Introduction to Operating Systems

www.cs.mta.ac.il/~carmi

Instructor: Carmi Merimovich
TA: Tzvi Melamed

Objectives

- Understand kernel(?!) implementation.

Prerequisites:

- Hardware issues.
- 'Usual' (i.e., user) programming issues.

Method:

- Understanding the xv6 kernel.

The xv6 kernel

- Educational kernel.
- Copyrighted by Frans Kaashoek, Robert Morris, and Russ Cox.
- Written for the 32-bit x86 machine.
- Written mostly in C.
- Very (very) small part in assembly.
- Modelled after Unix V6 (circa 1976 on PDP11).
- (The kernel was written by Ken Thompson.)
- Supports multiprocessors.

The xv6 (rev10) booklet should be with you. Always.

Operating systems

- Windows (Microsoft).
- Unix (AT&T originally, free nowadays).
- UBUNTU.
- Feodora.
- Android. (Google, Linux distro).
- iOS. (Apple, Unix based).

Linux?

- Linux is mostly kernel only.
- Ubuntu, feodora are Linux distributions, i.e., operating systems.

Kernel mission

- Manage resources:
  - Processor(s).
  - Work memory (ROM/RAM).
  - Storage memory (Disk).
  - Keyboard.
  - Mouse.
  - Screen
- Presents programming model for 'applications': Process.
- Security.

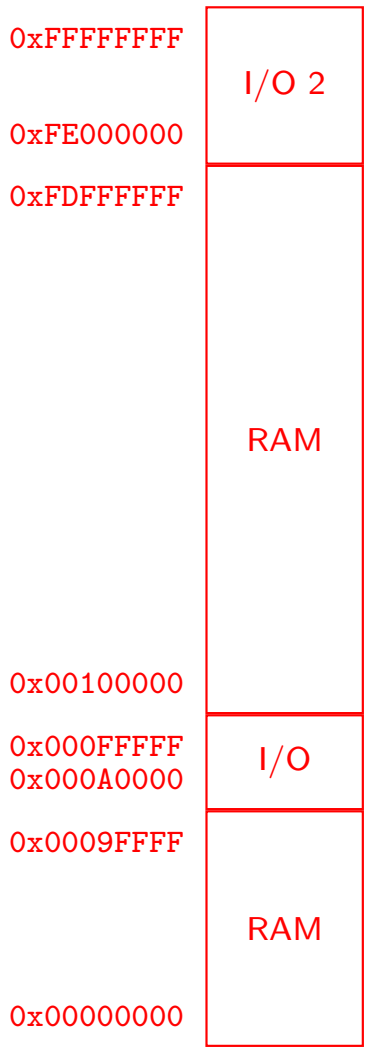
Processor(s)

- Usually there are more running applications than processors.
- Kernel code (scheduler) decides which 'application' gets the processor.
- Kernel might need to remove application from processor.
- Replacing apps in a processor is called context switch.
- Time multiplexing.

Work memory

- Part reserved for the kernel.
- Rest of memory divided between running applications.
- Space multiplexing.
- We want STRONG separation between apps memories.
- This requires hardware help.

32-bit x86 physical memory layout



Storage memory

- Partition disk between apps: Files.
- Space multiplexing.
- Disk is slow and executes ONE read/write at a time.
- Processor(s) is fast (and maybe many).
- Time miltiplexing.

screen

- ?
- Windowing system: space multiplexing.
- Without windowing system: Quite a mess on screen.

Keyboard/Mouse

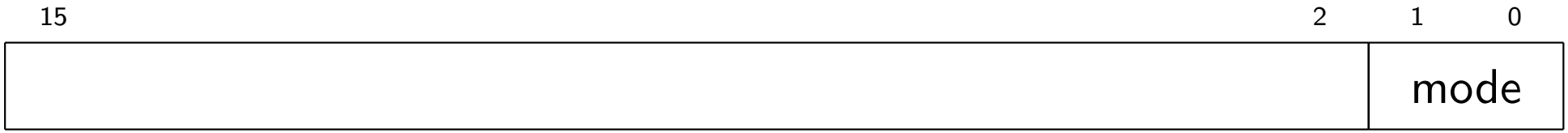
- ?
- Windowing system: Time multiplexing.
- Without windowing system: Quite a mess

CPU modes

- In order to allow something to control the computer the processor has two states:
  - Privileged: Kernel mode.
  - Not privileged: User mode.
- In kernel mode the processor is not restricted.
- In user mode the process aborts when instructed to do 'privileged' stuff.
  - Privileged instructions, e.g., `hlt`.
  - Accessing some (privileged) registers, e.g., `cr3`, `idtr`, etc...

Pragmatically, kernel code is the code running when the processor is in kernel mode.

x86 privilege state



register cs

- 00: Kernel mode.
- 11: User mode.

Mode switch:

- Switching between modes programmatically is not easy.
- Interrupt servicing switches the processor to kernel mode.