OPERATING SYSTEMS-1
CS 241, SPRING 2020
The kernel Abstraction

Ass. Prof. Ghada Ahmed
ghada_khoriba@fci.helwan.edu.eg

Edited slides,
main reference

Operating systems:
principles and practice, Tom Anderson, 2nd ed

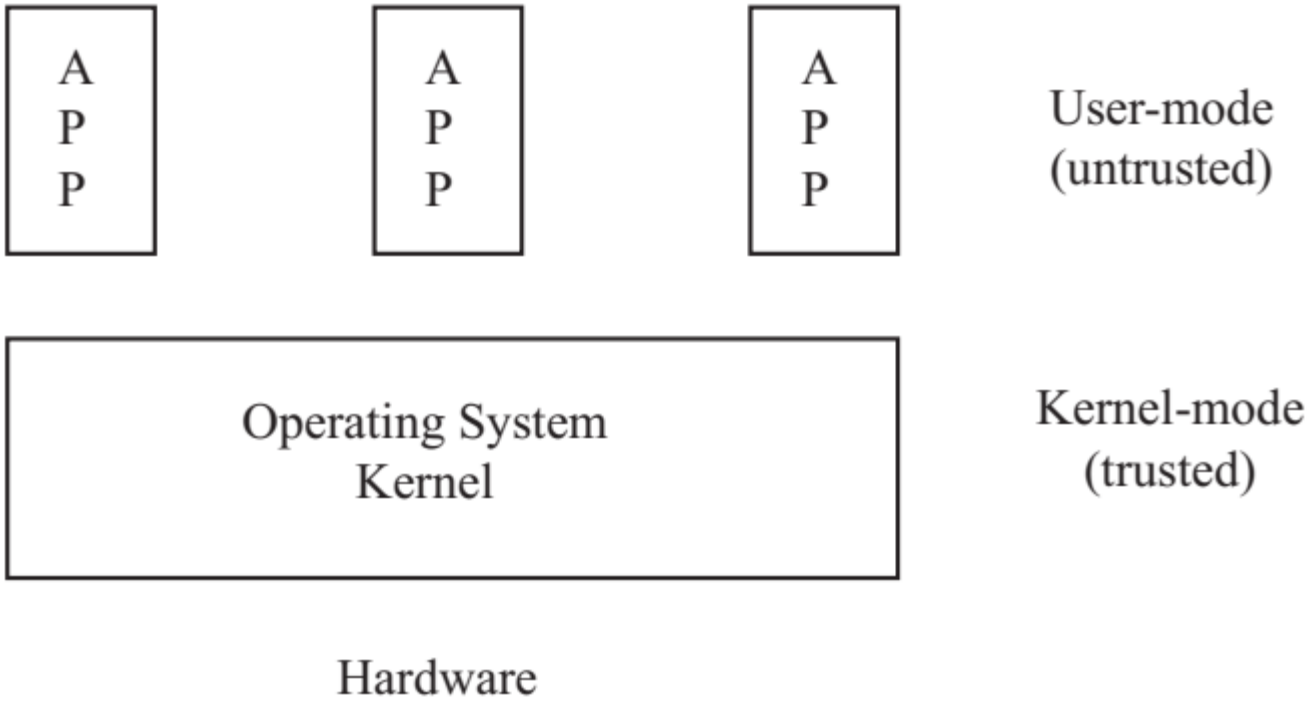
DEBUGGING AS ENGINEERING

- Much of your time will be spent debugging
  - In industry, 50% of software development is debugging
  - Even more for kernel development
- How do you reduce time spent debugging?
  - Produce working code with smallest effort
- Optimize a process involving you, code, computer

