Developing a MicroKernel for Raspberry Pi Zero

Presenters: Parker Folkman & Kemal Turksonmez

Project Scope and Goals

1. To more deeply understand the base structure of an OS implementation.
2. Implement a functional microkernel for Raspberry Pi Zero

What is a MicroKernel?

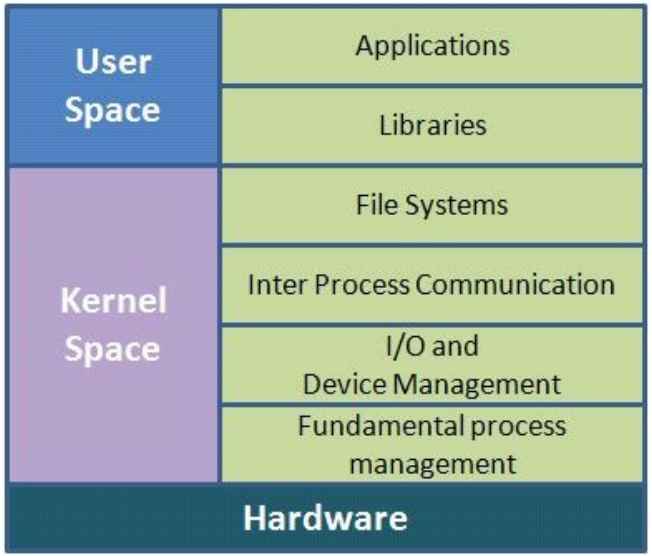
Definition: The bare minimum amount of software needed to control a computer's hardware. Abstracts away OS services to user space.

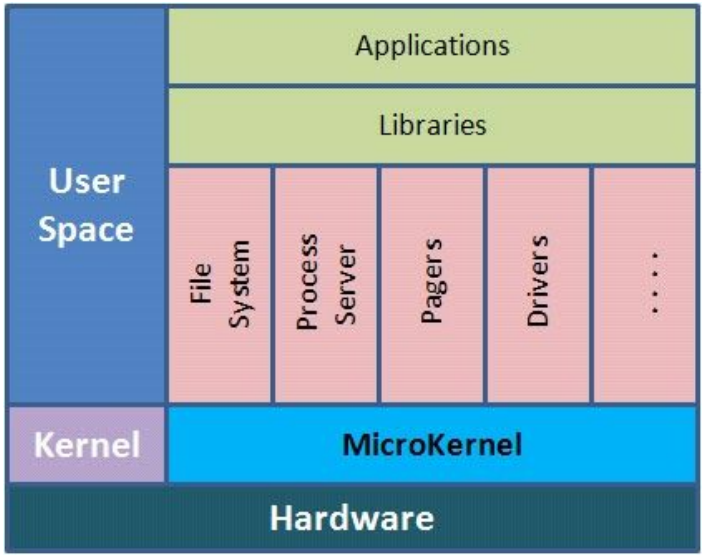
Composed of:

1. Bootloader
2. Process Scheduler
3. Memory Manager
4. IPC (Inter Process Communication)

What is a MicroKernel?

Monolithic vs MicroKernel





Images obtained from: http://linuxkernel51.blogspot.com/2011/02/difference-between-microkernel-and.html

Booting

- **boot.S** - Entry point for the kernel. This is the first piece of code executed on the hardware. Sets up the C environment.
- **kernel.c** - Sets up basic IO
- **linker.ld** - Script to link the kernel space files together into single executable
- Compile code into kernel image using the `gcc-arm-none-eabi` compiler
- Boot the image using Raspbian's bootcode.bin

Kernel Memory Management

- Implemented as an OS service module in user space
- Use simple paging
  - 4 kb pages
- Metadata for memory is found using atags.

```
void kernel_main(uint32_t r0, uint32_t r1, uint32_t atags)
```

Cross Compiling

- Cross Compilers are used to generate executables for systems that are not your own (ex. different CPU or OS)
- In our case, compiling a 32-bit ARMv6 architecture on an x86 machine
- Without a cross compiler, it would be impossible for us to develop the OS
- The cross compiler we’re using is the gcc arm-none-eabi

Resources

- https://jsandler18.github.io/tutorial/dev-env.html
- https://github.com/s-matyukevich/raspberry-pi-os
- https://wiki.osdev.org/Raspberry_Pi_Bare_Bones

Questions?