HelenOS

CHARLES UNIVERSITY PRAGUE

faculty of mathematics and physics

Introduction

Martin Děcký

10th May 2006

HelenOS Project

Experimental development operating system

- http://www.helenos.eu/
- C, assembly
- Multiplatform
  - IA-32, IA-64, AMD64, MIPS (32), Sparc V9 (64), PowerPC (32, 64)
- SMP support
- “Monolithic micro-kernel”
- BSD license

Motivations

- Understand the design of an OS
  - From the bottom: synchronization, memory management, exceptions, linkage, booting, etc.
  - From the top: subsystems and interfaces design
  - Understand the whole system
- Testbed for experimental ideas
  - Easy to port, easy to enhance, easy to rewrite
  - Try to figure out new paradigms (files → objects, drivers → methods of tasks, etc.)
- Understand other interactions
  - Compilers, boot loaders, emulators/simulators

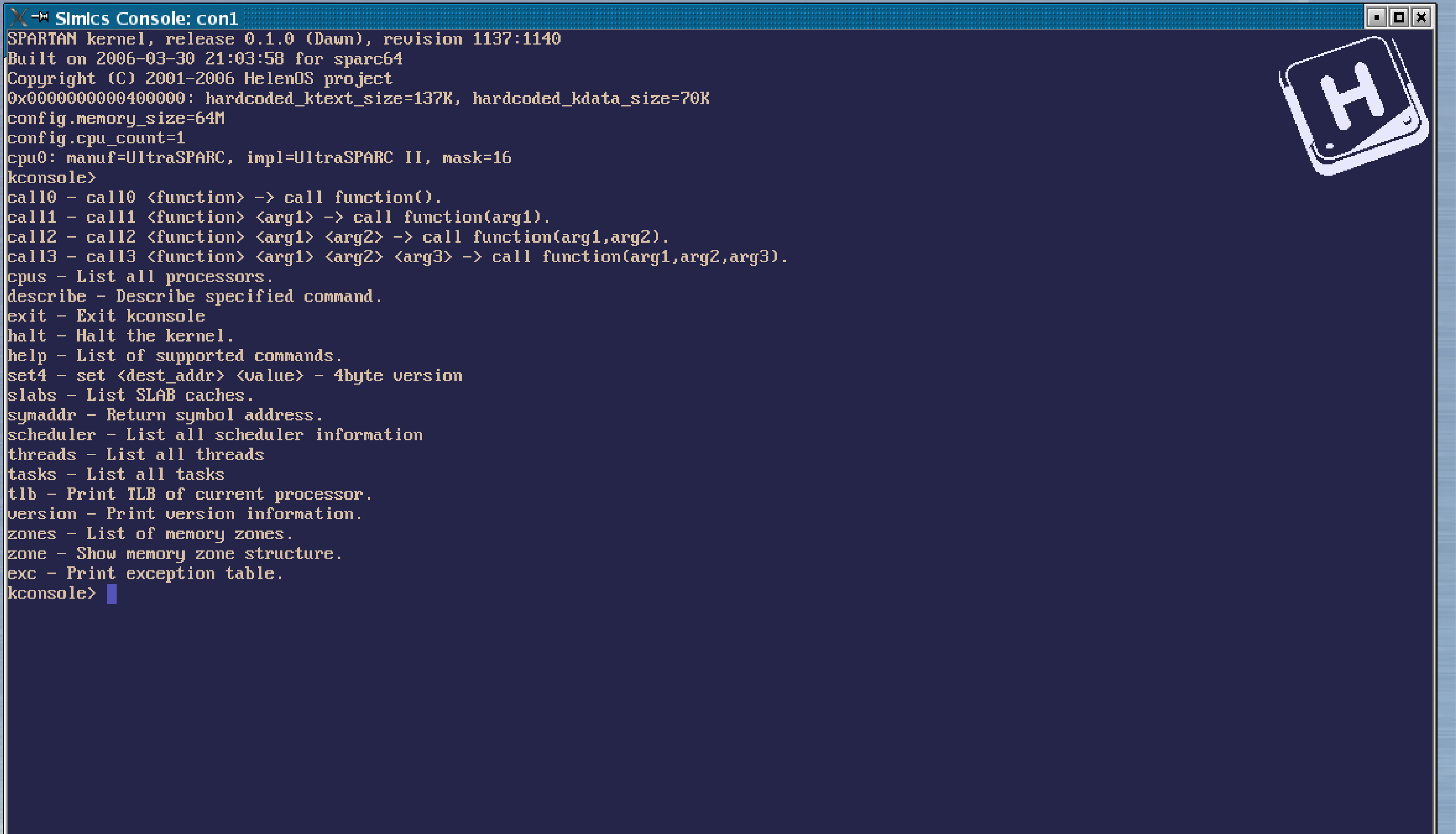
Brief History

- 2001 – 2004
  - SPARTAN kernel developed by Jakub Jermar (IA-32)
  - SMP support on IA-32
- Late 2003
  - Port of SPARTAN to MIPS
- Late 2004
  - A team software project at Faculty of Mathematics and Physics (six developers, one senior supervisor)
  - First specification
- 2005
  - Kernel work
    - Ports to IA-64, AMD64, Sparc and PowerPC

Current Status

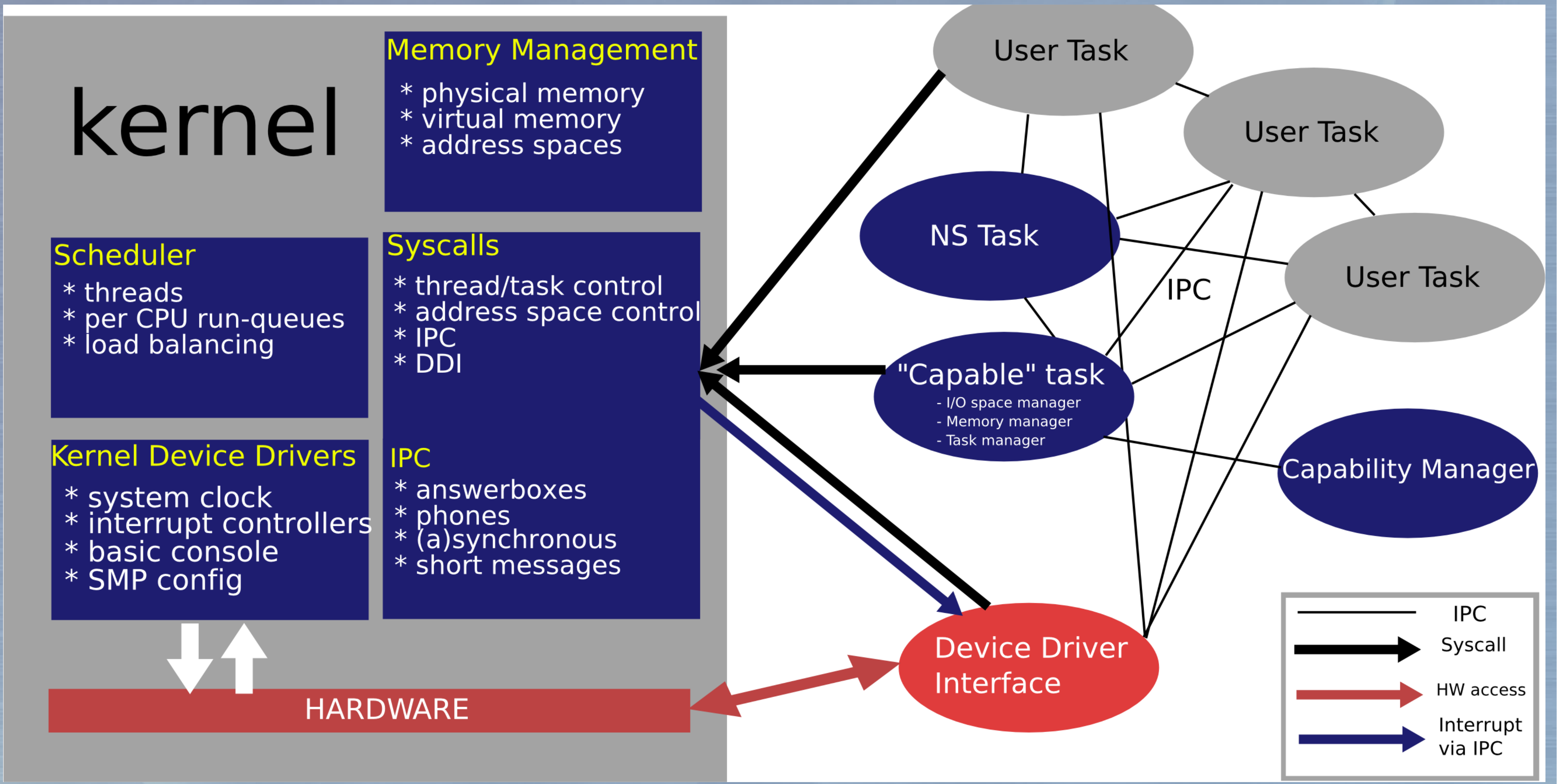
- Kernel
  - Full functionality according the specs on all platforms
  - Ability to host user space on all platforms
- User space
  - Preliminary syscall API, a few basic C functions
  - Support for kernel-managed threads and user-managed (pseudo) threads
  - IPC framework (messages, shared memory)
  - Preliminary user space driver interface

