University of Nottingham
UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 2: Hardware

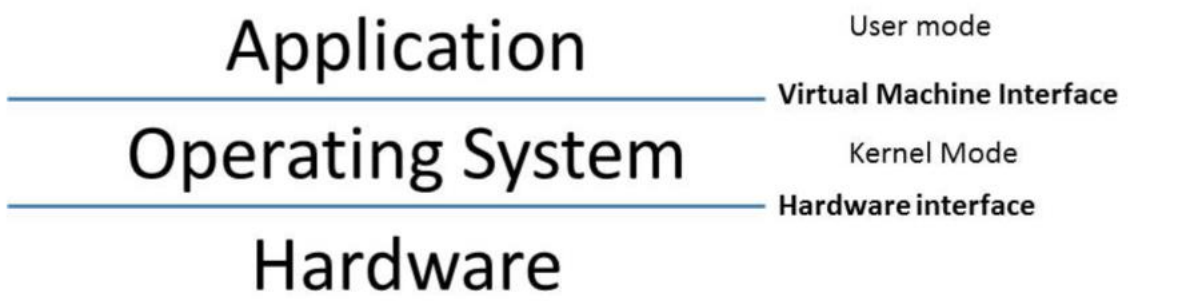
University of Nottingham, Ningbo China

2024

Recap
Last Lecture

- The **code** we write heavily **uses operating system functionality**

- The **operating systems provides abstractions** (e.g., it hides hardware details) and **manages resources**



Goals of Today

Overview

- Computer Hardware
  - Interrupts, context switching
- OS structures/implementation

Overview of Computer Hardware

- Computer hardware forms the foundation for the operation of any system.
- Component of Modern Computer
  - Memory (Registers, Cache, Main Memory, Disk)
  - Disk Storage
  - Processors (CPU)
  - Input/output Devices (I/O)
  - Buses

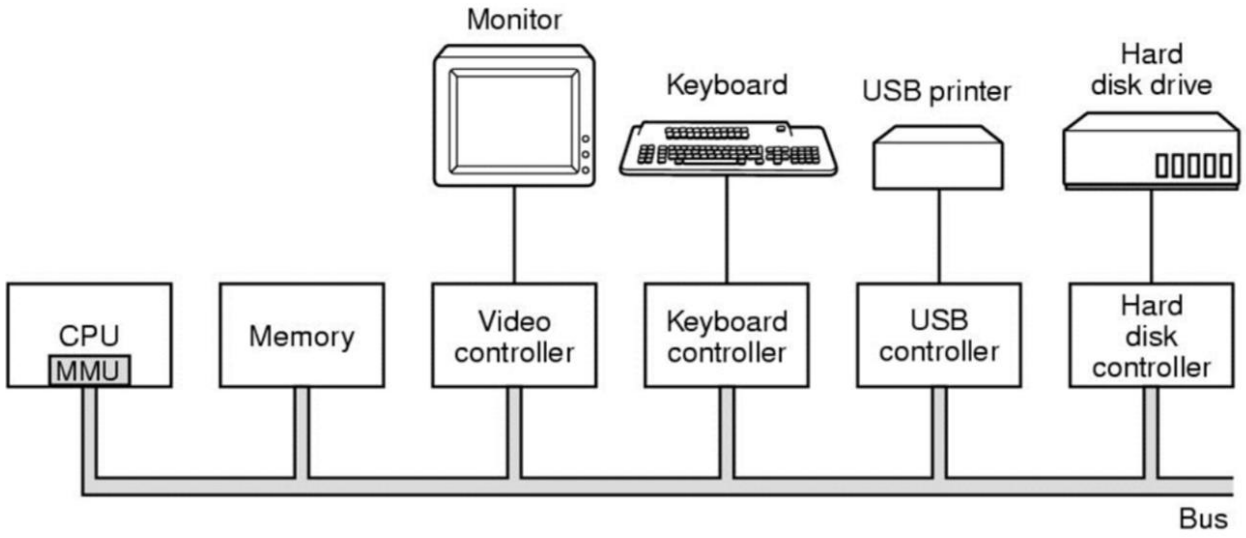
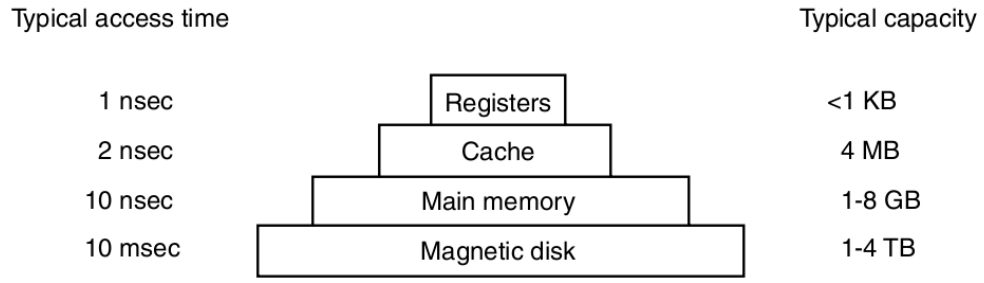


