# Introductory Concepts

# Functionality comes with complexity?

- Every piece of computer hardware different
  - Different CPU - Pentium, PowerPC, ARM, MIPS
  - Different amounts of memory, disk, ...
  - Different types of devices
    - Mice, Keyboards, Sensors, Cameras, Fingerprint readers
  - Different networking environment
    - Cable, DSL, Wireless, Firewalls,...

# Key Questions

- Questions:
  - Does the programmer need to write a single program that performs many independent activities?
  - Does every program have to be altered for every piece of hardware?
  - Does a faulty program crash everything?
  - Does every program have access to all hardware?

# The OS paradigm

## Two main functions:

**Manage physical resources:**

- It drives various devices
  - Eg: CPU, memory, disks, networks, displays, cameras, etc
- Efficiently, reliably, tolerating and masking failures, etc

**Provide an execution environment to the applications running on the computer (programs like Word)**

– Provide virtual resources and interfaces
  - Eg: files, directories, users, threads, processes, etc

– