Introduction to Operating Systems

44-550: Operating Systems

What is an OS? Why is it Important?

Operating System

Software responsible for allocating and managing hardware resources in a computing system. An abstraction layer that simplifies access to computational hardware.

Consider ENIAC (the first general purpose computer, 1946). It was capable of running a single calculation by being reprogramed (physically reconfigured) to perform a single task. Programming ENIAC required rewiring parts of the machine and using switches to define ”functions” and ”subroutines” to complete a calculation. Initially, there were six people (Kay McNulty, Betty Jennings [Jean Jennings Bartik], Betty Snyder, Marlyn Meltzer, Fran Bilas, and Ruth Licterman) who could program ENIAC (though the team was expanded to 100 scientists later in its lifetime).

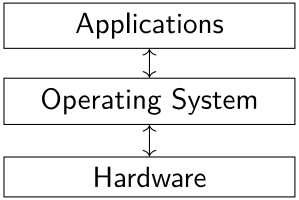
ENIAC

ENIAC is an example of a computer without an operating system. Access to these early computational systems was handled by physical limitations; need a calculation? Get in line! Need access to storage or program logic? We will need to physically reconfigure the system to handle this. Every program required the computer to be "rebuilt" (though it had enough commonly used operations built in that it could be "easily" reconfigured).

Fast forward to modern day computation, and we see devices with significantly more capable hardware than the earliest computers (the NES was more powerful than the computers that got humans to the moon!). Managing resources has become more complex than any single application can handle, and as such modern operating systems exist with the sole purpose of allocating and managing hardware so the myriad of applications we use on a regular basis can coexist on the same computer.

OS Overview

- Hardware: hurts when they are thrown at you.
    - CPU
    - Storage and Memory
    - ...
- Applications: The software we use to complete tasks
    - Browsers
    - Office software
    - Games
    - ...
- **Operating System**: Manages application access to hardware resources (and abstracts/simplifies this across different hardware of the same type).



Operating System Structures

An OS consists of a **kernel** and software that uses the functionality exposed by the kernel to perform tasks. These components run in different *memory spaces*:

- **Kernel Space** (or **Kernel Mode**)
  - memory locations the kernel executes from
  - processes that live in kernel space have no limits in accessing hardware and can execute any CPU instruction and access *any* memory location.
- **User Space** (or **User Mode, Userland**)
  - restricted memory space from which applications run
  - processes that live in userland cannot directly access hardware or kernel mode memory

We categorize Operating Systems based on where various system utilities live in memory.

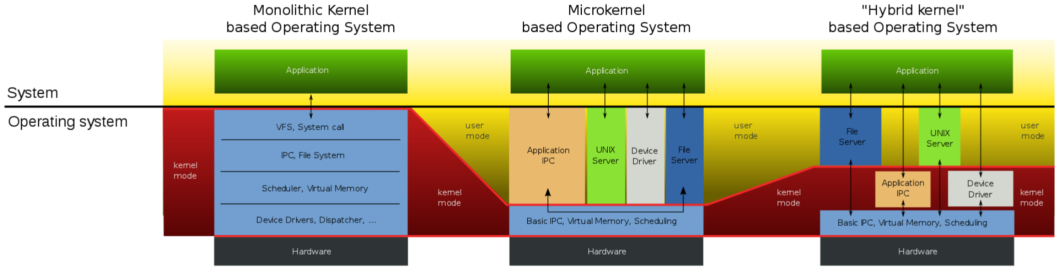
Monolithic Kernels

- All kernel services live in kernel space
- Systems like virtual file systems, schedulers, device drivers, etc share memory/communicate directly
- Device drivers can be added as modules
- The kernel provides the Application Binary Interface (ABI) to all running applications directly

Micro Kernels

- The kernel provides a limited and basic set of functionality
  - Inter Process Communication (IPC)
  - CPU Scheduling
  - Virtual Memory
- All other services (complex IPC, file system support, drivers) are called “servers” and run in user space
- Micro kernel servers communicate with each other using IPC methods such as sockets or message passing

Comparison of OS Types


Monolithic Kernel based Operating System
Microkernel based Operating System
"Hybrid kernel" based Operating System
System
Operating system
Application
VFS, System call
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
kernel mode
user mode
Hardware
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling

By Golftheman – http://en.wikipedia.org/wiki/Image:OS-structure.svg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4397379

Pros and Cons of Monolithic Kernels

Pros

- Direct communication between OS tasks is efficient, tend to be faster
- Kernel is a single process running in a single address space
- “Easier” to maintain and develop (IPC is HARD)

Cons

- Larger than a microkernel
- If one part of the kernel crashes the entire kernel (and system) crashes
- More difficult to extend (though can be done through dynamic module loading); usually requires a recompile of the kernel

Pros and Cons of Micro Kernels

Pros

- More robust: if a server crashes the system can still run (desirable for important real time operating systems)
- Smaller kernel
- Easy to extend (just write a new server!)

Cons

- Tends to be slower than a monolithic kernel (YMMV)
- More complicated code base

Kernel Types in the Wild

- **Windows NT**: Hybrid; emulation subsystems (Win32/WOW32, POSIX, OS/2) run in user space, but most major components run in kernel space
- **Linux**: Monolithic
- **BSD**: Monolithic
- **Mach**: Microkernel
- **MacOS/XNU**: Hybrid... it gets weird (it's a combination of Mach and BSD, with some other UNIX/POSIX sprinked in for flavor)
- **DOS/Windows 9x**: Monolithic