Figure: Simplified computer model (Tanenbaum, 2014)

Computer Hardware
Memory

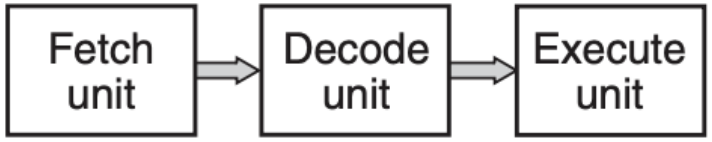
- Memory is where the computer temporarily or permanently stores data for quick access.
- Registers:
  - The smallest and fastest memory units located inside the CPU.
  - Hold data that the CPU is currently working on, such as operands for arithmetic calculations or the memory address for data retrieval.
- Cache:
  - Faster than main memory (RAM) but smaller in size.
  - Located close to the CPU and stores frequently accessed data, reducing the time to retrieve it from main memory.
  - There are usually multiple levels of cache (L1, L2, L3) with L1 being the fastest and smallest.



- Main Memory (Random Access Memory) (RAM)
  - The computer's primary working memory.
  - It temporarily stores data and programs that the CPU is currently processing.
  - RAM is faster than disk storage but slower than cache.
- Disk Storage:
  - The long-term storage of the computer, used to save data permanently, even when the power is off.
  - Disk storage includes hard drives (HDD), solid-state drives (SSD), and other forms of storage media.
  - Much slower than RAM and cache but has a significantly larger capacity.

Computer Hardware
CPU Design

- The CPU (Central Processing Unit) is the "brain" of the computer.
- A CPU's basic cycle consist of **fetch**, **decode**, and **execute**
  - The CPU fetches instructions from memory, decodes them to understand what the instruction is, and then executes them. This repeats continuously.
- Every CPU has his own instruction set that it can execute.
- A CPU has a set of registers (extremely fast memory close to the CPU "core")



Computer Hardware

CPU Components

Component	Function
Control Unit (CU)	Directs operations, fetches and decodes instructions.
Arithmetic Logic Unit (ALU)	Performs arithmetic and logical operations.
Registers	Temporary storage for data, instructions, and addresses.
Cache	High-speed memory for frequently used data and instructions.
Bus Interface Unit (BIU)	Manages communication between CPU and memory/I/O devices.
Floating Point Unit (FPU)	Performs complex floating-point mathematical calculations.
Instruction Decoder	Interprets instructions and generates control signals.
Clock Unit	Provides timing signals to synchronize CPU operations.
Memory Management Unit (MMU)	Manages virtual-to-physical address translation and memory protection.
Prefetch Unit	Fetches instructions ahead of time to reduce wait time.