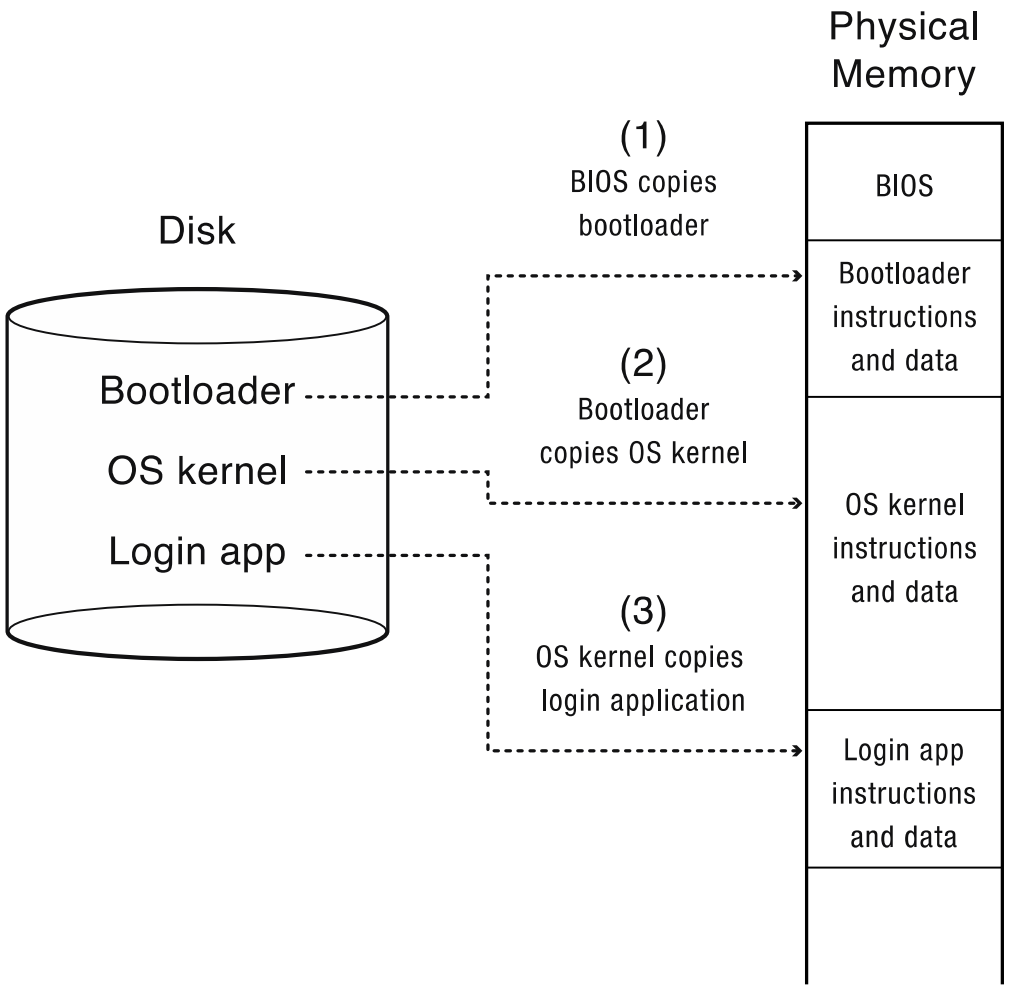
DEBUGGING AS SCIENCE

- Understanding -> design -> code
  - not the opposite
- Form a hypothesis that explains the bug
  - Which tests work, which don't. Why?
  - Add tests to narrow possible outcomes
- Use best practices
  - Always walk through your code line by line
  - Module tests – narrow scope of where problem is
  - Develop code in stages, with dummy replacements for later functionality

