Chapter 2 Introduction to Operating Systems

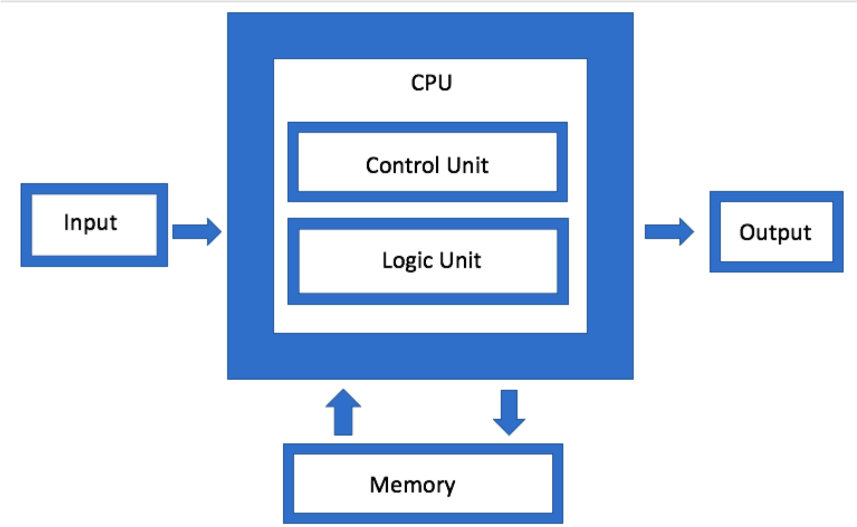
Why study OS?

- Nice to know “how stuff works”
- Some conceptually interesting and beautiful ideas in OS
- First steps towards real understanding of data centres and other systems
-

Quick reminder of how a computer works:

https://cpuvisualsimulator.github.io/

https://www.vnmsim.app/en-us



What happens when a program runs?

- Load program from disk -> memory
- Processor executes instructions (run)—That the CPU can understand “Instruction set architecture (ISA)”
  - Fetch instruction from memory
  - Decode instruction to figure out which instruction it is
  - Execute the thing the instruction is supposed to do
    - Instructions:
      - Add two numbers: ADD #5, D1
      - Access memory (LOAD / STORE)
      - Comparison: check condition: CMP #0, D0
      - Jump to function instructions
    - Move to next instruction
  - Does this process many billions of times per second
  - Until program completes

Life cycle of a program. This computing model is often called **von Nuemann’s model** of computing.

What is the purpose of the OS?

- Make it easy to run programs
- Share the processor
- Interact with devices
- Allow programs to share memory

The OS must operate **correctly** and **efficiently**.

Virtualization

OS achieve these goals with a general technique: **Virtualization**

- Take physical resource (CPU, memory, disk …)
- Transform this resource into a more general and easy-to-use virtual form.

Virtual computer (or machine)

- 1 CPU -> Many CPUs — because we can then run multiple programs at the same time (seemingly)
- 1 Memory -> Many programs appear to have large private memories

- CPU: Switch between many programs (time sharing)
- Memory: divide memory between many programs (space sharing)

Standard library

- Typically an OS exports or provides a few hundred **system calls**.
  - Also called: Application Programming Interface (API)
  - Examples: Open(), Read(), Write(), Close()
- Make system easy to use

OS role as a resource manager

- **Scheduling**: OS must manage access to the CPU
- Similar for memory (sharing memory) and disk (sharing storage space)
- Must be managed
  - Fairly - can't let one program consume whole CPU, when other programs don't get any CPU
  - Efficiently - can't cause excess overhead
  - Securely - must protect memory of other programs (isolation); avoid programs harm each other, including harming the OS.
  - Reliable - must run non-stop; if it fails, all applications fail as well!!

Design Goals

Abstractions are fundamental to everything in computer science

- Abstractions makes it possible to write a large program by dividing it into small and understandable chunks…

Examples:

- to write such a program in a high-level language like C without thinking about assembly
- to write code in assembly without thinking about logic gates,
- to build a processor out of gates without thinking too much about transistors and physics!

Questions:

- What does a compiler do?
- What does mean that a program can run on a x86 or ARM machine?
- What does it mean that a CPU is slow or quick?
- What will that program print?