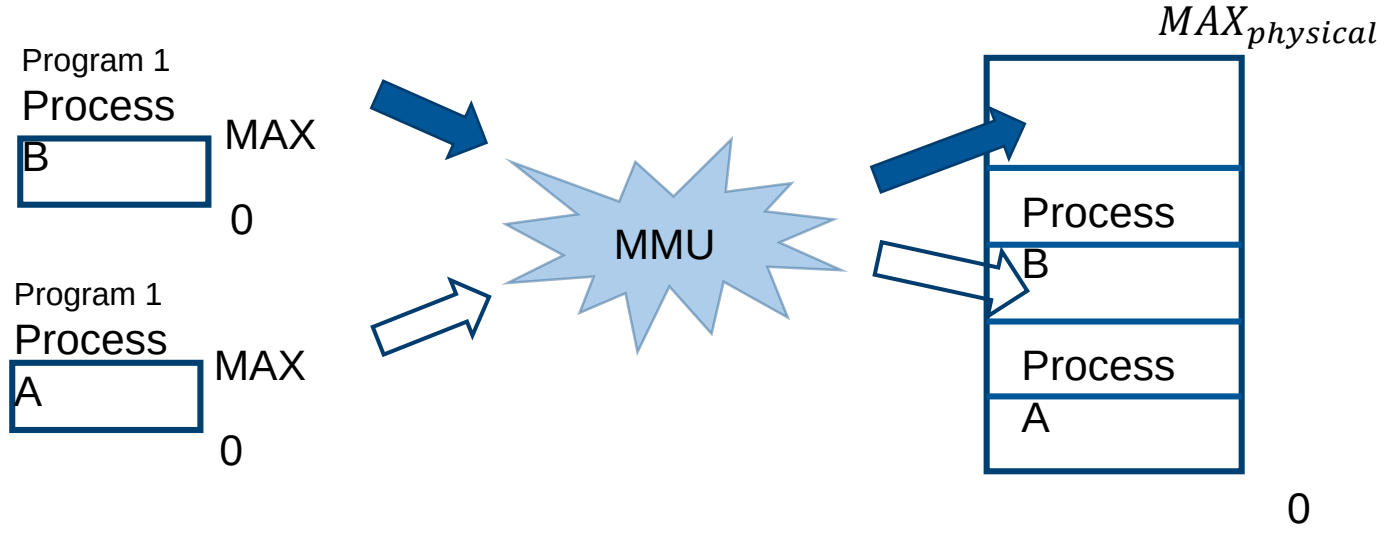
Computer Hardware
CPU Components: Register

- Are small, high-speed storage locations within the CPU.
  - Store data, instructions, and intermediate results temporarily during the execution of a program.
- Types of registers:
  - General type of register: Used for temporary data storage during program execution.
  - Special type of register:
    - Stack Pointer (SP): Points to the top of the current stack in main memory.
    - Status Register (Flags): Holds flags that reflect the current state of the CPU or the outcome of recent operations. (e.g., zero flag, carry flag).
    - Program Counter (PC): Holds the address of the next instruction to be executed.
    - Instruction Register (IR): Stores the current instruction being executed.
    - Accumulator (ACC): Used to store intermediate results of arithmetic and logic operations.

Computer Hardware

CPU Components: Memory Management Unit

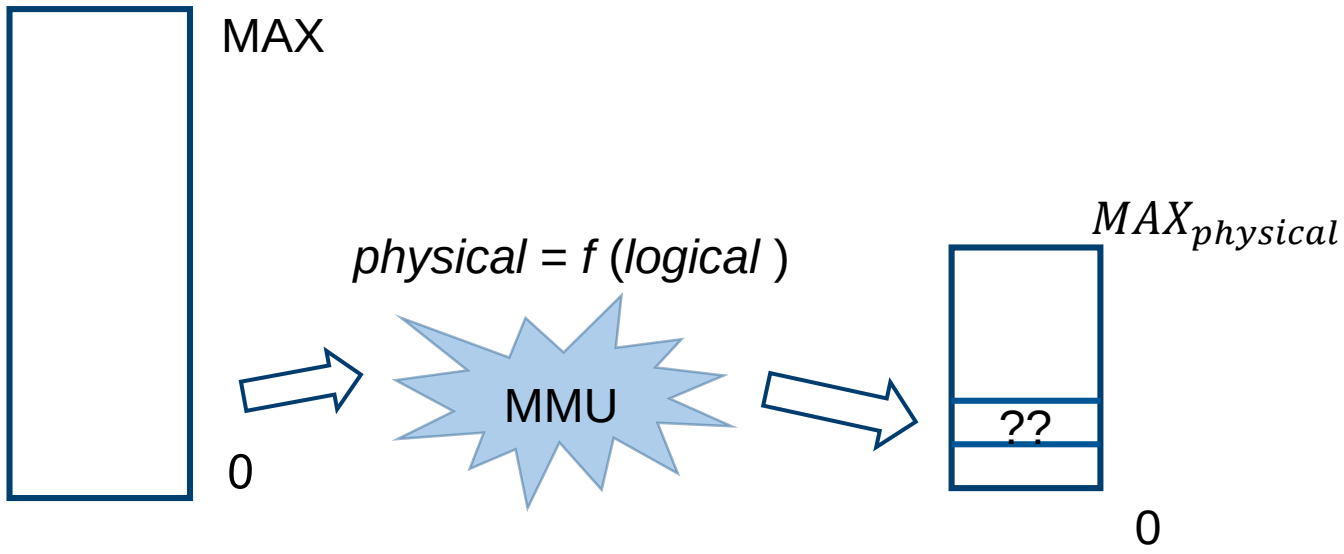
- The MMU handles the translation of virtual memory addresses (used by programs) to physical memory addresses (actual locations in RAM). It also manages memory protection and ensures that processes do not interfere with each other.
- At runtime, **multiple programs** (processes) will be running on a computer that **share memory**
- We **cannot know** beforehand in what **part of the address space** the code will run.



