Design and Interfaces of a Device Service for L4 SDI OS

Felix Palmen <felix@palmen-it.de>
Alexander Roeckel

System Architecture Group

June 18th 2009

1. **Design goals**
2. **Device Manager**
3. **Drivers**
4. **Console**

Design goals

What should be achieved

- Central management of all hardware usage
- One driver thread per piece of hardware
- Generic interface between driver and device manager

Design constraints

- Keep the amount of required state low
- Use IPC economically
- Optimize performance

Design goals

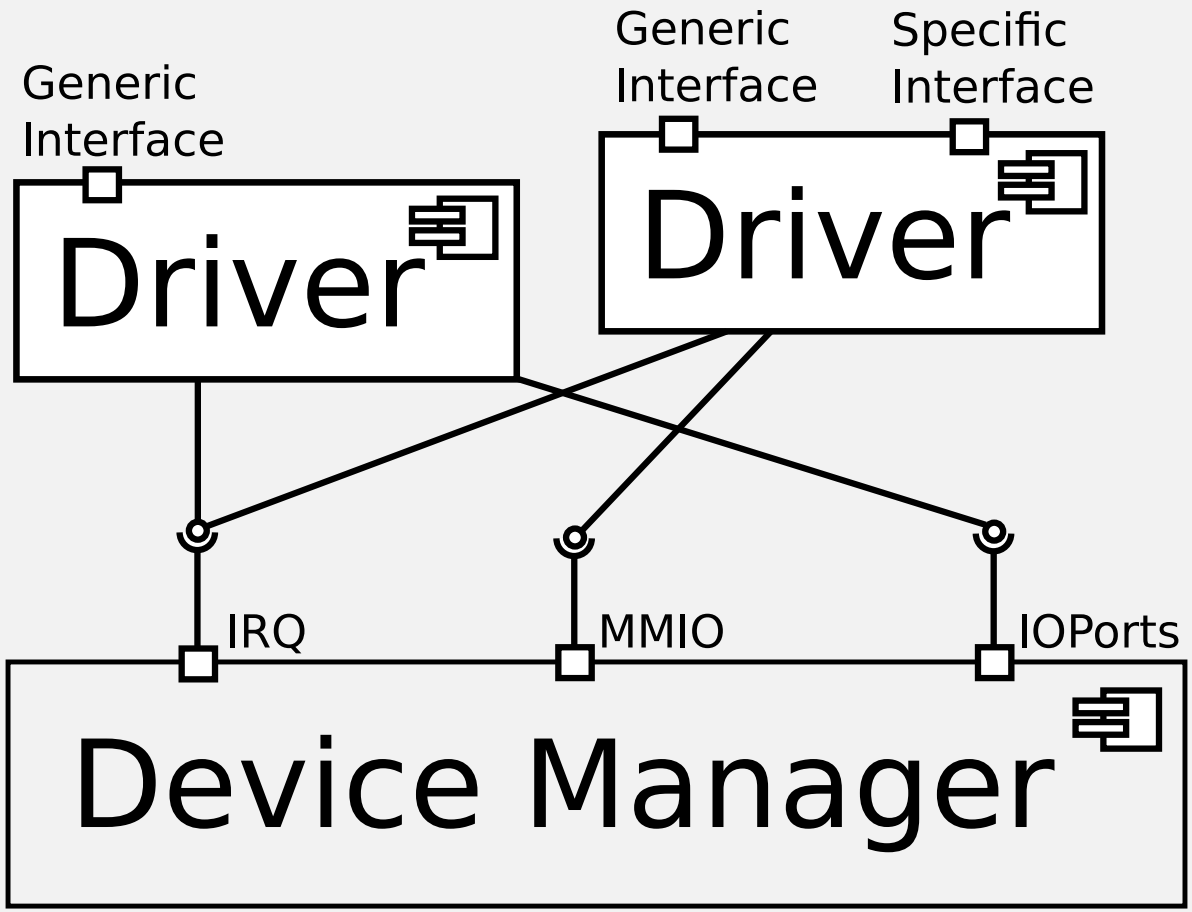
What should be achieved

- Central management of all hardware usage
- One driver thread per piece of hardware
- Generic interface between driver and device manager

Design constraints

- Keep the amount of required state low
- Use IPC economically
- Optimize performance

Component diagram



Registering Interrupts

- `registerInterrupt` reserves given IRQ (exclusively). Device Manager forwards IRQ IPC to driver thread.
- `releaseInterrupt` releases a previously registered IRQ.
- *Exceptions:*
  - *occupied* – IRQ already registered by other thread
  - *denied* – IRQ not registered by client
  - *invalid* – IRQ number does not exist

Interface "IRQ"

- `registerInterrupt(in short number, in short exclusive) raises (occupied, invalid);`
- `releaseInterrupt(in short number) raises (denied, invalid);`

Registering Interrupts

- `registerInterrupt` reserves given IRQ (exclusively). Device Manager forwards IRQ IPC to driver thread.
- `releaseInterrupt` releases a previously registered IRQ.
- *Exceptions:*
  - *occupied* – IRQ already registered by other thread
  - *denied* – IRQ not registered by client
  - *invalid* – IRQ number does not exist

Interface "IRQ"

- `registerInterrupt(in short number, in short exclusive) raises (occupied, invalid);`
- `releaseInterrupt(in short number) raises (denied, invalid);`

Registering Interrupts

- `registerInterrupt` reserves given IRQ (exclusively). Device Manager forwards IRQ IPC to driver thread.
- `releaseInterrupt` releases a previously registered IRQ.
- *Exceptions:*
  - *occupied* – IRQ already registered by other thread
  - *denied* – IRQ not registered by client
  - *invalid* – IRQ number does not exist

Interface "IRQ"

- `registerInterrupt(in short number, in short exclusive) raises (occupied, invalid);`
- `releaseInterrupt(in short number) raises (denied, invalid);`

Requesting MMIO access

- Device manager needs mappings for MMIO space directly from σ_0 (must know physical address).
- `requestMMIO` maps fpage to client containing a given MMIO address.
- `releaseMMIO` unmaps fpage

Interface "MMIO"

- `requestMMIO(in L4_Word_t base, in L4_Word_t size, out fpage page) raises (occupied, invalid);`
- `releaseMMIO(in L4_Word_t base) raises (denied, invalid);`

Requesting MMIO access

- Device manager needs mappings for MMIO space directly from σ_0 (must know physical address).
- `requestMMIO` maps fpage to client containing a given MMIO address.
- `releaseMMIO` unmaps fpage

Interface "MMIO"

- `requestMMIO(in L4_Word_t base, in L4_Word_t size, out fpage page) raises (occupied, invalid);`
- `releaseMMIO(in L4_Word_t base) raises (denied, invalid);`

Requesting MMIO access

- Device manager needs mappings for MMIO space directly from σ_0 (must know physical address).
- `requestMMIO` maps fpage to client containing a given MMIO address.
- `releaseMMIO` unmaps fpage

Interface "MMIO"

- `requestMMIO(in L4_Word_t base, in L4_Word_t size, out fpage page) raises (occupied, invalid);`
- `releaseMMIO(in L4_Word_t base) raises (denied, invalid);`

Requesting I/O Ports

- Device manager needs *iofpage* mapping for complete I/O AS
- `requestIOPort` maps iofpage to client
- `releaseIOPort` unmaps iofpage

Interface "IOPorts"

- `requestIOPort(in L4_Word_t base, in int size_bits, out iofpage page) raises (occupied, invalid);`
- `releaseIOPort(in L4_Word_t base) raises (denied, invalid);`

Requesting I/O Ports

- Device manager needs *iofpage* mapping for complete I/O AS
- `requestIOPort` maps iofpage to client
- `releaseIOPort` unmaps iofpage

Interface "IOPorts"

- `requestIOPort(in L4_Word_t base, in int size_bits, out iofpage page) raises (occupied, invalid);`
- `releaseIOPort(in L4_Word_t base) raises (denied, invalid);`

Requesting I/O Ports

- Device manager needs *iofpage* mapping for complete I/O AS
- `requestIOPort` maps iofpage to client
- `releaseIOPort` unmaps iofpage

Interface "IOPorts"

- `requestIOPort(in L4_Word_t base, in int size_bits, out iofpage page) raises (occupied, invalid);`
- `releaseIOPort(in L4_Word_t base) raises (denied, invalid);`

Requirements for DMA

- Mapping
  - Need physical address to setup DMA buffers in DMA controller
  - Pager for device mapper must know physical addresses
  - Pager could use 1:1 mappings from σ_0
  - Device manager could map pages to driver threads
- Programming the DMA controller
  - Driver must provide information for addressing the physical device in the DMA controller

Drivers (1)

Driver design

- One driver thread per device
- Driver registers with name service (no indirection)
- Driver provides generic interface to clients
  - Contains calls for character and block devices
  - Raises “unsupported” exceptions when call not applicable to device
  - Contains “ioctl” call for special commands
- Driver may provide additional specific interface

Drivers (1)

Driver design

- One driver thread per device
- Driver registers with name service (no indirection)
- Driver provides generic interface to clients
  - Contains calls for character and block devices
  - Raises “unsupported” exceptions when call not applicable to device
  - Contains “ioctl” call for special commands
- Driver may provide additional specific interface

Drivers (1)

