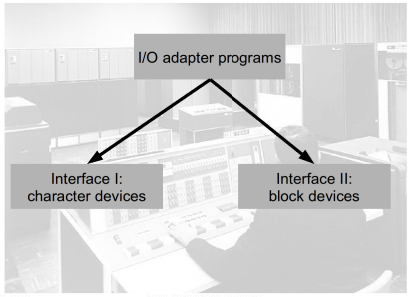



NICTA Copyright 2010 From imagination to impact

## OS archeology

NICTA

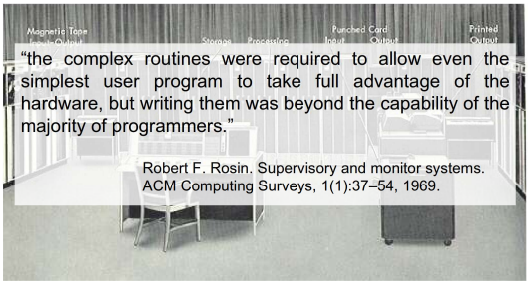
IBM 7094 [1962] supported a wide range of peripherals: tapes, disks, teletypes, flexowriters, etc.




## OS archeology

NICTA

IBM 7090 [1959] introduced I/O channels, which allowed I/O and computation to overlap



"the complex routines were required to allow even the simplest user program to take full advantage of the hardware, but writing them was beyond the capability of the majority of programmers."

Robert F. Rosin. Supervisory and monitor systems. ACM Computing Surveys, 1(1):37–54, 1969.

NICTA Copyright 2010 From imagination to impact

## OS archeology

NICTA

GE-635 [1963] introduced the master CPU mode. Only the hypervisor running in the master mode could execute I/O instructions

NICTA Copyright 2010

## Functions of a driver

- Encapsulation
  - Hides low-level device protocol details from the client
- Unification
  - Makes similar devices look the same
- Protection (in cooperation with the OS)
  - Only authorised applications can use the device
- Multiplexing (in cooperation with the OS)
  - Multiple applications can use the device concurrently

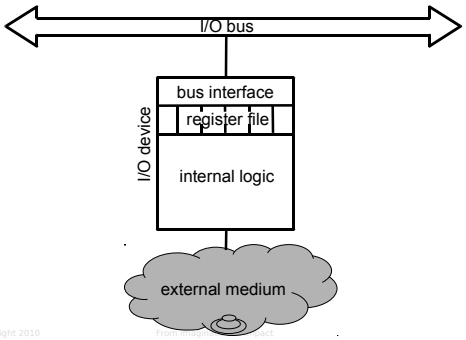
NICTA Copyright 2010 From imagination to impact

## PCI bus overview

- PCI bus
  - Conventional PCI
    - Developed and standardised in early 90's
    - 32 or 64 bit shared parallel bus
    - Up to 66MHz (533MB/s)
  - PCI-X
    - Up to 133MHz (1066MB/s)
  - PCI Express
    - Consists of serial p2p links
    - Software-compatible with conventional PCI
    - Up to 16GB/s per device

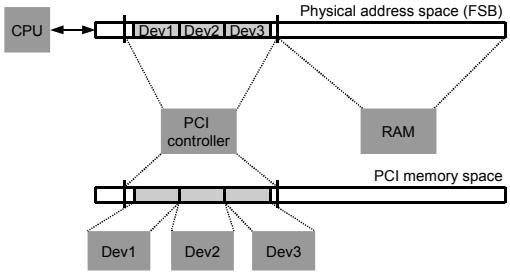
NICTA Copyright 2010 From imagination to impact

