Operating Systems
2-7029110

Lecture 1 - Intro

Ariel University
Computer Science Department

Welcome to Operating Systems course!

- Who Am I ? (Where is Dr. Kogan ?)
- Contacts and reception hours
- Why this course?
- Duties

Welcome to Operating Systems course!

- Who am I ?
  Arkady (אהרן) Gorodischer.
  About 10 years of developing experience
- Contacts
  Phone: 0546407773 (limited availability)
  Email: arkady82@gmail.com
  Reception hours: Sunday, 17:30 – 19:00

Welcome to Operating Systems course!

Why this course?

OS is a running environment and should be effectively used. It may help with a lot of already made tools but may become a nightmare if one is not aware of how should it be handled.

Welcome to Operating Systems course!

Duties:

- No attendance is required but what is told during the lectures is obligatory.

- 70% - Exam (> 56)
  30% - Homework (avg > 56)

A little bit of History

- 1960’s – IBM OS/360 – OS for a product line
- 1970’s – PLATO – Chat, multi-player games
- 1980’s – MS DOS – for a PC. CMD , single user
- 1980’s – Apple (Xerox) Windows interface
- 1990’s – Win95,NT, GNU/Linux, Symbian
- 2000’s - WinXP, iOS, Android
- 2010’s – Raspbian, ROS, SailFish, Fuchisa
- 2020’s – You may change the world :-)

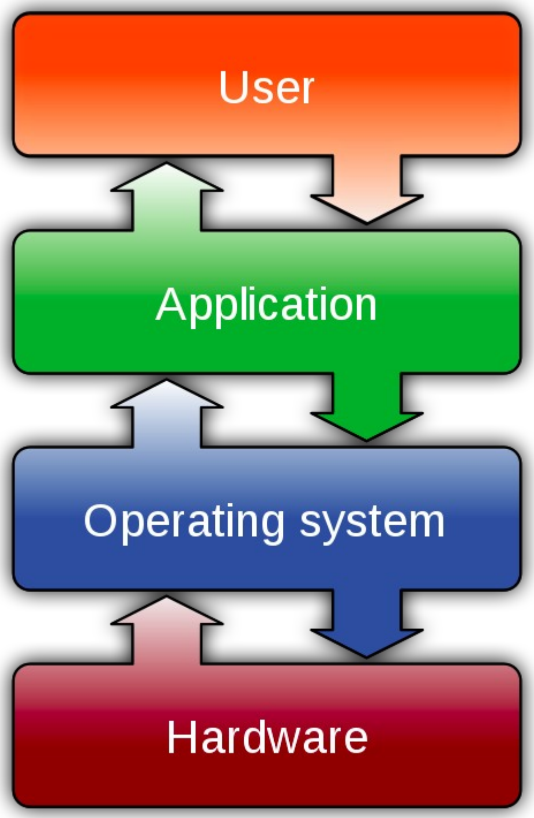
What is OS - Operation System

<u>The world before and after</u>

- Bootloader /BIOS – minimalistic OS

- Bare Metal / Machine – how was it done
  Extended Machine – the better way
- OS:
  An Extended machine
  A Resource Manager