Current Status (2)



```
SPARTAN kernel, release 0.1.0 (Dawn), revision 1137:1140
Built on 2006-03-30 21:03:58 for sparc64
Copyright (C) 2001-2006 HelenOS project
0x0000000000400000: hardcoded_ktext_size=137K, hardcoded_kdata_size=70K
config.memory_size=64M
config.cpu_count=1
cpu0: manuf=UltraSPARC, impl=UltraSPARC II, mask=16
kconsole>
call0 - call0 <function> -> call function().
call1 - call1 <function> <arg1> -> call function(arg1).
call2 - call2 <function> <arg1> <arg2> -> call function(arg1,arg2).
call3 - call3 <function> <arg1> <arg2> <arg3> -> call function(arg1,arg2,arg3).
cpus - List all processors.
describe - Describe specified command.
exit - Exit kconsole
halt - Halt the kernel.
help - List of supported commands.
set4 - set <dest_addr> <value> - 4byte version
slabs - List SLAB caches.
symaddr - Return symbol address.
scheduler - List all scheduler information
threads - List all threads
tasks - List all tasks
tlb - Print TLB of current processor.
version - Print version information.
zones - List of memory zones.
zone - Show memory zone structure.
exc - Print exception table.
kconsole>
```

Architecture



kernel

Memory Management

* physical memory
* virtual memory
* address spaces

Scheduler

* threads
* per CPU run-queues
* load balancing

Syscalls

* thread/task control
* address space control
* IPC
* DDI

IPC

* answerboxes
* phones
* (a)synchronous
* short messages

Kernel Device Drivers

* system clock
* interrupt controllers
* basic console
* SMP config

Kernel Subsystems

- Physical memory management
  - Buddy system atop of frame zones (self-contained)
  - Slab allocator
- Virtual memory management
  - Generic interface for address space management
    - Page Table (4-level) instance, Global Hash Table instance
    - TLB interface
    - User address space divided into areas
- Time management
  - Preemptive scheduling
  - Generic timeout interface

Kernel Subsystems (2)

- Synchronization
  - Spin-lock
    - On non-SMP systems just disabling preemption
    - Some ability to detect deadlocks
  - Wait queue
    - Basic passive primitive, threads waiting for an event
  - Semaphore, mutex, condition variable, RW lock, futex
- Scheduler
  - Round-robin with multiple priority queues
  - Each CPU has his own queues, load-balancing thread
  - Lazy FPU context switching (if supported by HW)
  - Task management (common address space)

Kernel Subsystems (3)