Computer Hardware

CPU Components: Memory Management Unit

- Modern computers use a **logical** and **physical** memory addresses:
  - Every process has a logical address space – [0,MAX_{64}] (theoretically),
  - (*e.g.* 32-bit and 64-bit CPU giving address space of 2^{32} or 2^{64} bytes, respectively.)
  - The machine has a physical address space – [0, *MAX*] (MAX determined by the amount of physical memory)
  - (*e.g.* modern personal computers have physical memory of 512 MB, 1 GB, 2GB,…)
- **Address translation** takes place in the **Memory Management Unit** (MMU)



Computer Hardware
CPU Design: Memory Management Unit

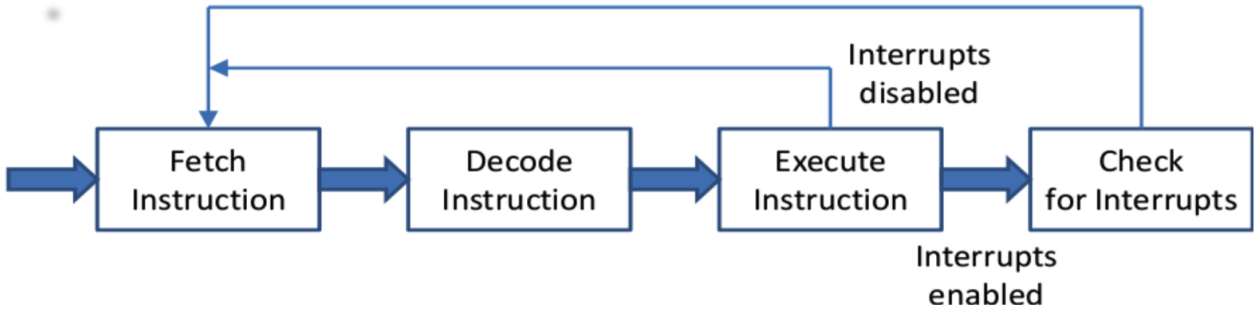
```
1  #include <stdio.h>
2
3  int iVar = 0;
4  main() {
5      int i = 0;
6      while(i < 10) {
7        iVar++;
8        sleep(2);
9        printf("Address:%x; Value:%d\n",&iVar, iVar);
10       i++;
11     }
12 }
```

- If running the code twice (simultaneously):
  - Will the same or different addresses be displayed for *x*
  - Will the value for *x* in the first run influence the value for *x* in the second run