Operation System Place



CMD and GUI interfaces

Editors, File managers,communication

Multitask, singe/multi user,security

x86/arm, tape/disk/flash, networking

Extended Machine

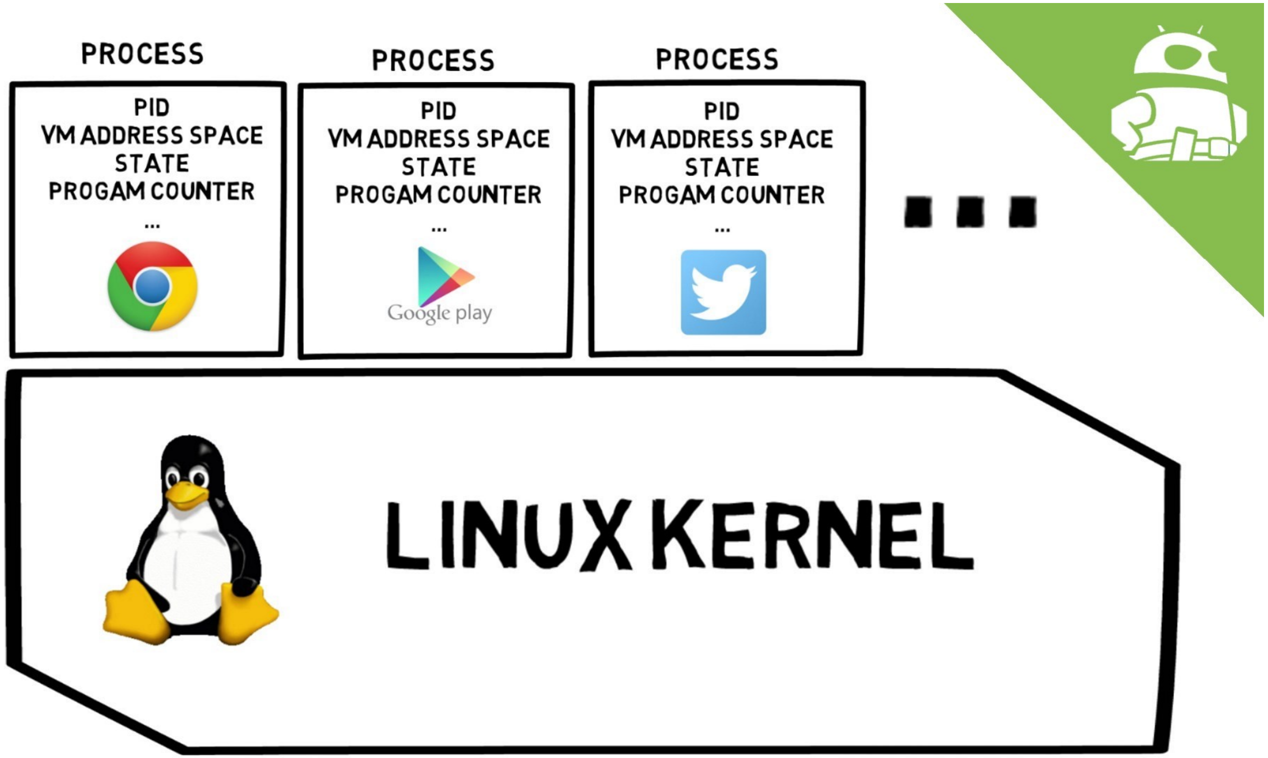
- Provides
  stable: doesn't crash
  portable: can run code on more than
  one type of machine
  reliable: always reacts in the same way
  safe: doesn't do something dangerous
  well-behaved: acts in a proper manner environment
- Computer "appears" to more than it is
  "appears" to be many processors
  "appears" to be many, large memories
- Features:
  threads, processes, files, communication channels

Resource Manager

- Support many devices simultaneously
  e.g. keyboard, mouse, printer, speakers, microphone
- Share resources among users and programmes
  fairly: each programme gets a change to run
  safely: protects against corruption
  efficiently: using the available resources to provide
  the best service possible
- Allocates resources to users
  Disks, memory, network interfaces, timers,
  terminals/displays, laser printers, etc
  Who’s using what?
  How is it shared?

