CS415 Module 1 Part A - Introduction

Athens State University

Contents

1 A few preliminaries

What I expect you to be able to do

- You will write code...
  - Many questions on homework assignments will ask you to write small programs in C or C++
  - There will be four to six programming projects requiring you to write medium to large programs in C or C++
  - You will be writing system-level code on a production OS
    - So be ready for a certain level of pain

Data Structures

- This class requires CS372 as a pre-requisite
  - You need to be comfortable working with linked lists: Singly linked, doubly linked, and circularly linked lists
  - You need to understand how trees work
    - What is a binary search tree?
    - What is the worst-case performance of searching such trees?
    - How does one insert or delete an item from a tree?
    - What is meant by an in-order, pre-order, and post-order traversal of a tree
  - Hashes and bitmaps

2 A diversion: some history

Major Achievements

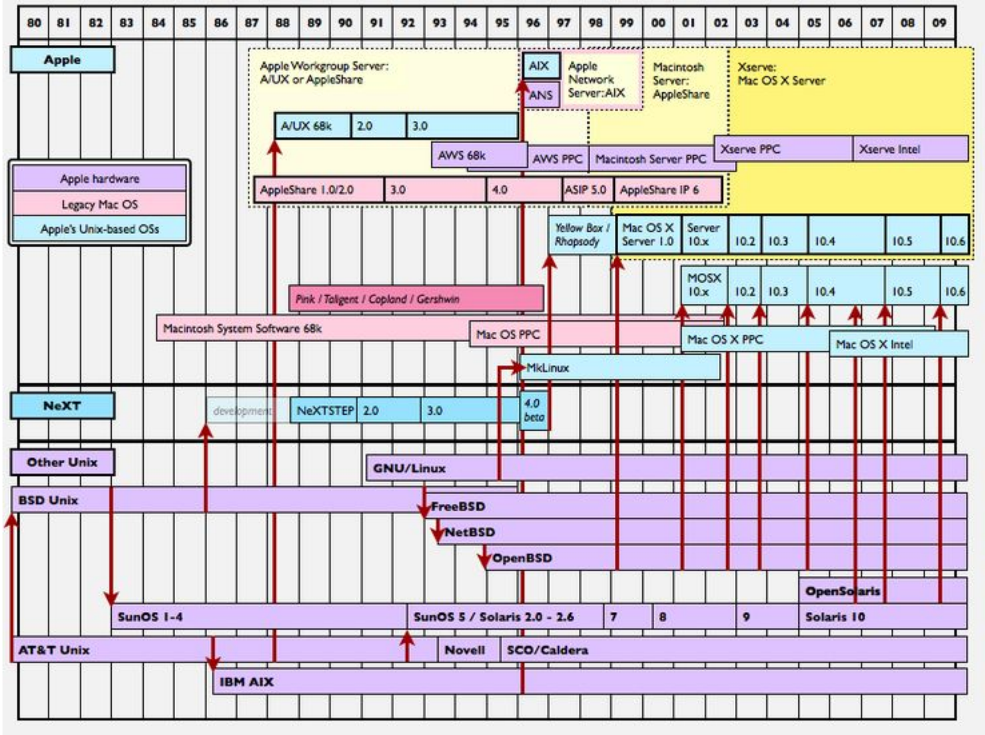
- Operating systems are among the most complex pieces of software ever developed
  - Major advances include

- ∗ Processes
- ∗ Memory management
- ∗ Information protection and security
- ∗ Scheduling and resource management
- ∗ System structure

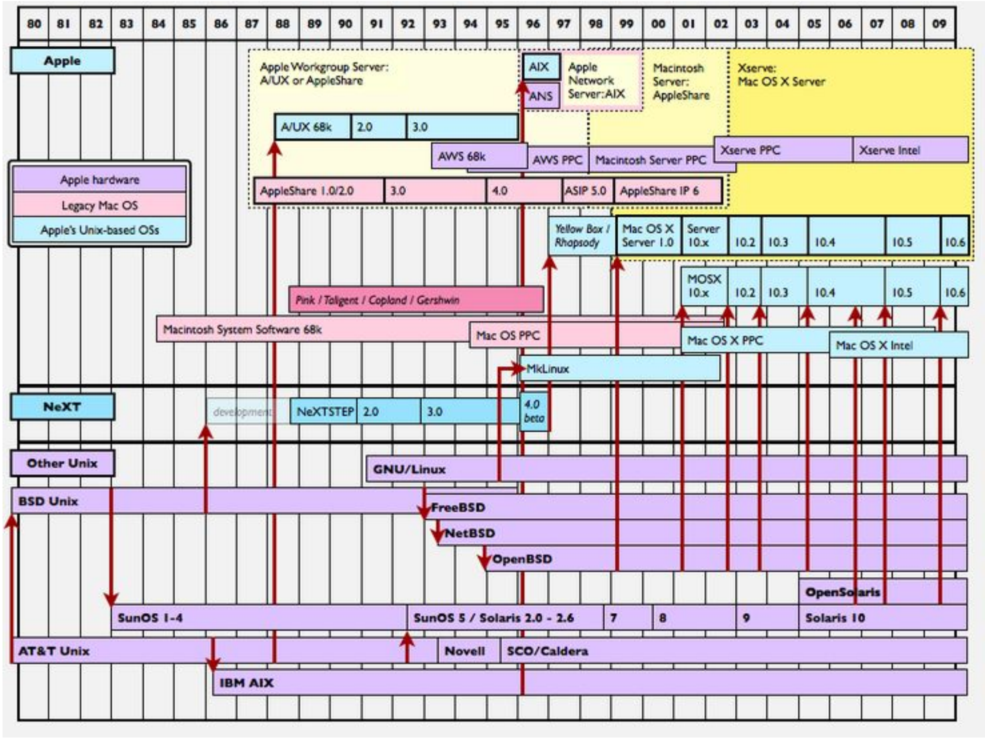
Operating systems are among the most complex pieces of software ever developed. This reflects the challenge of trying to meet the difficult and in some cases competing objectives of convenience, efficiency, and ability to evolve.