- Interrupt/Exception handling mechanism
- Syscalls, IPC
- Device drivers interface, Capabilities control
  - Covered in detail later
- Minor subsystems
  - Boot infrastructure
  - Data structures
    - Bitmap, B+ tree, chained hash table, lists, fifo
  - ELF loader
  - String, sort functions, printf(), debug macros
  - Kernel symbol table
  - Kernel console
    - Mostly for debugging purposes

User Space

- libc
  - Basic standard C functions and types
    - Environment functions (__main, __exit, etc.)
    - malloc, free (atop of AS areas)
    - puts, printf and other I/O
    - memcpy, strlen, etc.
- HelenOS specific
  - Thread management
    - Kernel-managed & user-managed threads (psthreads)
  - Capabilities
  - Synchronization
    - Futexes
  - Softint, softfloat

IPC

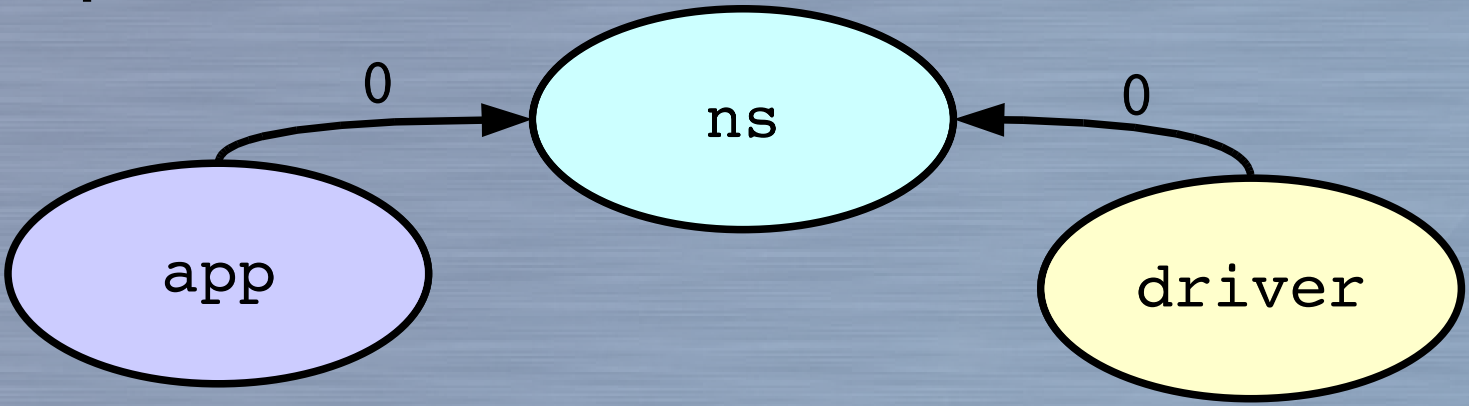
- Unidirectional communication
  - Phones
    - Identifies starting point (as file descriptor)
    - Phone 0 connected to Naming Service task
    - `call_sync`, `call_async`
  - Answerbox
    - Receives messages (`wait_for_call`)
      - 4 native integers (method, 3 arguments)
      - Answer expected by `answer` (return value, 3 arguments)
  - Synchronous messages
    - `call_sync` blocks
    - Returns the given answer

IPC (2)