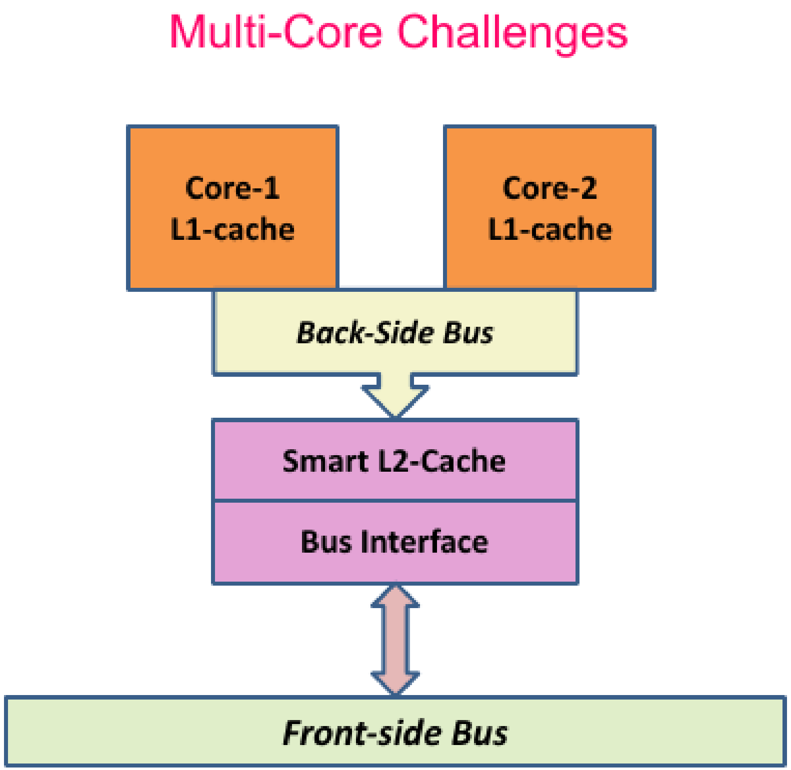
Process management

- Process and Thread Management, Synchronization and Scheduling



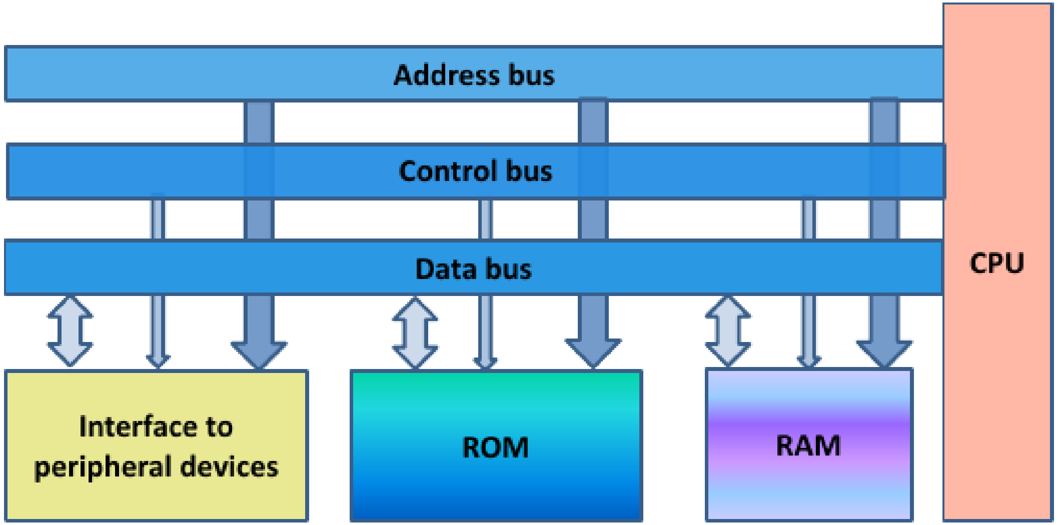
CPU management

- Processing and I/O
