# Computer Systems II: Kernels and Memory Management

Noah Singer, George Klees

Montgomery Blair High School Computer Team

September 29, 2017

# Overview

# Section 1: Kernels

# Stored-program computers

- Programs compiled into a sequence of instructions called **machine code**
- Stored as simple data in RAM
  - Very long sequence of **bytes** (8 **bits**)
  - Each byte has a numerical **memory address**
  - CPU can **load and store** data from RAM
    - CPU also has much faster, smaller memory called **registers**
- Data executed as code by CPU

# Operating systems



- Intermediary between hardware and applications
- Creates a high-level interface for application developers
- Controls access to hardware and enforces security procedures
- Often separated into core **kernel** and external **drivers**
  - Drivers are often used to interface with specific hardware or accomplish a specific task, and vary from computer to computer depending on hardware and setup
  - Kernel accomplishes core tasks regardless of specific hardware

# Operating modes



- When the CPU first loads, any executed code can access all memory, CPU functions, and hardware devices
  - On a secure and reasonable system, applications must be **sandboxed**
- Modern CPUs have two modes that code can execute in
  - **Supervisor**: unrestricted access to resources, only granted to kernel which accomplishes core OS functions
  - **User**: access restricted to only certain memory, certain devices, etc. (for applications)
- Special instructions called **syscalls** allow a user mode application to “jump” into the kernel in supervisor mode to accomplish an OS task
  - Read a file from the hard drive
  - Create (fork) a new process

# Types of kernels



- **Microkernel**: Only the bare minimum that requires supervisor mode is in the kernel
    - Many OS functions are user mode drivers
    - More secure and elegant
    - Slower, since many switches between user and supervisor modes may be required
- **Monolithic**: Most of the OS is in the kernel
    - Less secure since any vulnerability in the much larger kernel leads to supervisor control over the system
    - Faster and less complicated
- **Hybrid kernel**: Middle ground
    - Some user mode drivers, some supervisor mode drivers

# Core kernel/OS functions



- **Memory management**: allocating and controlling memory for applications
- **Task management**: allowing multiple applications to run simultaneously
- Virtual filesystem (**VFS**): access to stored data
  - Many layers of drivers involved: storage drivers, bus drivers, filesystem drivers, etc.
  - Must present a uniform interface for applications
- Device and power management

# Sample (simplified) x86 boot process

1. BIOS boot code is loaded from read-only memory (**ROM**), triggering code in hard drive Master Boot Record (**MBR**)
2. MBR triggers code in hard drive partition Volume Boot Record (**VBR**), loading the kernel and boot drivers from the filesystem with a filesystem driver
3. Kernel activates memory manager and enables paging
4. Kernel sets up syscalls and fault handlers
5. Kernel loads VFS using boot drivers and uses VFS to load other necessary drivers
6. Kernel initializes timers, various buses, and other hardware devices
7. Kernel begins multitasking (multicore?), switches into user mode, and loads user login application

# Section 2: Memory Management

# Memory management



- Every application's memory must be sandboxed from the others
- Each application has a unique **address space** of **virtual memory** mapped by the memory manager to **physical memory**
- When an application executes, it "sees" the address space of virtual memory
- In modern CPUs, memory is organized into medium-scale (often 4096 B) **pages**
- Task of kernel memory manager: for each application, create a correspondence between virtual pages and physical pages

# Kernel memory manager



- **Physical memory allocator** allots pages of physical RAM to applications and the OS
- **Virtual memory manager** organizes the address space of applications
- Kernel communicates with paging hardware (memory management unit, or **MMU**) inside of CPU
  - **Page tables** stored in memory; MMU contains a pointer in a special register
  - MMU translates memory accesses from virtual to physical addresses
  - Lookups cached in the translation lookaside buffer (**TLB**)

# Page tables



- Stored as an array of **page table entries (PTEs)**, each representing a page
- PTE stores detailed information, allowing for fine control over memory access:
  - Address of page in physical memory
  - Permissions
  - Caching information

# Application address space



- Four main regions:
  1. Program code and data
  2. **Heap**: dynamically allocated data growing upwards
  3. Unallocated memory
  4. **Stack**: return addresses/stack frames, local variables, parameters, execution context
- When invalid memory is accessed, MMU throws a **page fault**, halting execution; can be used creatively to great advantage
  - Memory-mapped files
  - Swapping
- OS memory manager must: allocate and free memory, allow shared memory, map and unmap files, etc.

# Conclusion



- OS interfaces between hardware and applications
- Code executes in user mode or supervisor mode
- Virtual memory allows kernel to control application address spaces