Lab Week 3

(due the night before Lab Week 5)

For this and the remaining labs you will be writing a small OS for the Raspberry 2 (which should also run unmodified on the Raspberry Pi Zero v1.3).

How? One step at a time. We'll begin with a Hardware Abstraction Layer (HAL)—like the one in Windows, but __*repeat_n_times(91, "much")*__ simpler!

Deliverables for this lab:

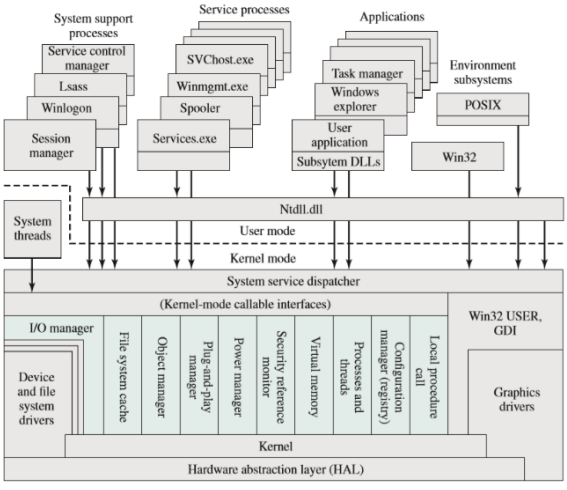
- (10%) A name for your OS
- A Hardware Abstraction Layer that supports:
    - (40%) *hal_io_serial_init*(), *hal_io_serial_putc*(serial id, character), and *hal_io_serial_getc*(serial id)
    - (20%) *hal_io_video_init*(), *hal_io_video_putpixel*(x, y, color)
    - (20%) *hal_io_video_putc*(x, y, color, character)
    - (10%) Linking *libgcc.a* to your Kernel (Don't worry too much if this seems difficult. It's supposed to be. That's why it's only 10%).

        (For full marks demo your new functionality on kernel startup)

        (Marks will be deducted for poor modularity and readability)
- (+10% Bonus) Run it on actual hardware (it's not terribly difficult, ask!)

Lab Theory (more details in Labs)

A HAL sits at the bottom of an OS (those who have one). See Window's HAL (Book page 102):



The HAL provides a) abstraction for upper layers of the kernel, and b) to an extent, portability.

This Week's Lab includes four files: boot.s, linker.ld, kernel.c, and video_sample.s. **Linker.ld** is a linker script and it describes how the different memory segments ought to be placed in memory. **boot.s** is the bootloader (more details in lab!), **kernel.c** is the entry point for you kernel, and **video_sample.s** is sample code that writes to the video screen the word "No OS".

To build everything you'll need to have **QEMU** installed. The latest 64-bit version will suffice (e.g. this one). You'll also need to have the **GNU ARM Embedded Toolchain.** The latest version will too suffice (here). Once you have everything set up, use these commands to build your kernel:

1. PATH_TO_GNU_TOOLS_ARM_EMBEDDED/bin/arm-none-eabi-as boot.s –o boot.o
2. PATH_TO_GNU_TOOLS_ARM_EMBEDDED/bin/arm-none-eabi-as video.s –o video.o
3. PATH_TO_GNU_TOOLS_ARM_EMBEDDED/bin/arm-none-eabi-gcc -mcpu=arm6 -fpic -ffreestanding -std=gnu99 -c kernel.c -o kernel.o -O0
4. PATH_TO_GNU_TOOLS_ARM_EMBEDDED/bin/arm-none-eabi-ld boot.o video_sample.o kernel.o -T linker.ld -o kernel.elf

At this point you should have a kernel.elf file. If you want to inspect binaries you can do, for instance:

1. PATH_TO_GNU_TOOLS_ARM_EMBEDDED/bin/arm-none-eabi-objdump –s kernel.elf

If you want to convert from ELF to IMG:

1. PATH_TO_GNU_TOOLS_ARM_EMBEDDED/bin/arm-none-eabi-objcopy kernel.elf –O binary kernel.img

To run everything on QEMU:

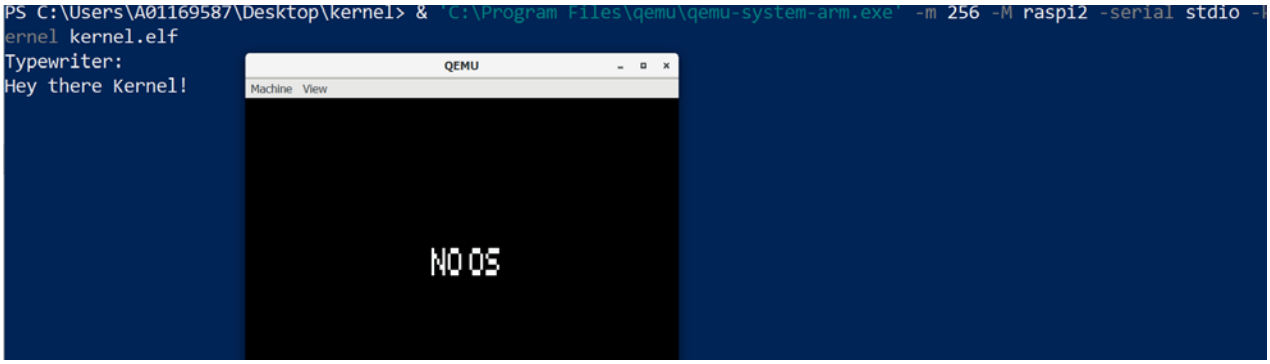
- PATH_TO_QEMU/qemu-system-arm -m 256 -M raspi2 -serial stdio –kernel PATH_TO_KERNEL/kernel.elf