Each advance is characterized by principles, or abstractions, developed to meet difficult practical problems. Taken together, these five areas span many of the key design and implementation issues of modern operating systems.

A Timeline of Operating Systems History



A diversion: history of Windows

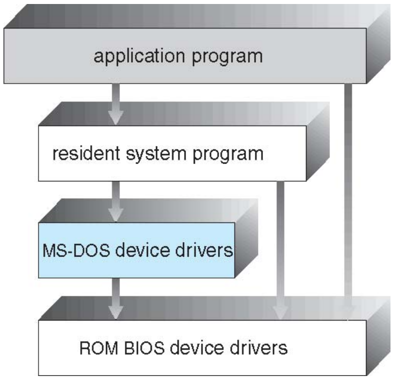


3 Operating System Structure

Simple Structure

Early mainframe systems and early PC operating system such as CP/M, MS-DOS, and classic MacOS operated under the *monitor* principle

- Written to provide the most functionality in the least space and with minimal resource use
  - Not divided into modules
  - The early PC operating system did not have well separated interfaces and levels of functionality



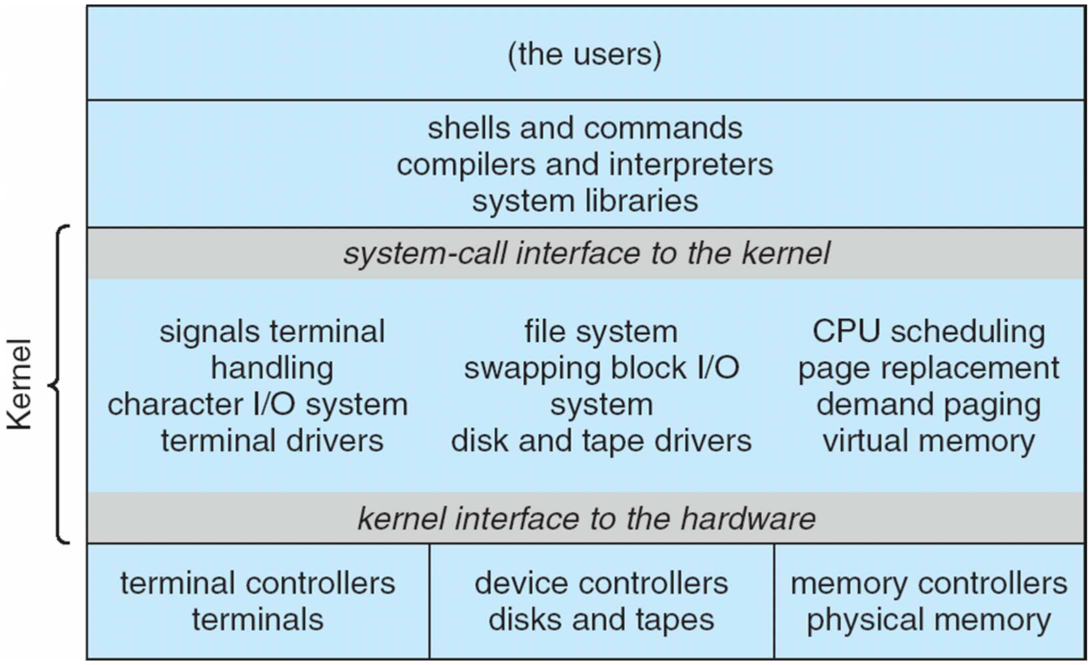
MULTICS and Unix

- The MULTICS mainframe operating system (mid-1960s) was one of first mainframe OSes to apply the *multi-layer bean dip pattern*

