운영체제의 기초: Review of Computer Hardware

2023년 3월 14, 16, 21일

홍 성 수

sshong@redwood.snu.ac.kr

SNU RTOSLab 지도교수

서울대학교 전기정보공학부 교수

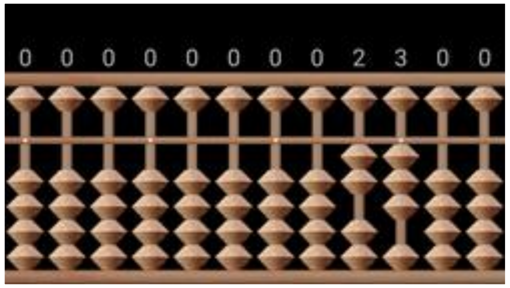
Seoul National University

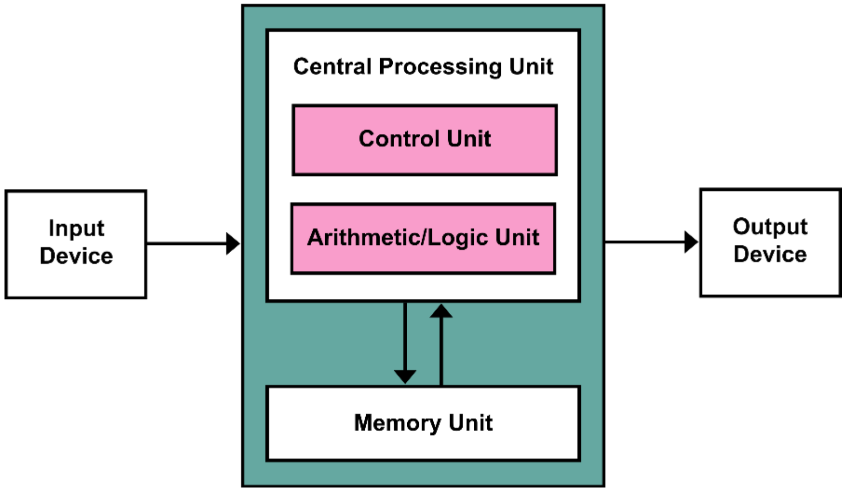
Agenda

I. Computer Systems Architecture

1. How "*Software*" Was Born? (1)

- Abacus vs. "*stored-program computer*"
  - Modern computing started with …
    - "*Stored-program computer*" (AKA "*von Neumann machine*")





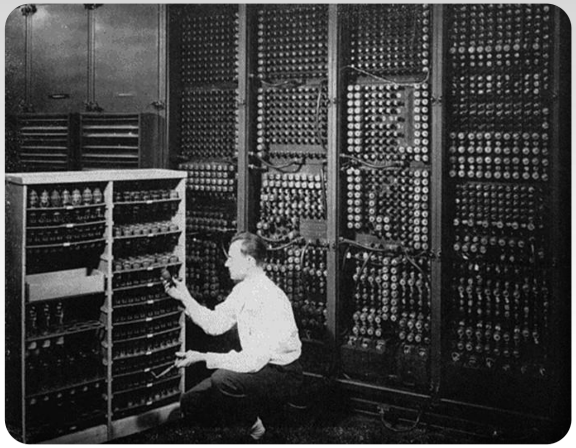
How "*Software*" Was Born? (2)

- "*Stored program concept*" …
  - Idea that code and data are stored together in memory
    - Instructions are then fetched from memory one at a time and executed (*fetch-decode-execute* (FDE) cycle)
  - Laid foundation for most computers used today
  - Described by Jon von Neumann in 1945
    - Hence named "*von Neumann Architecture*"
      - Aka Princeton architecture

Source: http://www.lanl.gov/history/atomicbomb/images/NeumannL.GIF

How "*Software*" Was Born? (3)