Computer Hardware

CPU Design: Interrupts and Context Switching

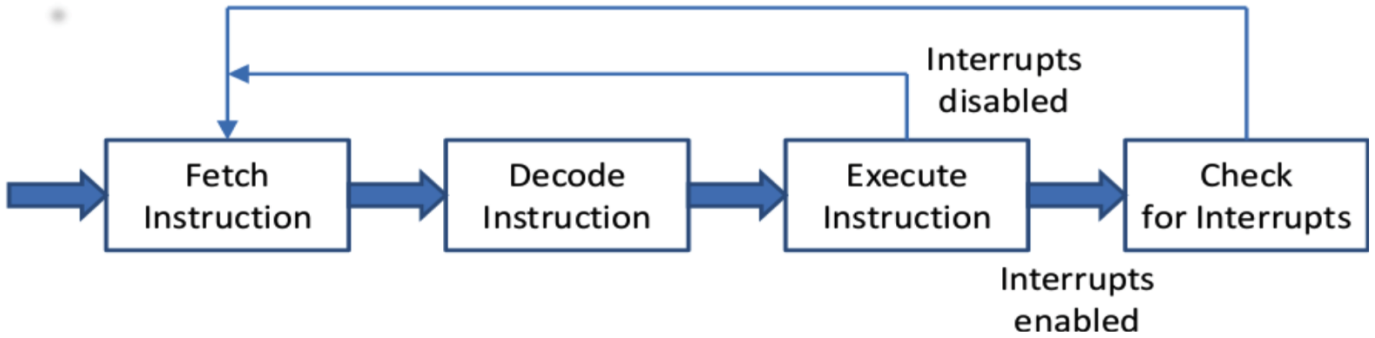
- Interrupts:
  - Signals that indicate an event requiring immediate attention, such as I/O completion or a system timer.
  - Cause the CPU to stop its current task, handle the event via an interrupt service routine (ISR), and then return to its previous task.
- Context Switching:
  - The process by which the CPU switches between different tasks (or processes) by saving and restoring their state (or context). This allows for multitasking in a system.
  - When time multiplexing (time-sharing) the CPU, the operating system will often stop the running program to (re)start another one.
  - Every time it stops a running program, the operating system must save all the registers so they can be restored when the program runs later.



Computer Hardware

CPU Design: Interrupts and Context Switching

- Different types of interrupts exist, including:
  - Timer interrupts by **CPU clock**
  - **I/O interrupts** for **I/O completion** or error codes
  - **Software generated**, e.g. errors and exceptions

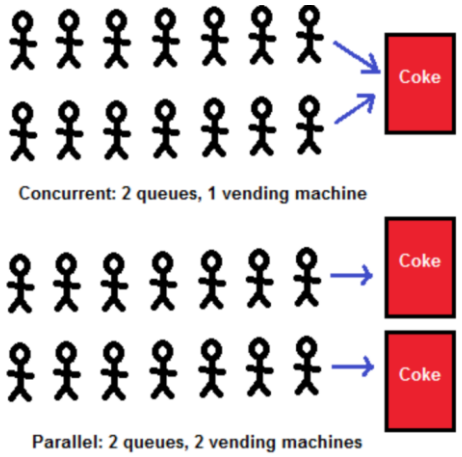


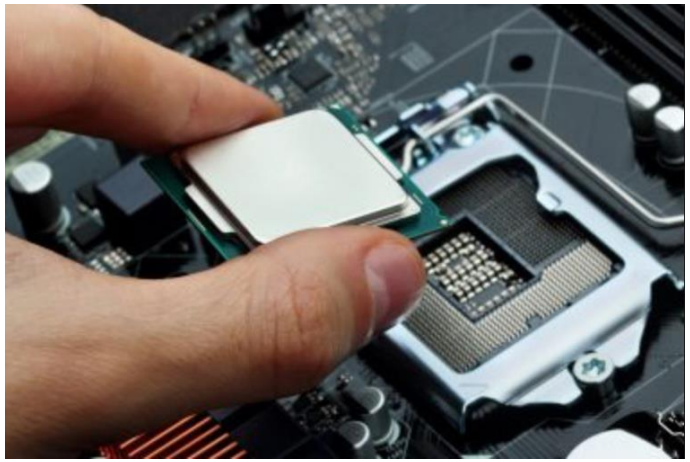
Computer Hardware
CPU Design: Timer Interrupts

1. Timer generates an interrupt
2. CPU finishes current instruction and tests for interrupt
3. Transfer to interrupt service routine
   - Hardware saves current process state (PSW, program counter)
   - Set program counter to interrupt service routine
   - Save registers and other state information
4. Carry out interrupt service routine (ISR) (scheduler)
   - The ISR (in this case, the scheduler) determines whether the currently running process should continue or if another process should be given CPU time.