## I/O device: a high-level view



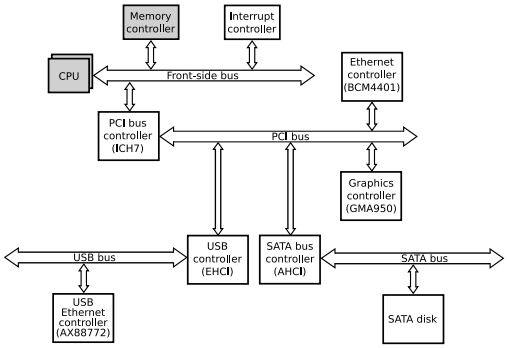

NICTA Copyright 2010 From imagination to impact

## PCI bus overview: memory space



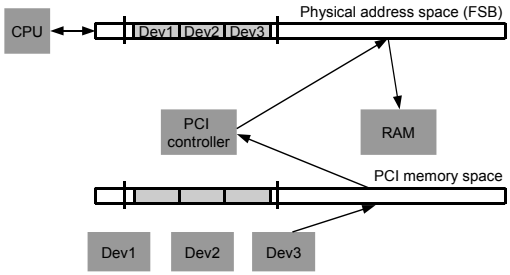

NICTA Copyright 2010 From imagination to impact

## I/O devices in a typical desktop system



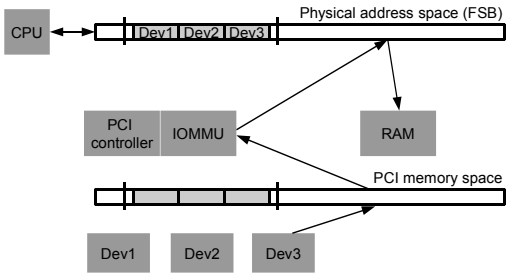

NICTA Copyright 2010 From imagination to impact

## PCI bus overview: DMA




NICTA Copyright 2010 From imagination to impact

## PCI bus overview: DMA




NICTA Copyright 2010 From imagination to impact

## PCI bus overview: config space

- PCI configuration space
  - Used for device enumeration and configuration
  - Contains standardised device descriptors

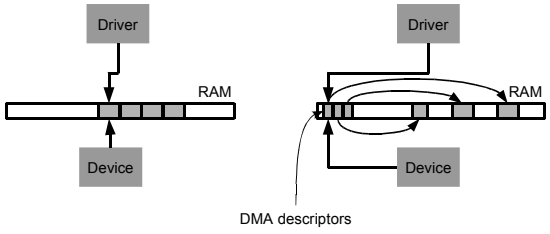
| 31 | 16 15 | | 0 | |
|---|---|---|---|---|
| Device ID | | Vendor ID | | 00h |
| Status | | Command | | 04h |
| Class Code | | | Revision ID | 08h |
| BIST | Header Type | Lat. Timer | Cache Line S. | 0Ch |
| Base Address Registers | | | | 10h<br>14h<br>18h<br>1Ch<br>20h<br>24h |
| Cardbus CIS Pointer | | | | 28h |
| Subsystem ID | | Subsystem Vendor ID | | 2Ch |
| Expansion ROM Base Address | | | | 30h |
| Reserved | | | Cap. Pointer | 34h |
| Reserved | | | | 38h |
| Max Lat. | Min Gnt. | Interrupt Pin | Interrupt Line | 3Ch |

NICTA Copyright 2010

## DMA descriptors

Permanent DMA mappings
- Set up during driver initialisation
- Data must be copied to/from DMA buffers

Streaming mappings
- Created for each transfer
- Data is accessed in-place



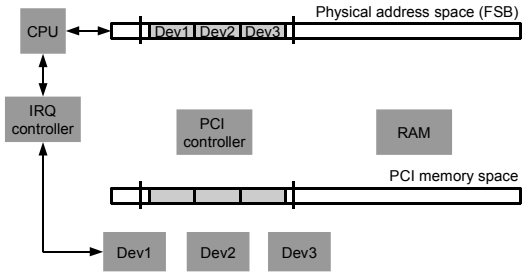

NICTA Copyright 2010 From imagination to impact

## PCI bus overview: I/O space

- I/O space
  - obsolete

NICTA Copyright 2010 From imagination to impact

## PCI bus overview: interrupts




NICTA Copyright 2010 From imagination to impact

## Writing a driver for a PCI device

- Registration
  - Tell the OS which PCI device ID's the driver supports
- Instantiation
  - Done by the OS when it finds a driver with a matching ID
- Initialisation
  - Allocate PCI resources: memory regions, IRQ's
  - Enable bus mastering
  - Permanent DMA mappings: disable caching