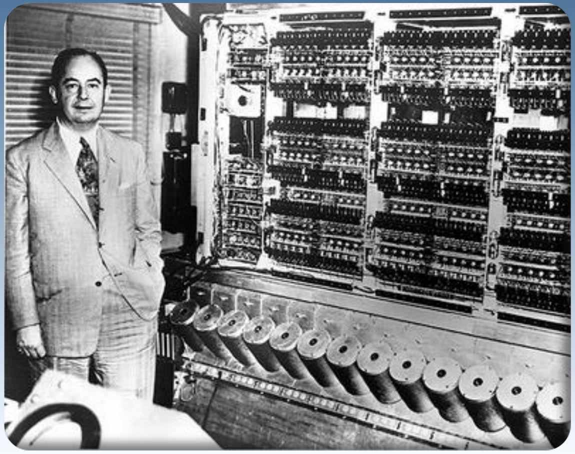
Hardware-Defined



1946년, 내부 프로그래밍을 구현하지 못해 프로그래밍을 하려면 외부에서 배선을 일일이 변경해 줘야 하는 방식과 기억과 연산을 10진 방식을 채용한 컴퓨터

ENIAC
(Electronic Numerical Integrator and Computer)

Software-Defined



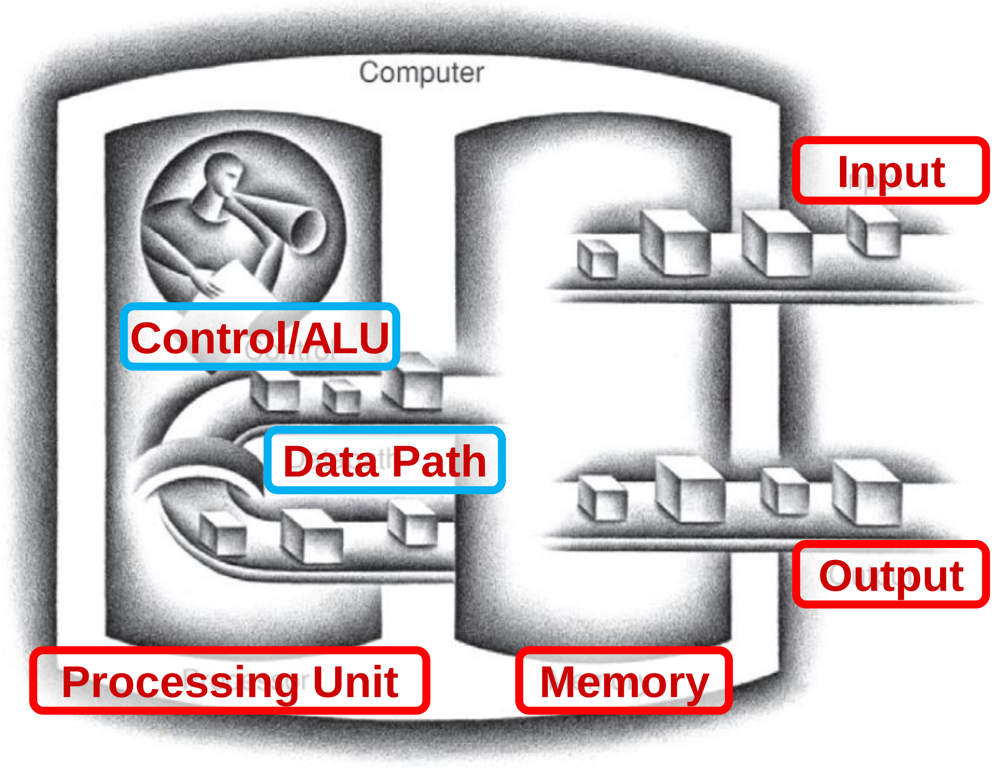
1950년, 최초의 이진수를 사용한 프로그램 내장 (Stored-Program) 컴퓨터

EDVAC
(Electronic Discrete Variable Automatic Computer)

Components inside Computer (1)

- “von Neumann architecture” must have
  1. “*Processing unit*”
     - “Arithmetic logic unit” and various “*registers*”
     - “*Control unit*”, an “*instruction register*” and “*program counter*”
  2. “*Memory*” that stores data and instructions
  3. “*Input* and *output*” devices

Components inside Computer (2)



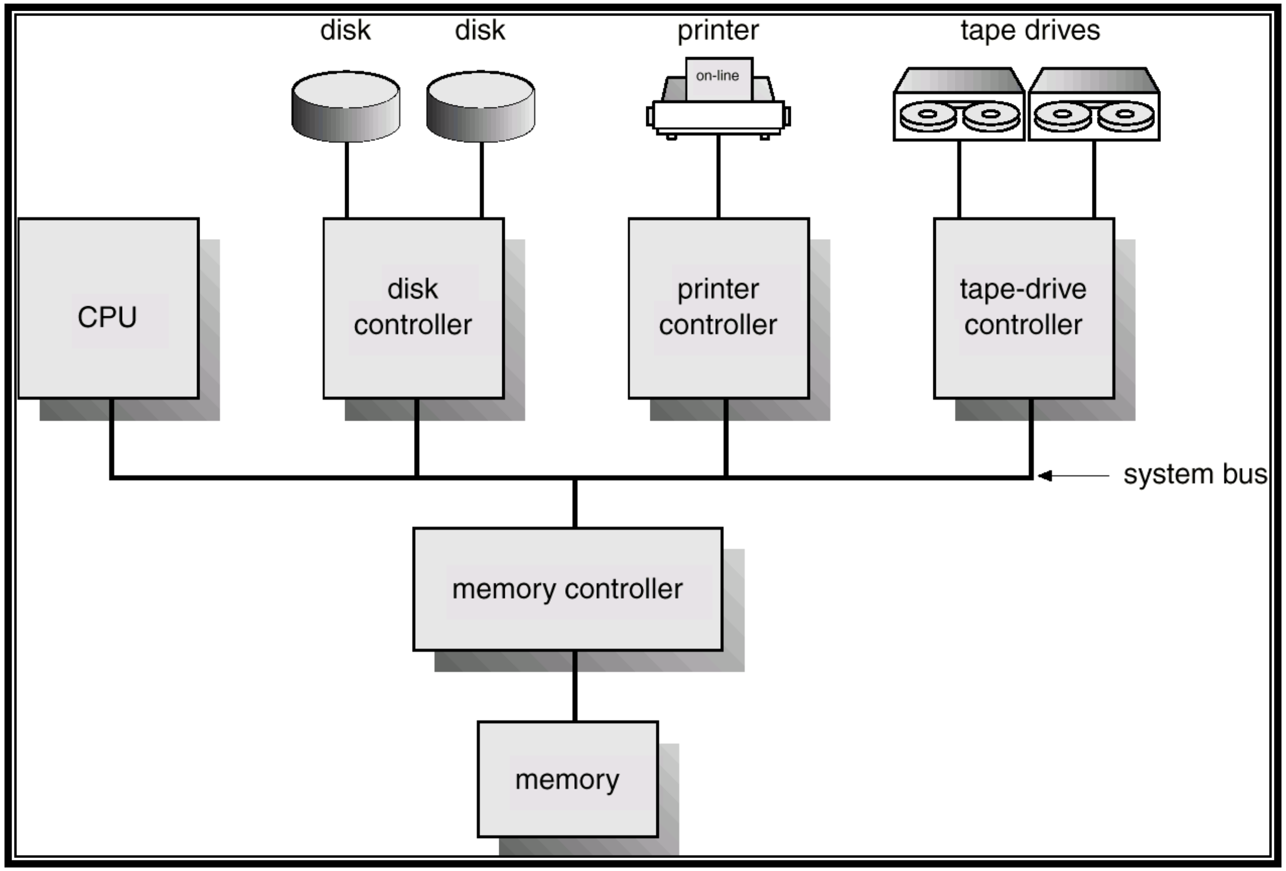
Components inside Computer (3)

- Processing unit (Control/ALU)
  - Component of the processor that commands the data path, memory, and I/O devices according to the instructions of the program
- Memory
  - Storage area where the running programs and their data are kept

Components inside Computer (4)

- Input/Output
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers
- Data path
  - Pathway used to transfer data and instructions between the components
  - Aka "*system interconnect*" or "*system bus*"

Components inside Computer (5)



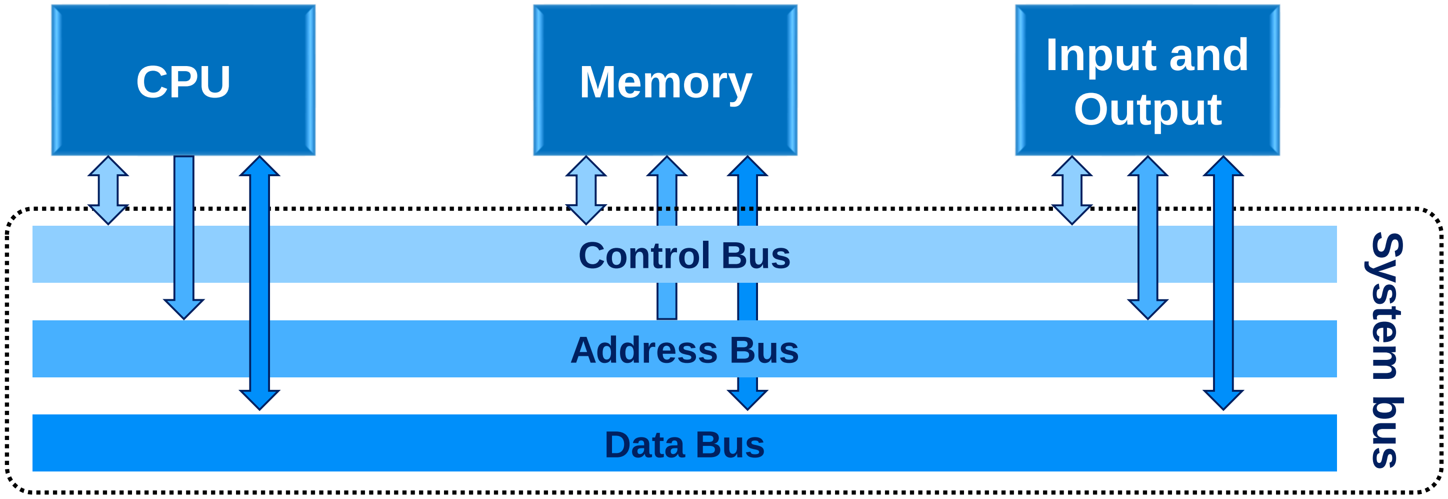
Source: Silberschatz, Galvin and Gagne, Operating System Concepts, 1998

Representation of Data

- Bits/Bytes
  - Bit: short for “binary digit”
  - 1 byte = 8 bits
- Words
  - 32-bit data called a “word” (in 32-bit machines)
  - “Word” is a basic data unit
- Numbers
  - Fixed-point numbers (integers)
  - Floating-point number (real numbers)

System Interconnect (1)

- System bus
  - Connects the major components of a computer system



System Interconnect (2)

- System bus
  - Consists of two types of buses
    - *Data* bus
      - Data pathway between bus master and slave
    - *Address* bus
      - Specifies the target location of data transfer
    - Control bus
      - Carries commands from the CPU and returns status signals from the controller of the devices
  - Two types of bus transactions
    - *Read*/*Write*

System Interconnect (3)

- Bus arbiter
  - Arbitrates conflict among multiple bus requests
    - Accepts bus request signal
    - Replies with bus grant signal
- Bus master
  - Can initiate bus transaction by sending bus request signal
  - *CPU*
    - Moves data between main memory and CPU registers
  - DMA controller
    - Moves data between main memory and I/O buffer without the help of CPU

System Interconnect (4)

- Bus slave
  - Takes command from bus master and serve it accordingly
  - *Memory controller (memory)*
  - *Device controller (I/O device)*
    - In charge of a particular device type
    - Has local registers and/or local buffer
    - Inform CPU that it has finished its operation by causing *interrupt*
      - Note that I/O devices and CPU can execute concurrently
    - I/O is from the device to local buffer of controller

I/O Operations (1)

- Performed by “I/O controller” under CPU command
  - Has registers
    - *Data* registers: input register, output register
    - *Control* register: control register, status register
  - I/O operations are initiated by CPU
    - Output operation
      - Check if output register is available by reading in status register
      - If so, move data to output register; move output command to control register
      - Otherwise, either repeat this process or wait until output register is available
    - *Polling* I/O vs. *interrupt-driven* I/O

I/O Operations (2)

- Two types of I/O addressing
  - Memory-mapped I/O
    - I/O registers are associated with memory locations
    - Uses the same address bus to address both memory and I/O
    - Uses *Load*/*Store* instructions for input/output
  - Port-mapped I/O
    - I/O registers are associated with special I/O addresses port numbers
      - Has separate I/O address space
    - Uses special I/O bus
    - Uses special *Input*/*Output* instructions

DMA (Direct Memory Access)

- DMA
  - Allows device controller to directly transfer block of data between buffer storage and main memory *without CPU intervention*
  - Only one interrupt is generated *per block*, rather than the one interrupt per byte
  - Used for *high-speed I/O devices* able to transmit information at close to memory speeds

II. Interrupt Mechanism

Basics

- Interrupt
  - Hardware mechanism that transfers control to *interrupt service routine* (*ISR*)
- Types of interrupts
  - Hardware interrupt
    - Caused by hardware signal
    - Asynchronous
  - Software interrupt (AKA *trap*)
    - Caused either by an error or instruction
    - Synchronous
- OS is *interrupt* driven

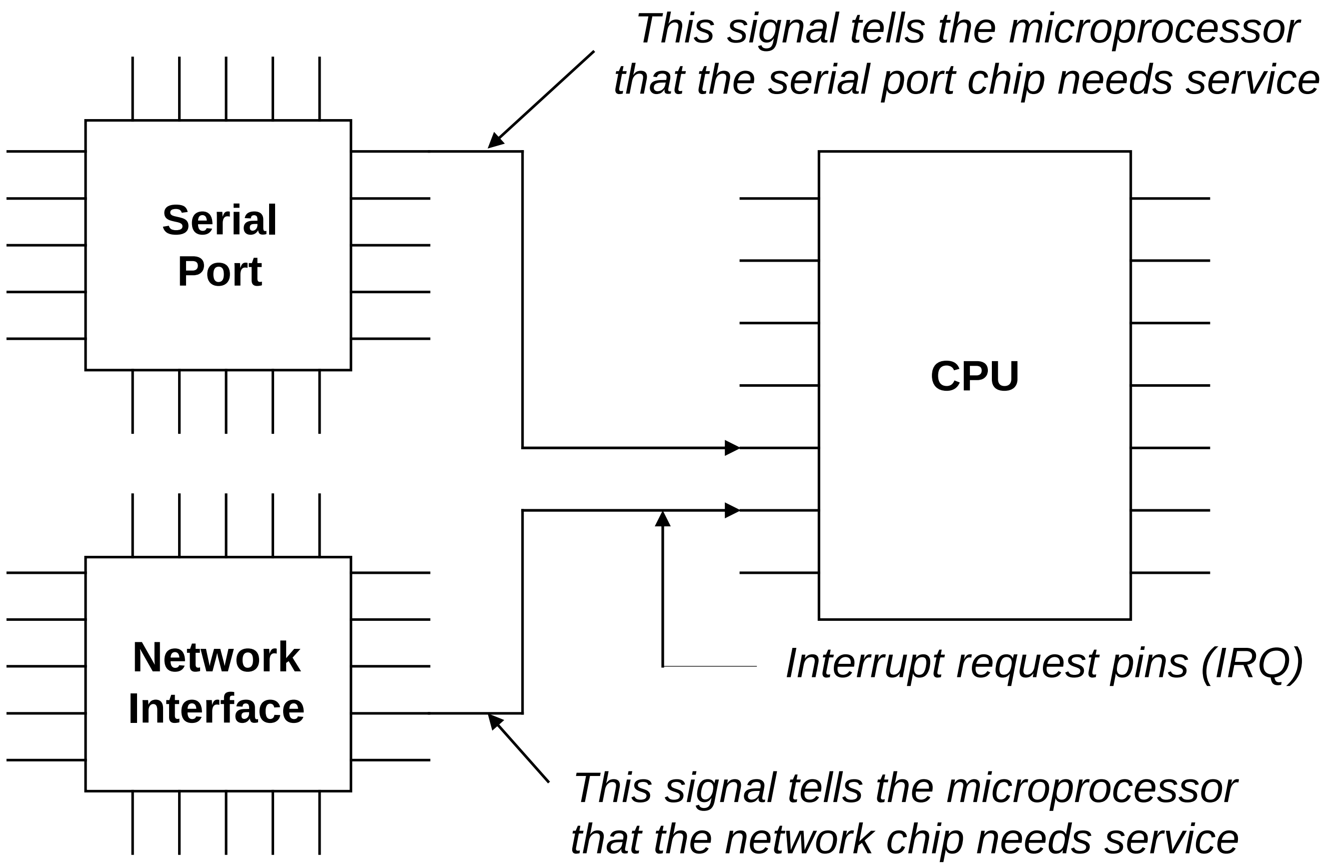
Interrupt Operation

- *Interrupt mechanism*
  - At the time of interrupt,
    - Stops the execution of the current program
    - Saves the address of the interrupted instruction
    - Gets the address of ISR via *interrupt request* (*IRQ*) number and *interrupt vector table*
    - Jumps to the ISR
  - While interrupt is being processed,
    - Incoming interrupts are *disabled* to prevent a *lost interrupt*
  - After the execution of ISR,
    - Returns to the interrupted program via the save address

