Kernel basics

CS390R - UMass Amherst

Course Information

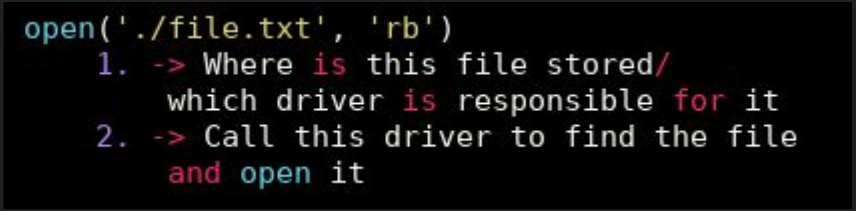
- Project 5 ongoing
- Homework assigned

Today’s Content

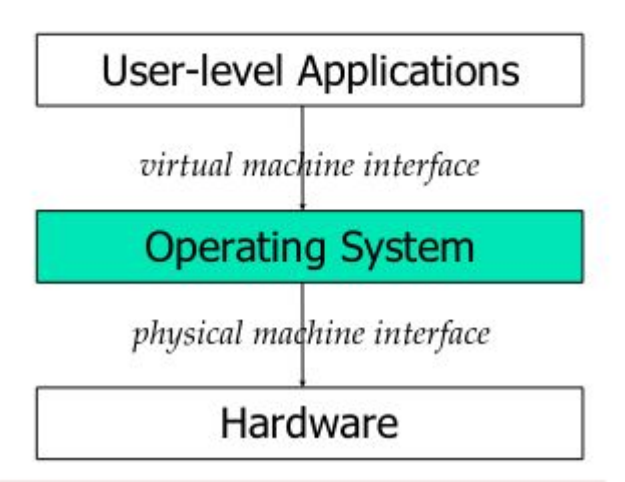
- Introduction
- Microkernel vs Monolithic Kernel
- Protection
- Syscalls
- Boot process
- Device drivers
- Extra resources

What is a Kernel?

- Most “central” part of the operating system
- Hardware -> Software interface
- Manages memory, networking, filesystem, processes, and device drivers
- Acts as an intermediary between programs and hardware
- Application makes a syscall, which the kernel then handles
- Implements a “virtual machine” that is easier to program than raw hardware



```
open('./file.txt', 'rb')
    1. -> Where is this file stored/
        which driver is responsible for it
    2. -> Call this driver to find the file
        and open it
```