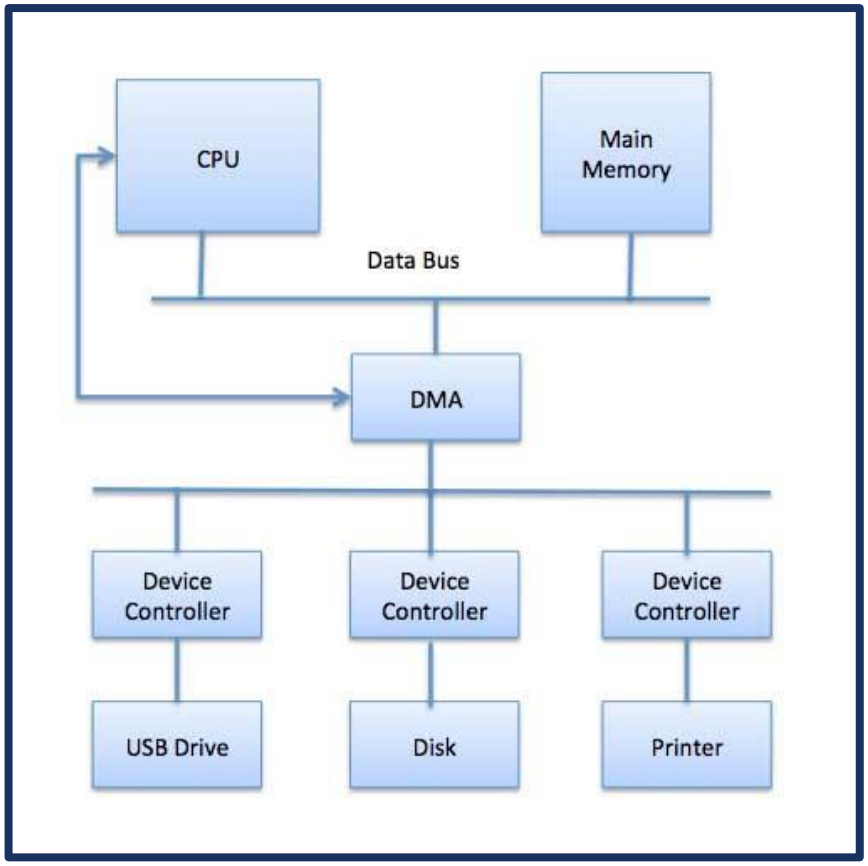
KERNEL MODE


A
P
P
A
P
P
A
P
P
User-mode
(untrusted)
Operating System
Kernel
Kernel-mode
(trusted)
Hardware

BOOTING


Physical Memory
(1)
BIOS copies bootloader
BIOS
Disk
Bootloader instructions and data
Bootloader
(2)
Bootloader copies OS kernel
OS kernel
OS kernel instructions and data
Login app
(3)
OS kernel copies login application
Login app instructions and data

DEVICE INTERRUPTS



- OS kernel needs to communicate with physical devices
- Devices operate asynchronously from the CPU
  - Polling: Kernel waits until I/O is done
  - Interrupts: Kernel can do other work in the meantime
- Device access to memory
  - Programmed I/O: CPU reads and writes to device
  - Direct memory access (DMA) by device
  - Buffer descriptor: sequence of DMA's
    - E.g., packet header and packet body
  - Queue of buffer descriptors
    - Buffer descriptor itself is DMA'ed