- Computation communication overlaping
- Scheduler

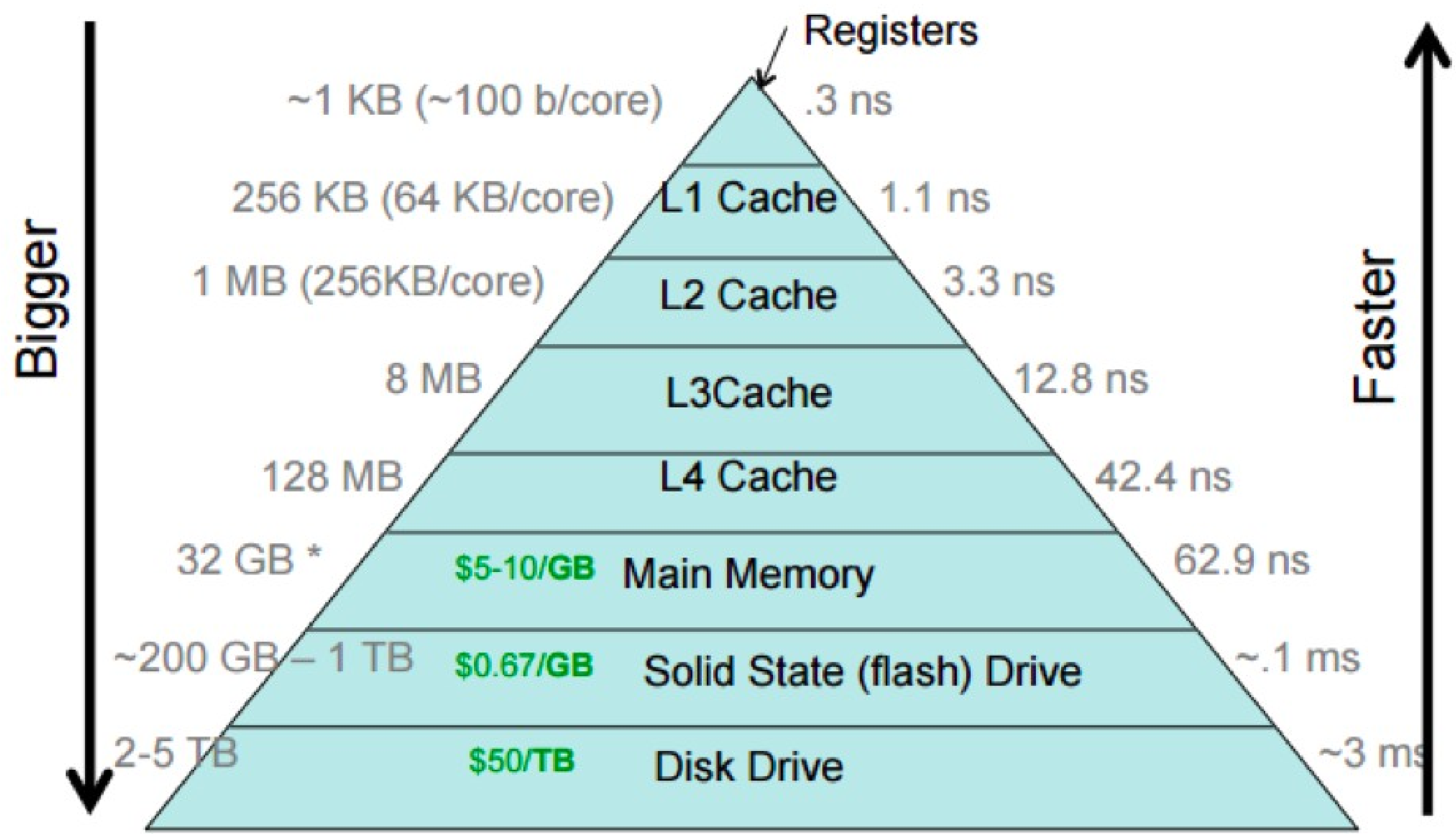


Memory management

- Main Memory management
  זכרון ראשי
- Secondary storage management
  זכרון משני