User-level Applications

virtual machine interface

Operating System

physical machine interface

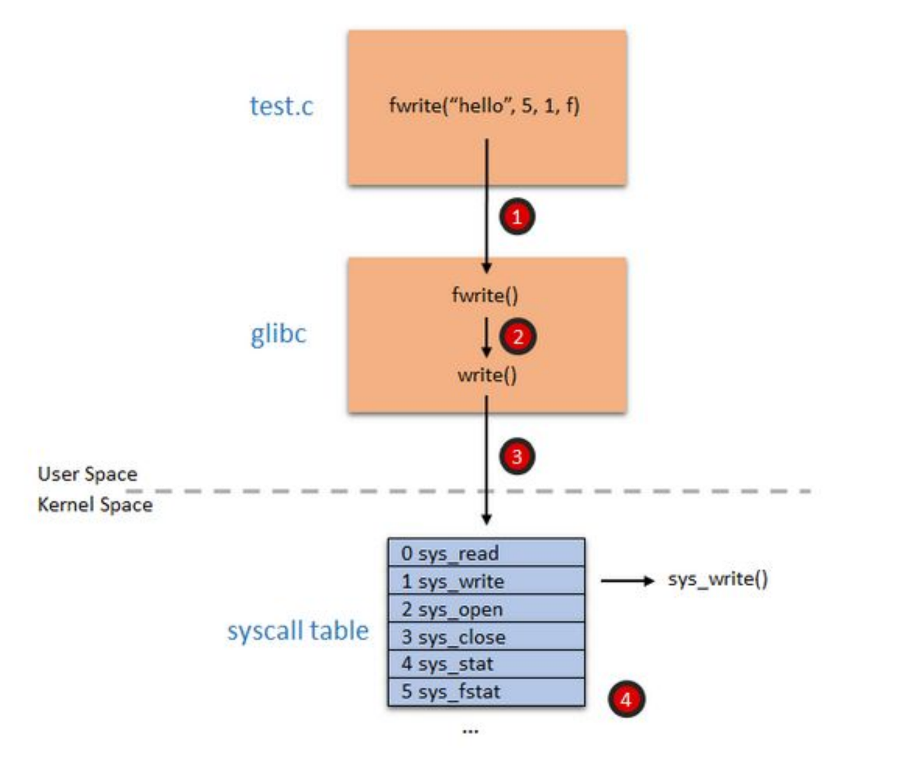
Hardware

Microkernel vs Monolithic kernel

- Monolithic Kernel
  - Entire OS runs in a single address space
  - Any module can corrupt any other part of the OS
  - Device drivers contain about 70% of OS-code, and contain ~3-7 times more bugs
  - Linux, FreeBSD
- Micro Kernel
  - Small kernel with most OS functionality implemented in user-space processes
  - More reliable, easier to extend, faults are isolated to faulting driver
  - Since everything is separated, the communication between different OS-interfaces comes at an added cost, resulting in reduced performance
  - Nowadays, mostly embedded systems

Switching between rings - Syscalls

- Usermode applications sometimes need to perform privileged operations (reading a file)
- Syscalls act as a bridge between user and kernel mode
- x86_64 -> Set rax to syscall number and use `syscall` instruction
- What happens when a syscall is used?
  1. Registers and stack-state are saved
  2. Privilege level switches to Ring 0
  3. Engage syscall handler routine
  4. Restore registers/stack
  5. Privilege level switches to Ring 3



Boot-process

1. BIOS - Basic input output system
   - Stored on EEPROM/ROM, loads a small program from device/disk and hands off control to it
2. MBR - Master Boot Record
   - Stage 1 boot-loader, 440 bytes, look for partition marked active and load code from it
3. Bootloader
   - Executable, often supports multi-boot, loads fs drivers, config files, drivers & launches os
4. Kernel
   - Mount actual fs, detect hardware, start INIT
5. Init
   - (systemd) ‘/sbin/init’, first process in the system, all other processes are its descendants
6. Runlevel
   - Run scripts from correct dir to start services, ‘/etc/rc.d/rc*.d/’, eg. networking, sshd, etc

Device Drivers /1

- Enables hardware devices to communicate with the kernel
- Without them the computer would not be able to interact with hardware
  - Keyboard, usb, mouse, hdmi, printers, filesystems, etc each have their own drivers
- Mature OS's generally provide support for most common hardware
- Drivers operate at ring 0 within the kernel (in monolithic kernels at least)
- In linux they can be loaded/unloaded without requiring a system reboot
- Device drivers setup custom system call handlers tailored specifically to their device
- These system call functions are invoked whenever the user attempts to interact with that device
- Different types of devices:
  - Character: Transfer data 1 byte at a time
  - Block: Communicate with drivers through file management subsystems, increases perf when dealing with large amounts of data

Device Drivers /2

- module_init(init_func);   - Setup data structures and register driver into kernel
- module_exit(exit_func); - Clean up remnants and unload driver
- Registering drivers into the kernel:
    - Each device has a unique major + minor number combination
    - Major number identifies device type, Minor number identifies specific device
    - Using these numbers, a driver can be assigned to specific device types
- We can create fake devices: ‘mknod /dev/a_device c 55 1’
    - Create character device ‘c’ ; Major number 55 ; Minor number 1
    - We can now register a driver using these numbers that is invoked whenever a process attempts to access this device

Device Drivers /3

- Required structures:
    - cdev - used to register devices in the system (holds major/minor and other important data)
    - file_operations - describes syscalls supported by this driver
    - Inode & file structures - inode represents files as they are stored in fs, file structure holds information about opened files (cursor position, flags, etc)
- Shell commands:
    - lsmod - list loaded kernel modules
    - insmod - load a new kernel module
    - rmmod - remove a loaded kernel module
    - dmesg - print the kernel ring buffer

Extra Resources

- UMass CS377
- UMass CS577
- https://seal9055.com/blog/kernel/char_driver_part_1
- https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html
- https://lwn.net/Kernel/LDD3/