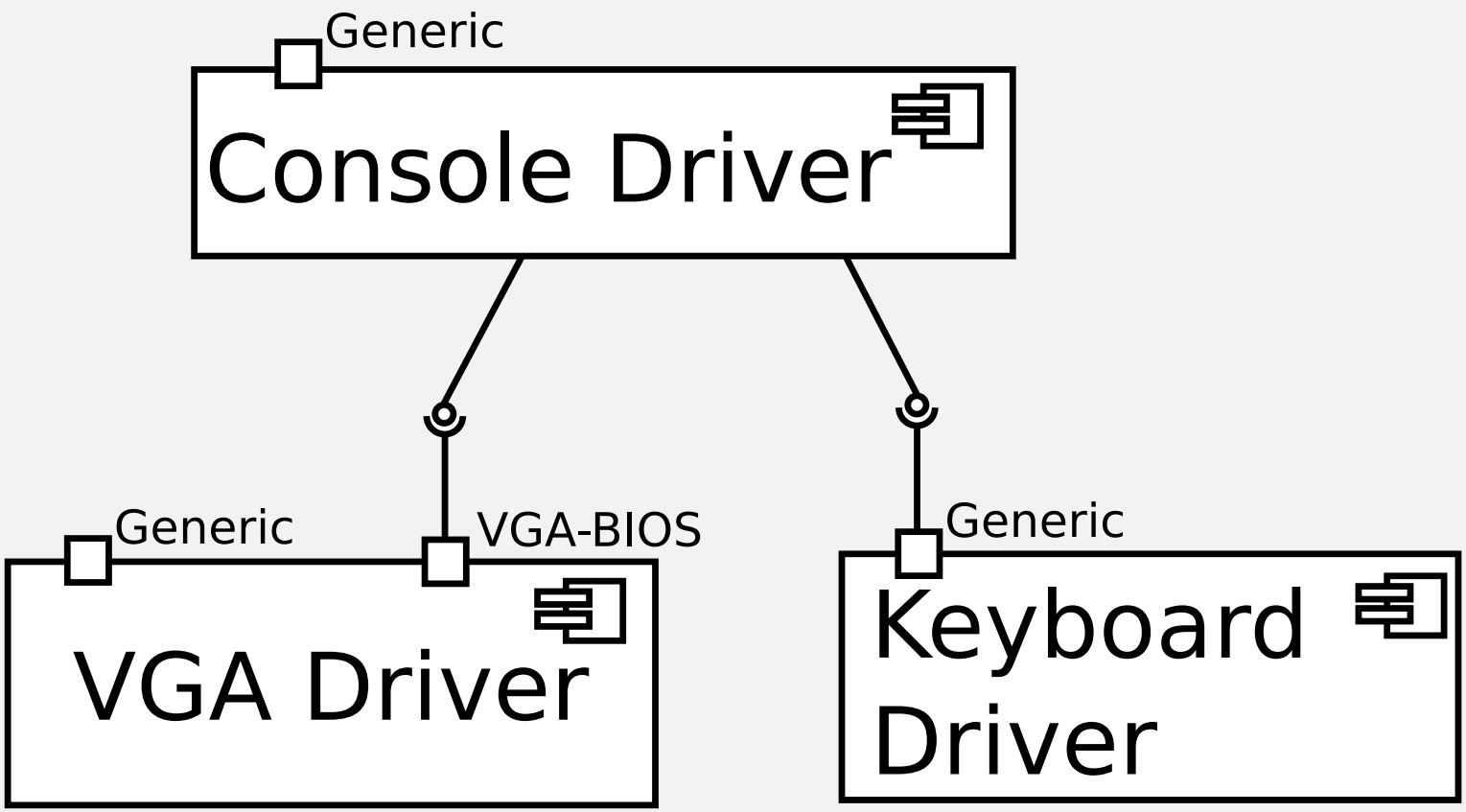
Driver design

- One driver thread per device
- Driver registers with name service (no indirection)
- Driver provides generic interface to clients
  - Contains calls for character and block devices
  - Raises “unsupported” exceptions when call not applicable to device
  - Contains “ioctl” call for special commands
- Driver may provide additional specific interface

Generic driver interface

- `read(out buffer_t data, inout L4_Word_t size) raises (unsupported);`
- `blockread(out buffer_t data, in L4_Word_t blockNumber, inout L4_Word_t size) raises (unsupported);`
- `write(in buffer_t data, inout L4_Word_t size) raises (unsupported);`
- `blockwrite(in buffer_t data, in L4_Word_t blockNumber, inout L4_Word_t size) raises (unsupported);`
- `ioctl(in L4_Word_t command, inout buffer_t data, inout L4_Word_t size) raises (unsupported, invalid);`

Component diagram



VGA Textmode

- 80x25 characters
- Two bytes per character: ASCII Code and display mode (color, blinking, ...)

Specific Interface "VGA-BIOS"

- `setChar(in short x, in short y, in short charCode, in short displayMode);`
- `getChar(in short x, in short y, out short charCode, out short displayMode);`
- `clearScreen();`
- `putString(in short x, in short y, in buffer_t data, in short displayMode);`

Console driver

- uses `read()` from generic interface of keyboard driver
- reads 2 bytes per call (scancode)
- uses specific interface of VGA driver to implement character device
- must remember cursor position
- must implement scrolling when reaching bottom end of screen
- provides generic interface, appears as a single read-/writable device to the outside world

Questions?