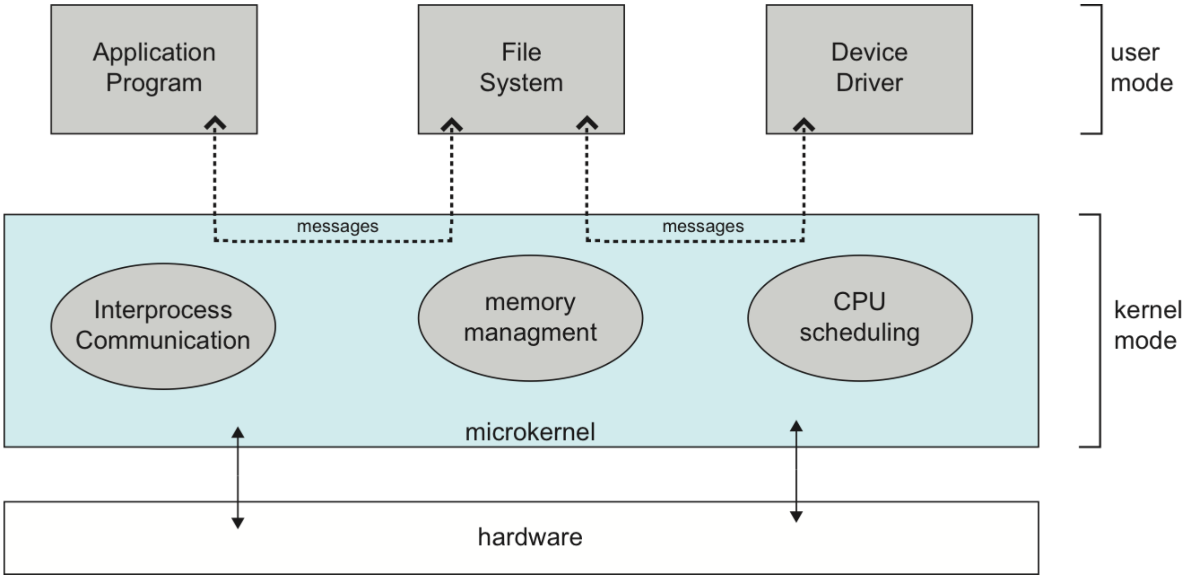
- Two separate parts: system programs and an OS kernel
- **Kernel**: Everything below the system call interface and above the the physical hardware
- The kernel provides services such as file system control, CPU scheduling, memory management, and other OS functions

Classic Unix System Structure

Beyond simple but not fully layered



Microkernel System Structure



Microkernel System Structure

- Move as much functionality as possible from the kernel to user space
- Communication takes place between user modules using *message passing*

Benefits:

- Easier to extend a microkernel
- Easier to port the OS to new machines
- More reliable and secure

Detriments:

- Performance overhead of message passing

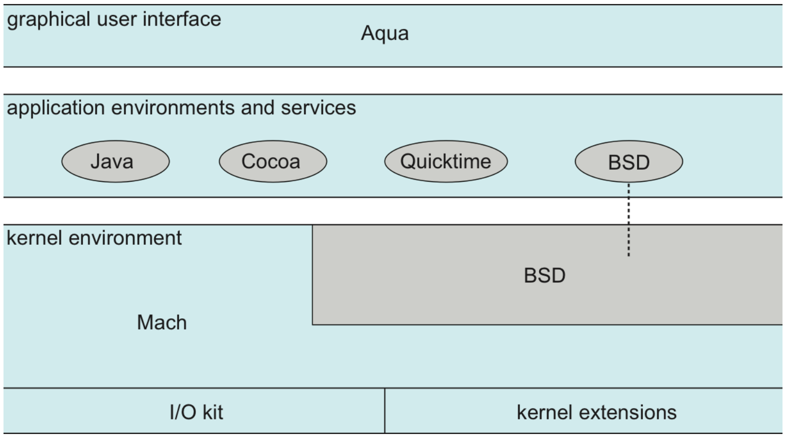
Modules and Device Drivers

- Most OSes implement *loadable kernel modules*
  - Uses an object-oriented approach
  - Each core component is separate
  - Each talks to others over known interface
  - Each is loadable as needed within the kernel
- Similar to layers but more flexible, used in Linux, FreeBSD, MacOS, Solaris, etc.

Hybrid Systems

- Modern operating systems actually not one pure model
  - Hybrid approach for performance, security, and usability needs
  - Linux, FreeBSD, Solaris kernels live in kernel address space with modules for dynamic loading of functionality
  - Windows is mostly monolithic, with microkernel model for different subsystem *personalities*
- Apple macOS is hybrid, layered environment with the Aqua UI and Cocoa programming environment
  - With a kernel composed of the Mach micorkernel with FreeBSD system calls, an OO-based model for devices and kernel modules

MacOS Hybrid Structure



Policy vs. Mechanism

- **Important principle**: Separation of policy and mechanism
  - **Policy**: *What* will be done?
  - **Mechanism**: *How* to do it?
- Policies specify what will be done, mechanisms determine how to do what needs to be done