NICTA Copyright 2010 From imagination to impact

## Writing a driver for a PCI device

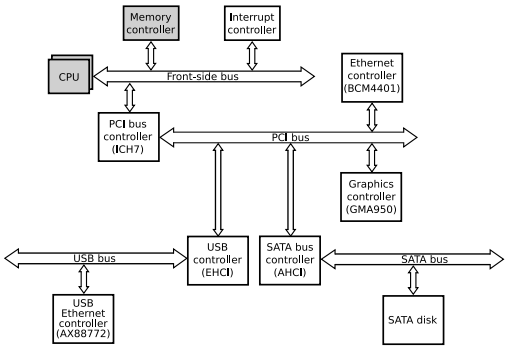
NICTA

- Interrupt handler
  - Return ASAP to re-enable interrupts; perform heavy-weight processing in a separate thread
- DMA
  - Streaming mappings: may require bounce buffers
- Power management
  - Prepare the device for a transition into a low-power state
  - Restore device configuration during wake-up

NICTA Copyright 2010 From imagination to impact

## I/O devices in a typical desktop system

NICTA




NICTA Copyright 2010 From imagination to impact

## USB bus overview

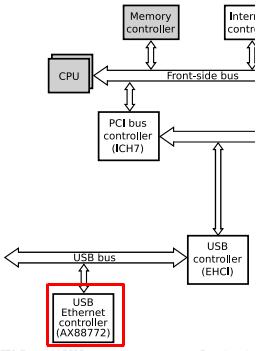
NICTA

- USB bus
  - Host-centric
  - Distributed-system-style architecture (packet-based transfers)
  - Hot plug
  - Power management
    - Bus-powered and self-powered devices
  - USB 1.x
    - Up to 12Mb/s
  - USB 2.0
    - Up to 480Mb/s
  - USB 3.0
    - Up to 4.8Gb/s