Interrupt Hardware (1)

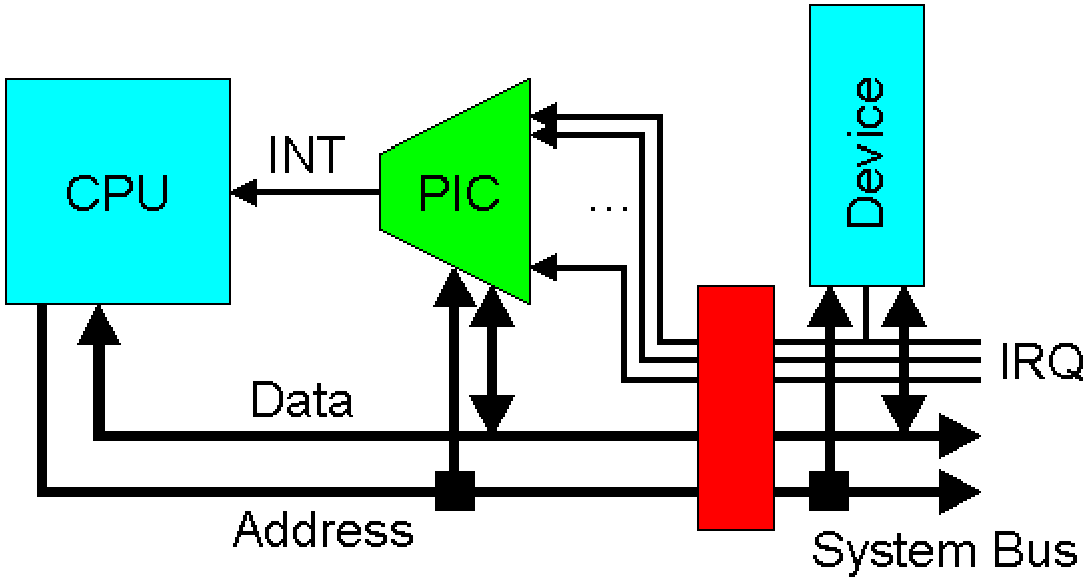
- Interrupt destination
  - Microprocessor has input pin called “*interrupt request (IRQ)*” that let the microprocessor know that some other chip in the system needs attention
  - Interrupts start with a signal from hardware
- Interrupt source
  - I/O chip has a pin that it asserts when it requires service

Interrupt Hardware (2)


This signal tells the microprocessor that the serial port chip needs service
Serial Port
CPU
Network Interface
Interrupt request pins (IRQ)
This signal tells the microprocessor that the network chip needs service

Interrupt Hardware (3)

- PIC (programmable interrupt controller)
  - Functions as an overall manager in an interrupt-driven system environment
  - Can support more I/O devices than the number of IRQ pins



Source: http://courses.engr.illinois.edu/ece390/lecture/lockwood/interrupt-hardware.gif

Interrupt Hardware (4)

- PIC operation
  - Monitors IRQ lines, checking for raised signals
  - If a raised signal occurs on an IRQ line:
    - Converts the received raised signal into a corresponding vector
    - Stores the vector in an interrupt controller I/O port, thus allowing the CPU to read it via the data bus
    - Sends a raised signal to the processor, INTR pin – that is, issues an interrupt
    - Waits until the CPU acknowledges the interrupt signal by writing into one of the PIC I/O ports; when this occurs, clears the INTR line
  - Goes back to the first step

III. Hardware Protection Mechanisms

Four Hardware Protection Mechanisms

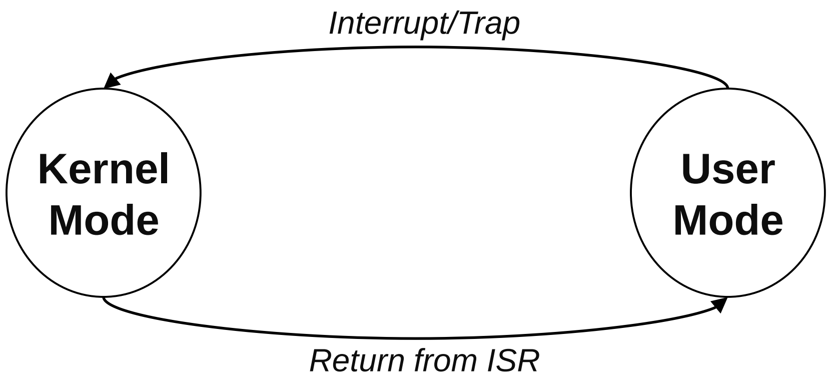
1. Basic mechanism: Dual mode operation
2. I/O protection
3. Memory protection
4. CPU protection