As an example, this is how I do it on my Windows machine (from PowerShell):

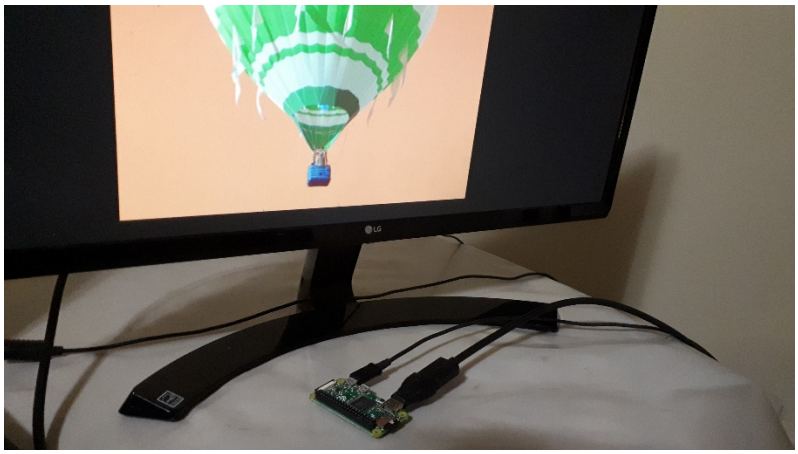
```
& 'C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin\arm-none-eabi-as.exe' boot.s -o boot.o

& 'C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin\arm-none-eabi-as.exe' video_sample.s -o video_sample.o

& 'C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin\arm-none-eabi-gcc.exe' -mcpu=cortex-a7 -fpic -ffreestanding -std=gnu99 -c kernel.c -o kernel.o -O0

& 'C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin\arm-none-eabi-ld.exe' boot.o video_sample.o kernel.o -T linker.ld -o kernel.elf

& 'C:\Program Files\qemu\qemu-system-arm.exe' -m 256 -M raspi2 -serial stdio -kernel kernel.elf
```

You probably want to make a Makefile or a script to automate this.

If everything went fine, you should now see something like this:



Now, after converting kernel.elf to kernel.img, the kernel failed to boot up properly in actual hardware, but [this other example](#) worked out of the box:

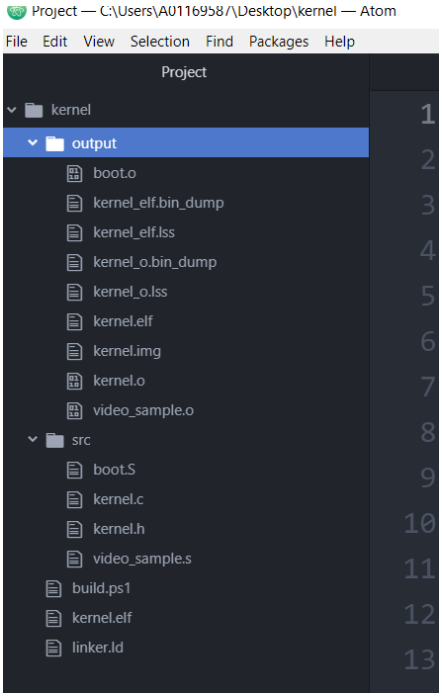


So it should be possible. It'll just need some thinking.

Feel free to use as much open source and as much stack overflow as needed (with proper acknowledgements)! You will when you go work in industry! Nothing useful can be written from scratch. Software is too complex nowadays!

Deliverables

- Your source code in a ZIP (or a link to source control if you prefer, as long as I don't need credentials to see it). DO NOT SEND CODE THAT DOES NOT COMPILE.
- *kernel.c* in a separate file **(outside of the ZIP)**
- In a PDF **(outside of the ZIP):**
    - Your OS's name
    - Screenshot(s) of your demo code (probably main() in *kernel.c)*
    - A screenshot of QEMU/PI running your demo.
      (Make sure your demo demonstrates every function you implemented)
    - A screenshot of the expanded view of the file structure for your submission. Example:



Resources on the web

- https://github.com/dwelch67/raspberrypi
- https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/index.html
- https://wiki.osdev.org/Raspberry_Pi_Bare_Bones

- https://wiki.osdev.org/Libgcc