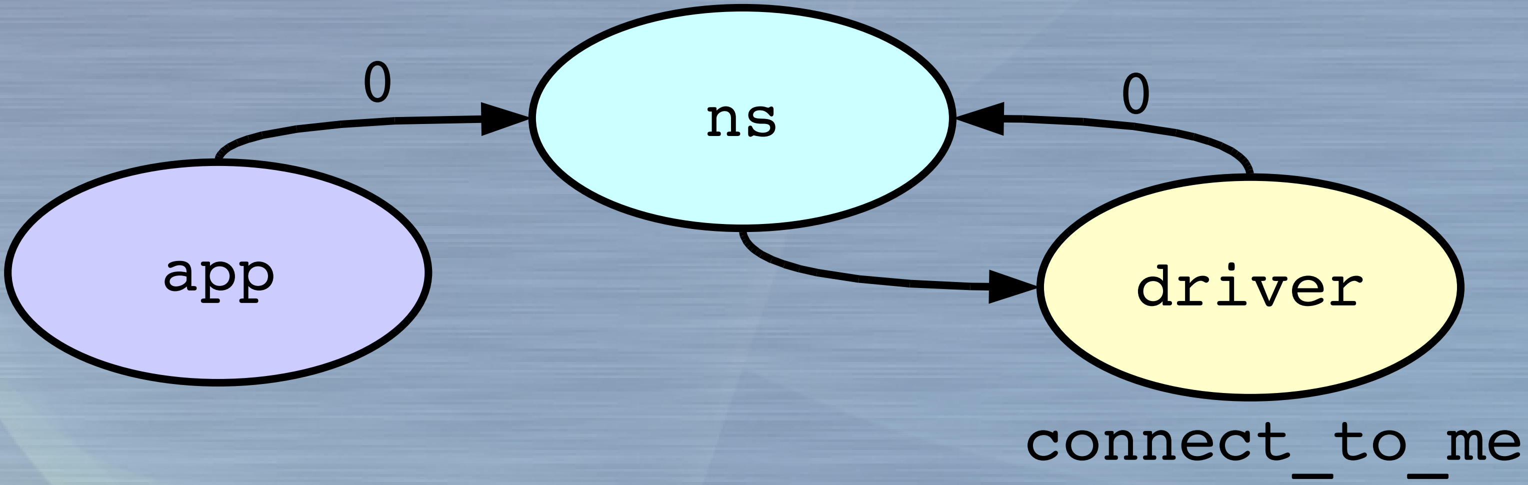
- Asynchronous messages
  - `call_async` never blocks
    - Fixed buffer in kernel, dynamic in user space
    - Registers callback
  - Answer received in `wait_for_call`
    - Answers have higher priority than calls
    - Runs callback
- Connections
  - `connect_me_to`
    - Client initiated connection
      - accept/refuse
      - forward (initially used by Naming Service)
  - `connect_to_me`
    - Server initiated connection

IPC (3)