NICTA Copyright 2010 From imagination to impact

## Driver stacking

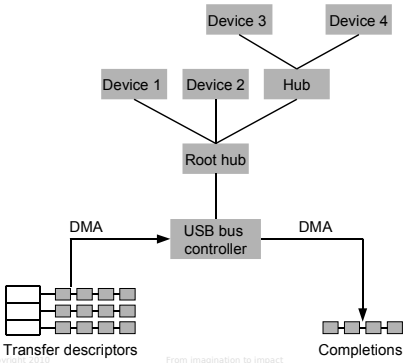
NICTA




NICTA Copyright 2010 From imagin

## USB bus overview

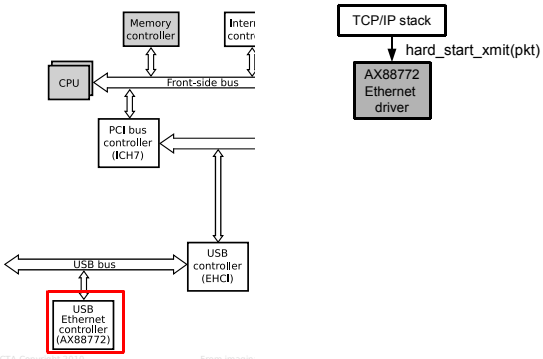
NICTA




NICTA Copyright 2010 From imagination to impact

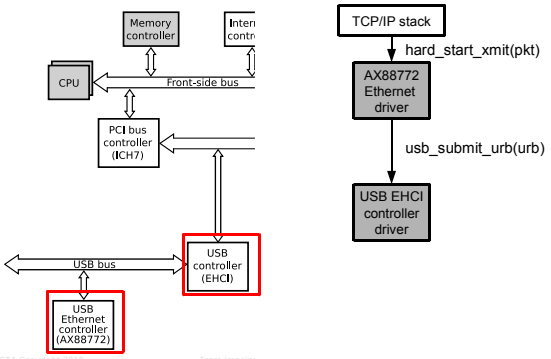
## Driver stacking

NICTA



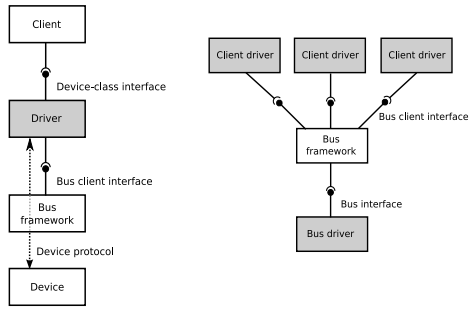

NICTA Copyright 2010 From imagin

## Driver stacking




NICTA Copyright 2010 From imagin

## Driver framework design patterns

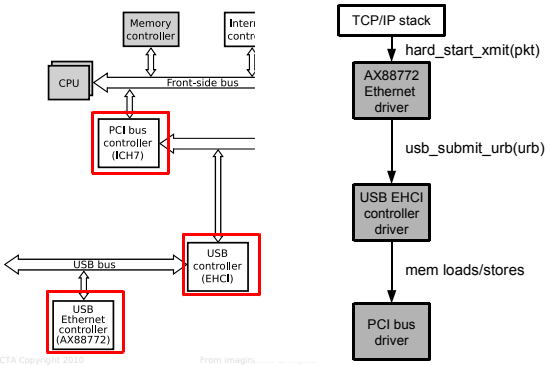



The driver pattern

The bus pattern

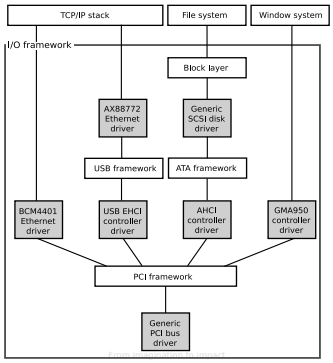
NICTA Copyright 2010 From imagination to impact

## Driver stacking



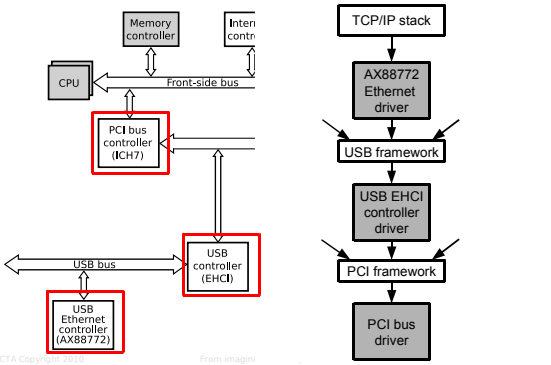

NICTA Copyright 2010 From imagin

## Driver framework software architecture




NICTA Copyright 2010 From imagination to impact

## Driver stacking




NICTA Copyright 2010 From imagin

Questions?

NICTA Copyright 2010 From imagination to impact

NICTA

Part 2: Overview of research on device driver reliability

NICTA Copyright 2010 From imagination to impact

## Understanding driver bugs

NICTA

- Driver failures
  - Memory access violations
  - OS protocol violations
    - Ordering violations
    - Data format violations
    - Excessive use of resources
    - Temporal failure
  - Device protocol violations
    - Incorrect use of the device state machine
    - Runaway DMA
    - Interrupt storms
  - Concurrency bugs
    - Race conditions
    - Deadlocks

NICTA Copyright 2010 From imagination to impact

## Some statistics

NICTA

- 70% of OS code is in device drivers
  - 3,448,000 out of 4,997,000 loc in Linux 2.6.27
- A typical Linux laptop runs ~240,000 lines of kernel code, including ~72,000 loc in 36 different device drivers
- Drivers contain 3—7 times more bugs per loc than the rest of the kernel
- 70% of OS failures are caused by driver bugs

NICTA Copyright 2010 From imagination to impact

## User-level device drivers

NICTA

- User-level drivers
  - Each driver is encapsulated inside a separate hardware protection domain
  - Communication between the driver and its client is based on IPC
  - Device memory is mapped into the virtual address space of the driver
  - Interrupts are delivered to the driver via IPC's