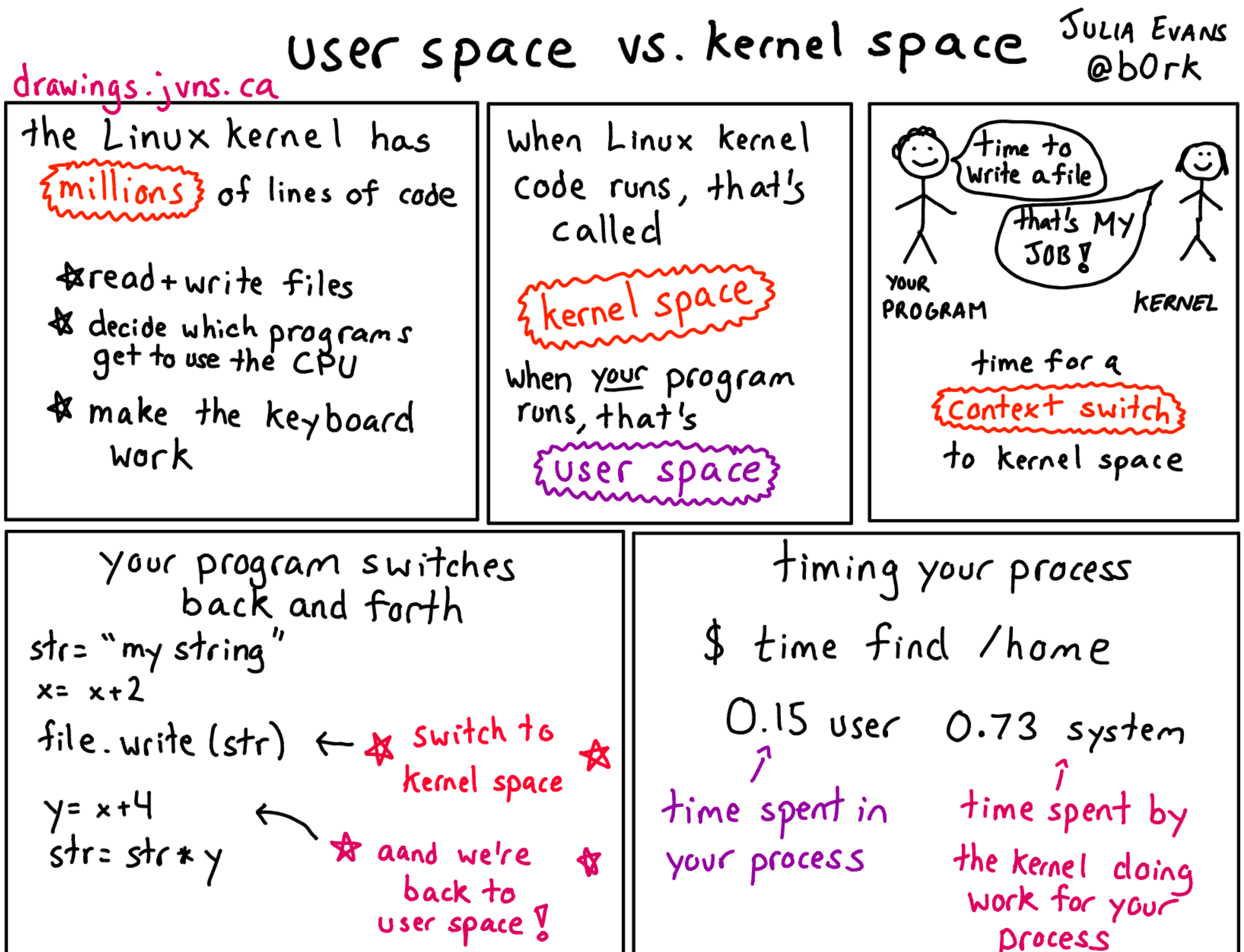
Memory management



Interupts and Interupts handlers

```
Do forever{
    IR = memory[PC];
    execute(IR);
    PC++;
    If(Interrupt_Request) {
        memory[0] = PC;
        PC = memory[1]
    }
}
```

“User space” vs “Kernel space”



User “space” vs “Kernel space”

The division is made to protect the system, and separate sensitive system parts from user code

Protecting CPU Privileged instructions
Protecting OS Scheduler, Interrupts, etc

So, how can we use the kernel ??

System Calls

An application can interact with the kernel via system call (syscall). A Dedicated API, that cares more about security, than being a user-friendly.

F.ex c++ fread() -> .... -> kernel read()

https://linuxhint.com/what-is-a-linux-system-call/
https://linuxhint.com/linux_system_call_tutorial_c/

System Calls 32bit example

```
# ------------------------------------------------------------------------------------------
# Writes "Hello, World" to the console using only system calls. Runs on 32-bit Linux only.
# To assemble and run:
#
#     gcc -m32 -c hello-32.s && ld hello-32.o && ./a.out
#
# or
#
#     gcc -m32 -nostdlib hello-32.s && ./a.out
# ------------------------------------------------------------------------------------------

        .global _start

        .text
_start:
        # write(1, message, 13)
        mov     $4, %eax                # system call 4 is write
        mov     $1, %ebx                # file handle 1 is stdout
        mov     $message, %ecx          # address of string to output
        mov     $13, %edx               # number of bytes
        int     $0x80                   # invoke operating system to do the write

        # exit(0)
        mov     $1, %eax                # system call 1 is exit
        xor     %ebx, %ebx              # we want return code 0
        int     $0x80                   # invoke operating system to exit
message:
        .ascii  "Hello, world\n"
```

System Calls 64bit example

```
# ----------------------------------------------------------------------------------------
# Writes "Hello, World" to the console using only system calls. Runs on 64-bit Linux only.
# To assemble and run:
#
#     gcc -c hello.s && ld hello.o && ./a.out
#
# or
#
#     gcc -nostdlib hello.s && ./a.out
# ----------------------------------------------------------------------------------------

        .global _start

        .text
_start:
        # write(1, message, 13)
        mov     $1, %rax                # system call 1 is write
        mov     $1, %rdi                # file handle 1 is stdout
        mov     $message, %rsi          # address of string to output
        mov     $13, %rdx               # number of bytes
        syscall                         # invoke operating system to do the write

        # exit(0)
        mov     $60, %rax               # system call 60 is exit
        xor     %rdi, %rdi              # we want return code 0
        syscall                         # invoke operating system to exit
message:
        .ascii  "Hello, world\n"
```