- 1st phase

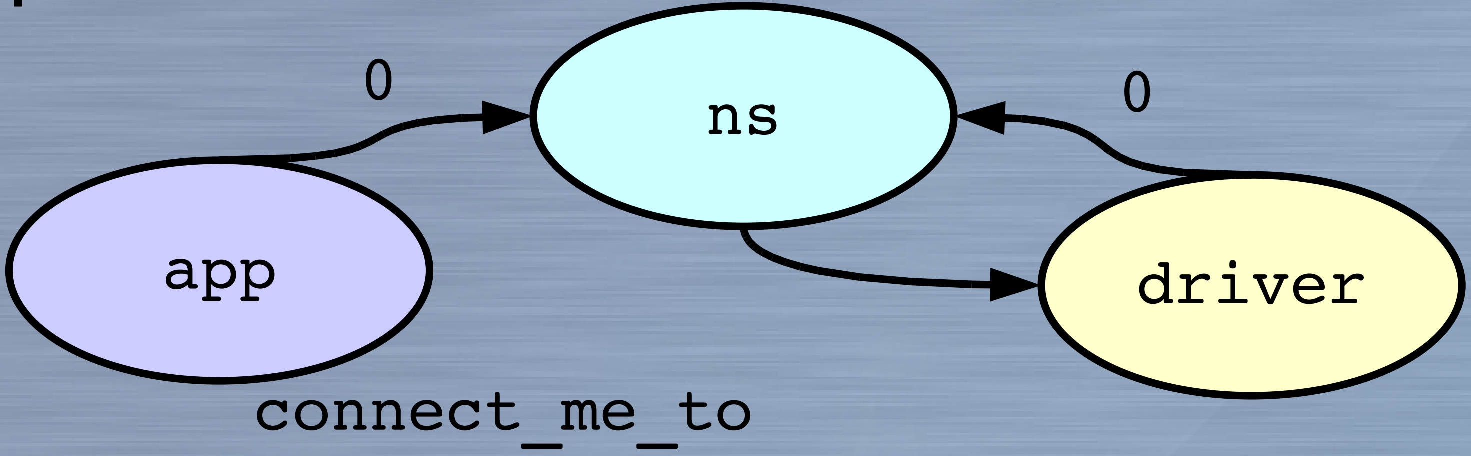


- 2nd phase

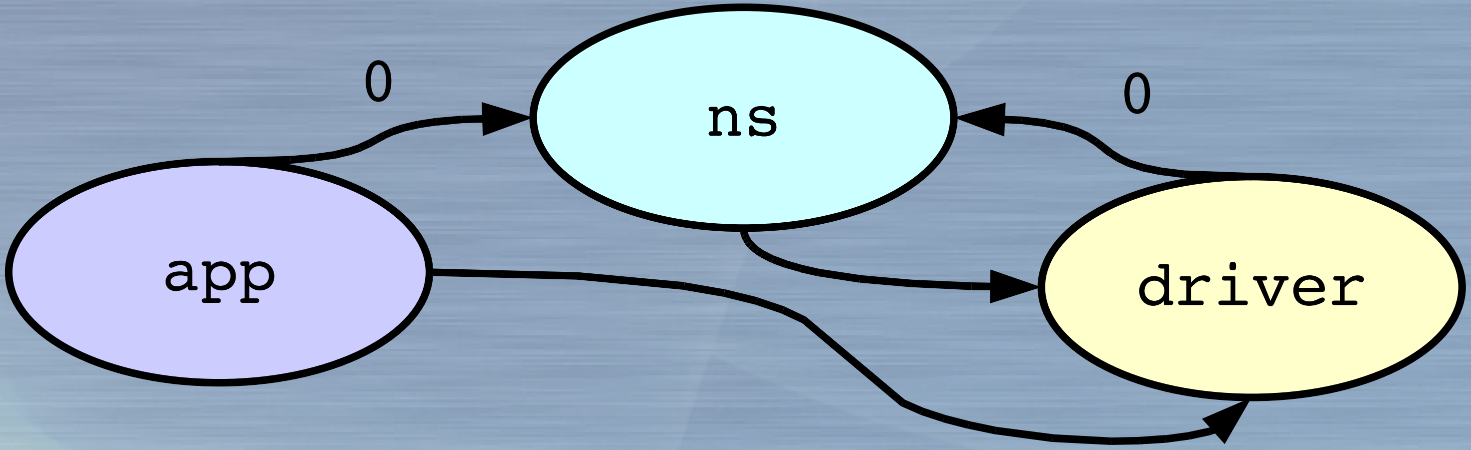


IPC (4)

- 3rd phase



- 4th phase



DDI

- User space hardware drivers
  - Task needs special capabilities
  - Map physical memory into AS
  - Map I/O space (mostly IA-32 specific)
  - Control preemption
  - Receive messages upon interrupt
    - Simple stateless language for handling level-triggered interrupts in kernel
  - Drivers and clients communicate using IPC
    - Keyboard driver
    - Framebuffer driver
    - Early PCI driver

Boot Process

- Hardware-dependent boot stages
  - Boot loader, loading of initial user space tasks into memory, bootstraping
- Hardware-dependent initialization
  - CPUs, memory, exceptions, interrupts, drivers, etc.
- Generic initialization
  - Buddy system, slab allocator
  - Main kernel thread, load-balancing thread
- Initial user space tasks
  - init (tests, capability manager)
  - ns (IPC naming service)
  - pci, fb (simple hardware drivers)

Near Future

- Finishing all missing bits in the ports
- Implement shutdown actions
- Stabilizing the DDI, useful drivers
  - Block devices
  - Read-only filesystem
- Implement more of libc
- First interactive user space programs
  - Shell
  - Tetris
- Kernel virtualization
  - Security contexts
  - XEN

Distant Future

- Major rewrite
  - Best way to evaluate gained knowledge
- Filesystem
- Component kernel
- Pure asynchronous IPC
  - Using threads and psthreads

To Sum Up

Every mistake in the computer industry gets made at least 3 times: once by the mainframe folks, once by minicomputer folks, and at least once by microprocessor folks.

– John Mashey