NICTA Copyright 2010 From imagination to impact

## Understanding driver bugs

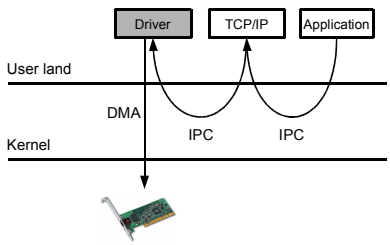
NICTA

- Driver failures

NICTA Copyright 2010 From imagination to impact

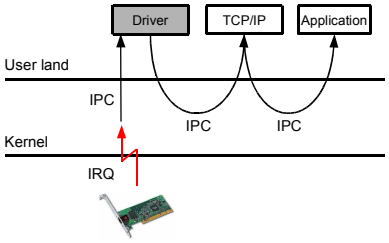
## User-level drivers in µ-kernel OSs

NICTA




NICTA Copyright 2010 From imagination to impact

## User-level drivers in µ-kernel OSs

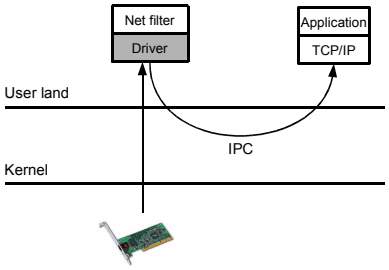



NICTA Copyright 2010 From imagination to impact

## Driver performance characteristics

NICTA Copyright 2010 From imagination to impact

## User-level drivers in µ-kernel OSs



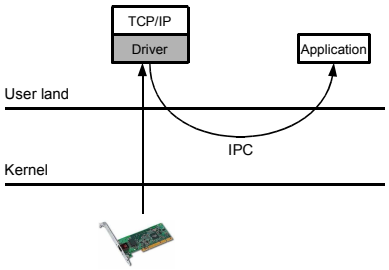

NICTA Copyright 2010 From imagination to impact

## Driver performance characteristics

- I/O throughput
  - Can the driver saturate the device?
- I/O latency
  - How does the driver affect the latency of a single I/O request?
- CPU utilisation
  - How much CPU overhead does the driver introduce?

NICTA Copyright 2010 From imagination to impact

## User-level drivers in µ-kernel OSs




NICTA Copyright 2010 From imagination to impact

## Improving the performance of ULD

NICTA Copyright 2010 From imagination to impact

## Improving the performance of ULD

- Ways to improve user-level driver performance
  - Shared-memory communication
  - Request queueing
  - Interrupt coalescing

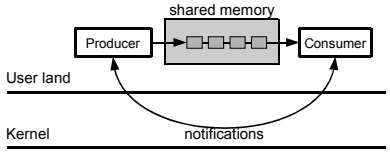
NICTA Copyright 2010 From imagination to impact

## Early implementations of ULD

- Michigan Terminal System [1970's]
  - OS for IBM System/360
  - Apparently, the first to support user-level drivers
- Mach [1985-1994]
  - Distributed multi-personality µ-kernel-based multi-server OS
  - High IPC overhead
  - Eventually, moved drivers back into the kernel
- L3 [1987-1993]
  - Persistent µ-kernel-based OS
  - High IPC overhead
  - Improved IPC design: 20-fold performance improvement
  - No data on driver performance available

NICTA Copyright 2010 From imagination to impact

## Implementing efficient shared-memory communication




- Issues:
  - Resource accounting
  - Safety
  - Asynchronous notifications

NICTA Copyright 2010 From imagination to impact

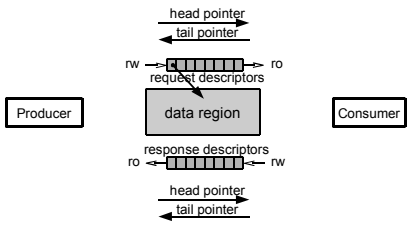
## More recent implementations

- Sawmill [~2000]
  - Multiserver OS based on automatic refactoring of the Linux kernel
  - Hampered by software engineering problems
  - No data on driver performance available
- DROPS [1998]
  - L4 Fiasco-based real-time OS
  - ~100% CPU overhead due to user-level drivers
- Fluke [1996]
  - ~100% CPU overhead
- Mungi [1993—2006]
  - Single-address-space distributed L4-based OS
  - Low-overhead user-level I/O demonstrated for a disk driver

NICTA Copyright 2010 From imagination to impact

## Rbufs

- Proposed in the Nemesis microkernel-based multimedia OS




NICTA Copyright 2010 From imagination to impact

## Currently active systems

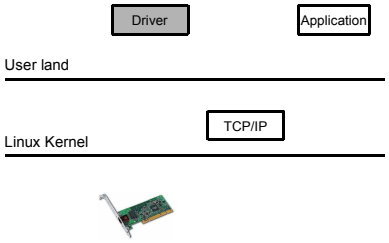
- Research
  - seL4
  - MINIX3
  - Nexus
- Commercial
  - OKL4
  - QNX
  - GreenHills INTEGRITY