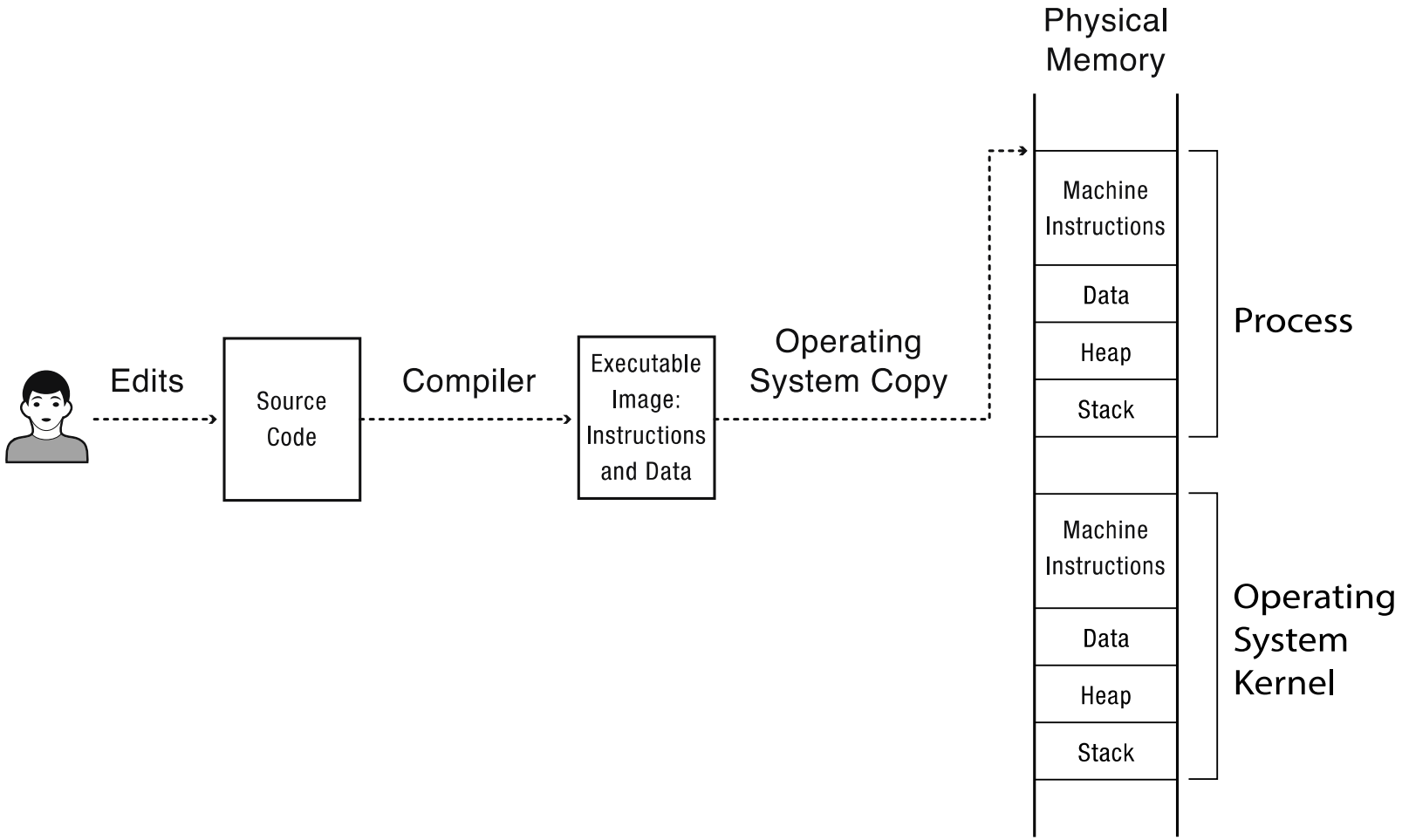
DEVICE INTERRUPTS

- How do device interrupts work?
  - Where does the CPU run after an interrupt?
  - What is the interrupt handler written in? C? Java?
  - What stack does it use?
  - Is the work the CPU had been doing before the interrupt lost forever?
  - If not, how does the CPU know how to resume that work?

CHALLENGE: PROTECTION

- How do we execute code with restricted privileges?
  - Either because the code is buggy or if it might be malicious
- Some examples:
  - A script running in a web browser
  - A program you just downloaded off the Internet
  - A program you just wrote that you haven't tested yet

A PROBLEM


Physical Memory
Edits
Source Code
Compiler
Executable Image: Instructions and Data
Operating System Copy
Machine Instructions
Data
Heap
Stack
Process
Machine Instructions
Data
Heap
Stack
Operating System Kernel

MAIN POINTS

- Process concept
  - A process is the OS abstraction for executing a program with limited privileges
- Dual-mode operation: user vs. kernel
  - Kernel-mode: execute with complete privileges
  - User-mode: execute with fewer privileges
- Safe control transfer
  - How do we switch from one mode to the other?

PROCESS ABSTRACTION

- Process: an *instance* of a program, running with limited rights
  - Thread: a sequence of instructions within a process
    - Potentially many threads per process (for now 1:1)
  - Address space: set of rights of a process
    - Memory that the process can access
    - Other permissions the process has (e.g., which system calls it can make, what files it can access)

THOUGHT EXPERIMENT

- How can we implement execution with limited privilege?
  - Execute each program instruction in a simulator
  - If the instruction is permitted, do the instruction
  - Otherwise, stop the process
  - Basic model in Javascript and other interpreted languages
- How do we go faster?
  - Run the unprivileged code directly on the CPU!

HARDWARE SUPPORT: DUAL-MODE OPERATION

- Kernel mode
  - Execution with the full privileges of the hardware
  - Read/write to any memory, access any I/O device, read/write any disk sector, send/read any packet
- User mode
  - Limited privileges
  - Only those granted by the operating system kernel
- On the x86, mode stored in EFLAGS register
- On the MIPS, mode in the status register

A MODEL OF A CPU