Computer Hardware
CPU Design: Moore's "law"

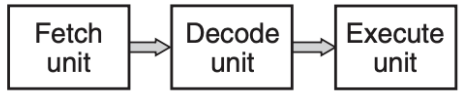
- Moore's law
  - *"The number of transistors on an integrated circuit (chip) doubles roughly every two years"*
- Problem:
  - What to do with abundance of transistor to increase the processing power?
  - How to use **massively parallel** computers/CPUs/many core machines
  - **Parallel vs. Concurrent**



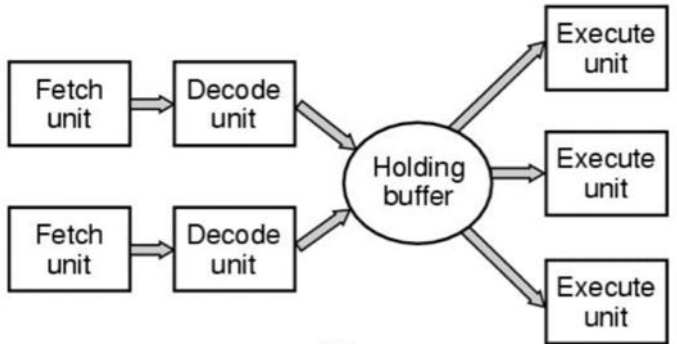


Computer Hardware
CPU Architecture

- **Pipelines:**
  - Allows the CPU to fetch, decode, and execute multiple instructions in different stages at the same time.
  - This increases the overall instruction throughput.

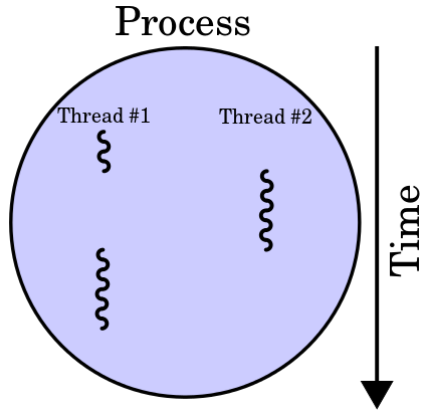


- **Superscalar CPUs:**
  - Superscalar CPUs can fetch and decode multiple instructions in parallel, and then assign them to different execution units (e.g., ALUs, FPUs) for processing.
  - These units operate in parallel, significantly improving performance.

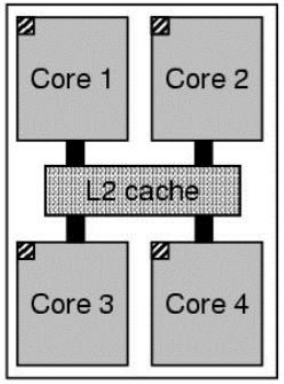


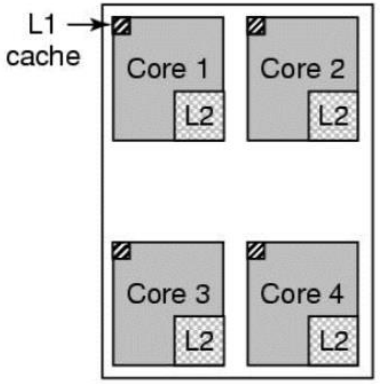
Computer Hardware
CPU Architecture

- **Multithreading:**
  - The CPU core can have multiple hardware threads (also called logical processors) that share the execution units of a single core.
  - Is a technique that allows a CPU to switch between multiple threads of execution within the same core. Each thread represents a separate sequence of instructions.



- **Multicore Chips (True Parallelism)**
  - A multicore processor is a CPU that contains multiple independent cores (processing units) on the same chip.
  - Each core can execute instructions independently of the others, essentially acting like a separate processor.





Computer Hardware
CPU Architecture: Real-World Example

- A modern Intel Core i7 or AMD Ryzen processor typically incorporates all of these technologies:
  - Pipelining to improve instruction throughput.
  - Superscalar execution and out-of-order execution to process multiple instructions at once.
  - Multithreading (e.g., Hyper-Threading) to improve the utilization of individual cores.
  - Multiple cores (e.g., 4, 8, 12, or more cores) to run many processes or threads in parallel.