1. Dual Mode Operation (1)

- Motivation
  - Sharing system resources requires OS to ensure that an incorrect program cannot cause other programs to execute incorrectly
- Key idea
  - Provides hardware support to differentiate between at least two modes of operations
    - *User* mode
      - Execution done on behalf of a user
    - *Kernel* mode (AKA *system* or *monitor* mode)
      - Execution done on behalf of OS
      - When executing in kernel mode, OS has unrestricted access to both kernel and user's memory

1. Dual Mode Operation (2)

- *Mode bit* added to computer hardware, particularly in *processor status register*, to indicate the current mode
  - 0: kernel mode
  - 1: user mode
- When an interrupt or trap occurs hardware switches to kernel mode
- Provides special instructions called *privileged instructions* which can be executed only in kernel mode

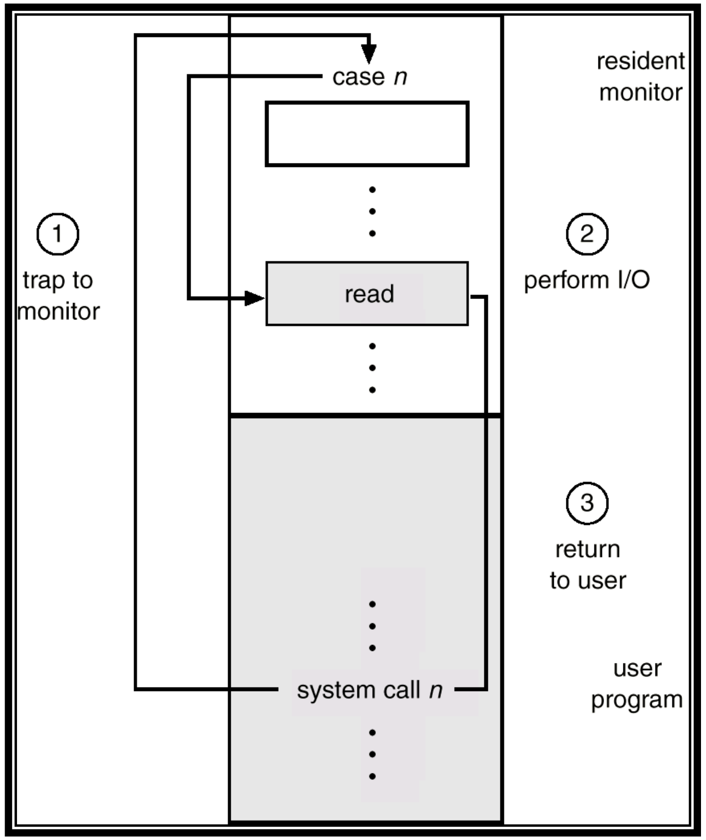


1. Dual Mode Operation (3)

- *System call*
  - A way of a user program invoking a kernel function in kernel mode
  - Always involves mode change from user to kernel mode
  - Compare it with function call

1. Dual Mode Operation (4)

- System call handling



Source: Silberschatz, Galvin and Gagne, Operating System Concepts, 1998

2. I/O Protection

- Motivation
  - Prevent I/O devices from being monopolized
- Key idea
  - All I/O instructions are *privileged* instructions
    - Must ensure that a user program can never gain control of the computer in kernel mode
    - Example:
      - A user program that, as part of its execution, stores a new address in the interrupt vector

3. Memory Protection

- Motivation
  - Protect memory outside the defined range
- Key idea
  - Add two registers that determine the range of legal addresses a program may access
    - Base register
      - Holds the smallest legal physical memory address
    - Bound register
      - Contains the size of the range
  - The load instructions for the base and bound registers is a privileged instruction

4. CPU Protection

- Motivation
  - Prevent CPU from being monopolized
- Key idea
  - Add timer which interrupts computer after specified period to ensure OS maintains control
    - Timer is decremented every clock tick
    - When timer reaches the value 0, an interrupt occurs
    - The load instruction for time is a privileged instruction