Complex Lab — Operating Systems
2014 Winter Term

Keyboard Device Driver &
Integration
Today's Goal

Pong Server

Paddle Client 1

Paddle Client 2

Keyboard Driver

Console

Memory Management

Moe

Sigma0

Fiasco Kernel
The PS/2 Keyboard

Source: http://wiki.osdev.org/PS2_Keyboard
Driving the keyboard

- Subscribe to interrupt 0x1

- On interrupt:
  - Read scan code from I/O port 0x60 (inb 0x60)
  - Translate scan code into key code and action

- That's it. Wrap a server interface around and you're done.
How to access an IO Port

Device Tree (x86—legacy.devs)

hw-root
{
  PS2 => new Device()
  {
    .hid = "PNP0303";
    new-res Io(0x60);
    new-res Io(0x64);
    new-res IRQ(1, 0x000000);
    new-res IRQ(12, 0x000000);
  }
}

TU Dresden, 09.01.15
How to access an IO Port

IO-Config file

```c
keyb => new System_bus()
{
    ps2dev => \
        wrap(hw-root.match("PNP0303"));
}
```
PKG cfg-File

local key = ld:new_channel();

ld:start(
    { caps = { icu = L4.Env.icu, sigma0 = ...
              keyb = key:svr()} },
    "rom/io rom/x86-legacy.devs rom/x86-fb.io");

ld:start(
    { caps = { icu = L4.Env.icu, vbus = key } },
    "rom/keyboard-server");
And the C-Code

```c
if ( l4io_request_ioport(0x64, 1) ||
    l4io_request_ioport(0x60, 1))
{
    printf("Requesting I/O failed\n");
    enter_kdebug("No keyboard");
}
```
• IRQs are bound to interrupt controllers (ICU)
  – HW ICU can be obtained using “icu=icu” in config file

• I/O library for managing I/O resources
  – Libio-direct – directly obtain resources from sigma0
  – No I/O manager involved – no security / management
  – Add “sigma0=sigma0” to config file

• l4/util/include/ARCH-x86/port_io.h
  – l4util_in<*>(), l4util_out<*>()

• l4/pkg/examples/interrupts → C version, C++ isn't hard either
Combine paddle client and keyboard server

• You have:
  – A keyboard server (last assignment)
  – A paddle client (l4/pkg/pong/examples)
    • Currently moving up and down

• Now:
  – Modify client to use keyboard input from your keyboard server
  – Play pong with two clients and different key settings
Graphical console multiplexing

- Enable your console server to switch between the pong console and your debug console

**Alternatives 1**
- Only console server has access to physical FB
- Clients get a virtual FB (== dataspace of the same size as the physical FB) and draw into it
- Console server periodically refreshes physical FB using memcpy from the currently active client FB

**Alternative 2**
- Active client directly renders into physical FB
- Inactive client(s) render into a virtual data space
- When switching active client, unmap all dataspaces and re-map physical/virtual FB data spaces

Elite Edition!
Console – Alternative 2

Client

Dataspace Server

Physical FB

Client
Console – Alternative 2

Client

Physical FB

Dataspace Server

Client
• Your server will need to implement a frame buffer interface as defined in `l4re/include/video/goos_fb`
  – you'll need to hand out a capability to a fb data space
  – read: an IPC gate that you'll use to handle all requests going to this DS

• Your virtual dataspace should implement the functions as defined in `l4re/include/dataspace`.

• You may also have a look at `l4re/util/include/dataspace_svr` for a nearly-complete data space server implementation.
1. User indicates client switch
2. Unmap physical framebuffer from client
3. Make client's FB point to a virtual copy
4. Unmap new client's virtual FB
5. Copy new client's virtual data into physical FB
6. Make new client's FB point to physical FB

• There is a race condition in there:
  – Between steps 2 + 3, the old client might draw, raise a page fault and get the physical pages mapped back
  – You'll need to handle this inside your implementation
Hand in everything until March 31st, 23:59:59 to your tutor.