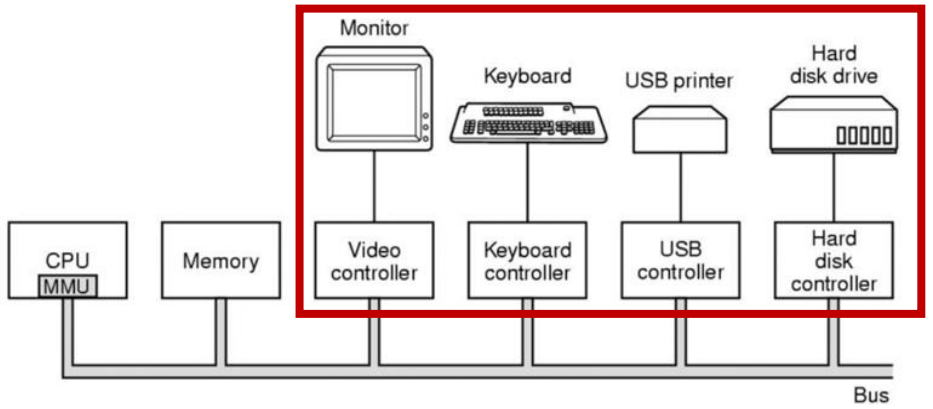
Computer Hardware
Can we reach to 100 GHz Pentium?

- **Three walls** prevent us from getting more operating power
  - The **memory** wall: the increasing **gap** between processor and memory speeds.
    - Modern CPUs are extremely fast, but the time it takes to access data from memory (RAM) has not improved as significantly.
  - The **ILP** (instruction level parallelism) wall: ILP refers to the ability of a processor to execute multiple instructions simultaneously within a single thread.
    - The increasing difficulty of finding enough **parallelism** in a single **instruction stream** to keep a high-performance single-core processor busy.
  - The **power** wall: faster computers get really hot and requires much more power.

Previous exam: Describe how, in your opinion, recent developments in computer architecture and computer design have influenced operating system design

Computer Hardware
I/O Devices

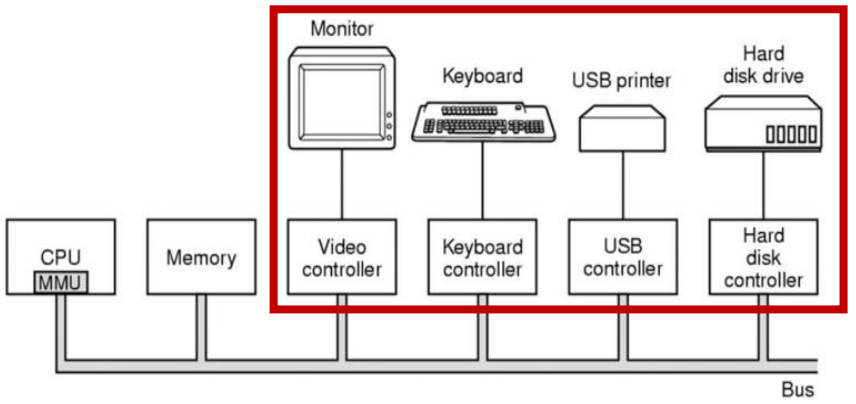
- The operating system manages I/O devices by providing an abstraction layer between the hardware and applications.
- **Device Driver:** A device driver is software that provides an interface between the operating system and a specific hardware device.
  - Abstracts the hardware-specific details and presents a standard interface for the OS.
- **Controller:** A controller is hardware responsible for managing a device, such as a disk or network interface.
  - It has a set of registers, each representing different commands, status flags, and data buffers.



Computer Hardware

I/O Devices: Communication via Registers:

- The OS/device driver communicates with the controller through **registers**.
  - These registers are memory-mapped or I/O-mapped, meaning they can be accessed via normal memory instructions or through specific I/O instructions (depending on the system architecture).
- The device driver typically reads from or writes to these registers to:
  - Send commands to the controller (e.g., read or write to disk).
  - Check the status of the controller (e.g., busy, idle, error).
  - Transfer data between the OS and the device.



Computer Hardware
I/O Devices

- I/O operations can be performed using:
  - Busy waiting (polling): where the CPU constantly checks the status of the device.
  - Interrupts: where the device informs the CPU when it is ready or when an operation is completed.
  - DMA (Direct Memory Access): where data is transferred directly between the device and memory, bypassing the CPU.
    - How it works
      - The CPU sets up the DMA controller by specifying the source (e.g., the I/O device), destination (memory address), and the amount of data to be transferred.
      - The DMA controller (not the CPU) handles the actual data transfer between the device and memory.
      - The DMA controller notifies the CPU (typically via an interrupt) that the transfer has finished.

