COMPLEX LAB:
MICROKERNEL-BASED SYSTEMS

CARSTEN WEINHOLD
Today's Goal

- Pong Server
- Paddle Client 1
- Paddle Client 2
- Keyboard Driver
- Memory Management
- Console
- Sigma0
- Moe
- Fiasco

You build this!
PS/2 Keyboard Controller

http://wiki.osdev.org/"8042"_PS/2_Controller
Driving the Keyboard

- Subscribe to interrupt 0x1
- On interrupt:
  - Read scan code from I/O port 0x60 (inb 0x60)
  - Translate scan code to key code and action
- Wrap a server around it
- Connect your clients
Getting Access to the I/O Port

- Add to x86-legacy.devs (inside outer function)

```javascript
PS2 = Hw.Device(function()
    Property.hid = "PNP0303";
    Resource.iop1 = Res.io(0x60, 0x60); -- PS/2 device 1
    Resource.iop2 = Res.io(0x64, 0x64); -- PS/2 device 2
    Resource.irq1 = Res.irq(1, 0x000000);
    Resource.irq2 = Res.irq(12, 0x000000);
end);
```
Getting Access to the I/O Port

- The following is already in x86-fb.io (and probably shouldn't be called gui, feel free to rename)

```plaintext
Io.add_vbus("gui", Io.Vi.System_bus {
    ps2 = wrap(hw:match("PNP0[3F]??"));
})
```

- Then give IO a server cap (called gui) to a gate, and give the client cap to your keyboard server (called vbus)
Getting Access to the I/O Port

- For interrupts, look at pkg/examples/sys/isr (it's C, you can figure out the C++ interface)
- Request I/O port from vbus: `l4io_request_ioport(0x60, 1)`
- After you received an interrupt, read value from I/O port: `l4util_in8(0x60)`
Assignment 1: Keyboard Driver

- Build a working keyboard server
- You already have working pong clients in `src/l4/pkg/pong/examples`
- Modify the pong clients to be controllable by keyboard, with different controls
Frame Buffer Multiplexing

- Now there are two programs that can draw:
  - The pong game server
  - The console
- We need to multiplex graphics between the two programs
- One of them should render into physical frame buffer, while the other renders into a virtual frame buffer in memory
- You will need a **Dataspace** server that serves both clients
Frame Buffer Multiplexing

Frame Buffer Client 1
- Mapped FB

Frame Buffer Client 2
- Mapped FB

Frame Buffer Multiplexer
- Virtual FB
- Virtual FB

Physical FB
Frame Buffer Multiplexing

Frame Buffer Client 1
Mapped FB

Frame Buffer Client 2
Mapped FB

Frame Buffer Multiplexer
Virtual FB
Virtual FB

Physical FB
How to Switch Between Console Clients

1. User indicates a client switch
2. Unmap physical frame buffer from foreground client
3. Make this client's frame buffer point to a virtual copy
4. Unmap previously backgrounded client's virtual frame buffer
5. Copy this client's virtual screen into physical frame buffer
6. Make new foreground client use the physical frame buffer

- There is a race condition here:
  - Between steps 2+3 and 4+5, clients might draw, raise a page fault, and request the previously unmapped pages be mapped again
  - You will need to handle that in a sensible way
Handling Client State

- Your server will need to:
  - Hand out two capabilities to frame buffers (i.e., IPC gates that your server will listen on)
  - Implement the frame buffer interface as defined in src/l4/pkg/l4re-core/l4re/include/video/goos
  - Implement dataspaces as defined in src/l4/pkg/l4re-core/l4re/include/dataspace
  - Have a look at src/l4/pkg/l4re-core/l4re/util/include/dataspace_svr for a nearly complete Dataspace implementation
Assignment 2: Console Multiplexer

- Implement console switching, so that the user can play pong and switch to the console at any time.
- On real hardware you can't read pong's output: Edit `send_ipc()` in `pkg/pong/include/logging.h` to send all output to your log server.
Final Deadline

- Hand in full solution by **31.03.2019**
- Include information on how to use it (keyboard controls, etc.)