NICTA Copyright 2010 From imagination to impact

## User-level drivers in a monolithic OS

NICTA

Ben Leslie et al. User-level device drivers: Achieved performance, 2005

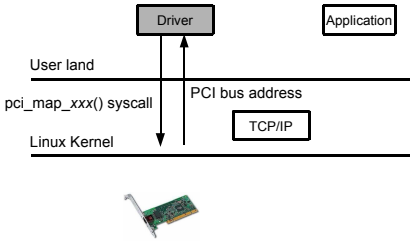



NICTA Copyright 2010 From imagination to impact

## User-level drivers in a monolithic OS

NICTA

Ben Leslie et al. User-level device drivers: Achieved performance, 2005

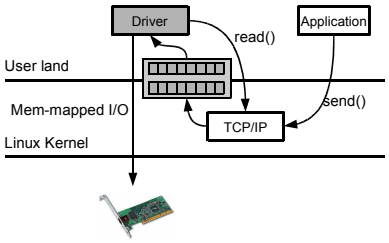



NICTA Copyright 2010 From imagination to impact

## User-level drivers in a monolithic OS

NICTA

Ben Leslie et al. User-level device drivers: Achieved performance, 2005




NICTA Copyright 2010 From imagination to impact

## User-level drivers in a monolithic OS

NICTA

Ben Leslie et al. User-level device drivers: Achieved performance, 2005

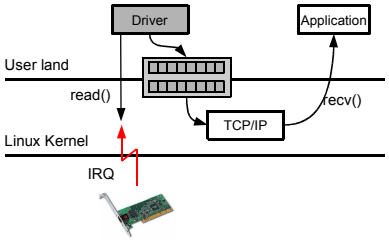
- Performance
  - Up to 7% throughput degradation
  - Up to 17% CPU overhead
  - Aggressive use of interrupt rate limiting potentially affects latency (not measured).

NICTA Copyright 2010 From imagination to impact

## User-level drivers in a monolithic OS

NICTA

Ben Leslie et al. User-level device drivers: Achieved performance, 2005

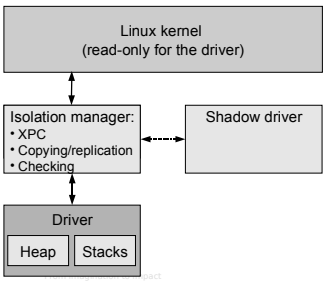



NICTA Copyright 2010 From imagination to impact

## Nooks

NICTA

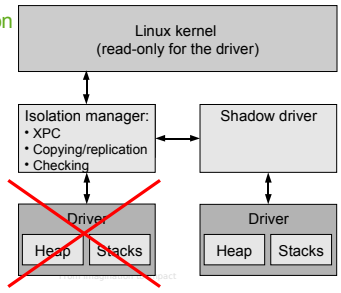
- A complete device-driver reliability solution for Linux:
  - Fault isolation
  - Fault detection
  - Recovery




NICTA Copyright 2010 From imagination to impact

## Nooks

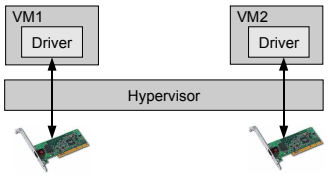
- A complete device-driver reliability solution for Linux:
  - Fault isolation
  - Fault detection
  - Recovery




NICTA Copyright 2010 From imagination to impact

## Virtualisation and user-level drivers

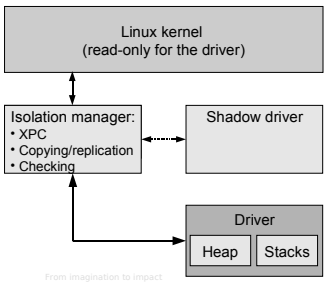
- Direct I/O




NICTA Copyright 2010 From imagination to impact

## Nooks

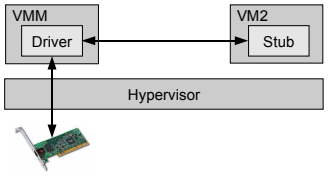
- A complete device-driver reliability solution for Linux:
  - Fault isolation
  - Fault detection
  - Recovery