Operating Systems Structure

- Operating Systems contain **a lot of functionality**, including:
  - Processes, process management, process scheduling, context switching, etc.
  - Memory management, virtual memory, etc.
  - File Systems
  - I/O Management
- Operating systems (OS) can be structured in various ways depending on how their core components are organized and how they interact with each other.
  - **Micro kernels**
  - **Monolithic**
  - **Hybrid Kernel**

Operating Systems Structure

Monolithic Kernels

- Single Address Space: All kernel services (e.g., process management, memory management, file systems, device drivers) run in the same memory space (privileged mode).
- Performance: Because everything runs in the same address space, inter-process communication (IPC) between different parts of the kernel is fast. No context switching is needed for calling kernel functions, making system calls faster.
- Complexity: The entire OS kernel is large and tightly coupled. A bug or crash in one component (e.g., a device driver) can potentially crash the entire system.
  - Examples: Linux, Windows (historically monolithic), Unix

Operating Systems Structure
Microkernels

- Minimal Kernel: The microkernel contains only the core services like low-level memory management, basic inter-process communication (IPC), and CPU scheduling.
- User-Space Services: All other services like device drivers, file systems, and network protocol stacks run in user space as separate processes.
- Communication: A microkernel relies heavily on IPC mechanisms (like message passing) to communicate between the kernel and user-space services.

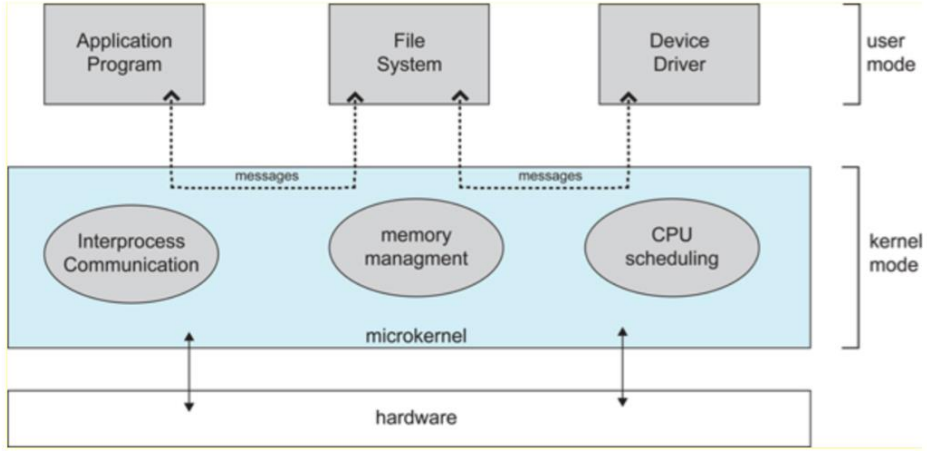


Figure: Structure of a Micro Kernel (Silberschatz, 2008)

Examples: Minix, QNX, Mach (the basis for macOS and iOS), Unix version, early versions of Windows (NT4.0) were (partially) micro kernels

Comparison of Monolithic and Microkernel architectures

Aspect	Monolithic Kernel	Microkernel
Structure	All core OS services run in the same address space.	Core OS services run in kernel space; others run in user space.
Performance	Faster system calls due to direct function calls.	Slower system calls due to IPC overhead and context switches.
Stability	Less stable—bugs in one part of the kernel can crash the system.	More stable—failures in user-space services don't crash the whole system.
Security	Less secure, as everything runs in the same address space.	More secure due to isolation between kernel and user-space services.
Ease of Maintenance	More difficult to maintain due to tightly coupled components.	Easier to maintain because services are modular and independent.

Operating Systems Structure

Comparison Between Monolithic and Microkernel Designs:

- A **hybrid kernel** is a compromise between the monolithic and microkernel architectures. It includes some aspects of both designs.
  - For example, core services like device drivers might run in kernel space (like in a monolithic kernel), but other services can be moved to user space (like in a microkernel).
- **Examples of Hybrid Kernel:**
  - **Windows NT kernel** (Windows 10, Windows Server)
  - **macOS kernel** (Darwin, based on Mach)

Recap

- Computer Hardware
  - Interrupts, context switching
- OS structures/implementation