NICTA Copyright 2010 From imagination to impact

## Virtualisation and user-level drivers

- Paravirtualised I/O



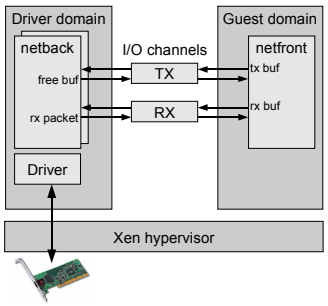

NICTA Copyright 2010 From imagination to impact

## Nooks

- A complete device-driver reliability solution for Linux:
  - Fault isolation
  - Fault detection
  - Recovery
- Problems
  - The driver interface in Linux is not well defined. Nooks must simulate the behaviour of hundreds of kernel and driver entry points.
- Performance
  - 10% throughput degradation
  - 80% CPU overhead

NICTA Copyright 2010 From imagination to impact

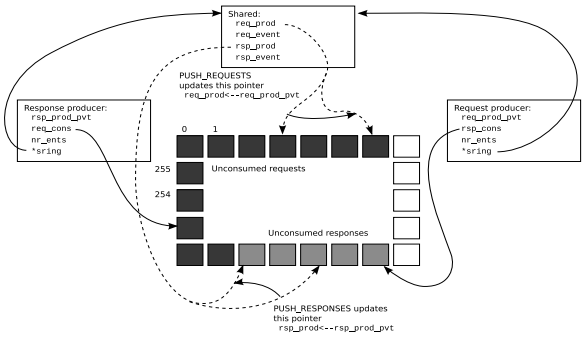
## Paravirtualised I/O in Xen




- Xen I/O channels are similar to rbufs, but use a single circular buffer for both requests and completions and rely on mapping rather than sharing

NICTA Copyright 2010 From imagination to impact

## Xen I/O channels




NICTA Copyright 2010 From imagination to impact

## Other driver reliability techniques

- Implementing drivers using safe languages
  - Singularity OS
    - The entire OS is implemented in Sing#
    - Every driver is encapsulated in a separate software-isolated process
    - Processes communicated via messages sent across channels
    - Sing# provides means to specify and statically enforce channel protocols

NICTA Copyright 2010 From imagination to impact

## Paravirtualised I/O in Xen

- Performance overhead of the original implementation: 300%
  - Long critical path (increased instructions per packet)
  - Higher TLB and cache miss rates (more cycles per instructions)
  - Overhead of mapping
- Optimisations
  - Avoid mapping on the send path (the driver does not need to "see" the packet content)
  - Replace mapping with copying on the receive path
  - Avoid unaligned copies
  - Optimised implementation of page mapping
  - CPU overhead down to 97% (worst-case receive path)

NICTA Copyright 2010 From imagination to impact

## Other driver reliability techniques

- Static analysis
  - SLAM, Blast, Coverity
  - Generic programming faults
    - Release acquired locks; do not acquire a lock twice
    - Do not dereference user pointers
    - Check potentially NULL-pointers returned from routine
  - Driver-specific properties
    - "if a driver calls another driver that is lower in the stack, then the dispatch routine returns the same status that was returned by the lower driver"
    - "drivers mark I/O request packets as pending while queuing them"
  - Limitations
    - Many properties are beyond reach of current tools or are theoretically undecidable (e.g., memory safety)

NICTA Copyright 2010 From imagination to impact

## Other driver reliability techniques

- Implementing drivers using safe languages
  - Java OSs: KaffeOS, JX
    - Every process runs in a separate protection domain with a private heap. Process boundaries are enforced by the language runtime. Communication is based on shared heaps.
  - House (Haskell OS)
    - Bare-metal Haskell runtime. The kernel and drivers are in Haskell.
    - User programs can be written in any language.
  - SafeDrive
    - Extends C with pointer type annotations enforced via static and runtime checking
    - `unsigned n;`
      `struct e1000 buffer * count(n) bufinfo;`

NICTA Copyright 2010 From imagination to impact

Questions?

NICTA Copyright 2010 From imagination to impact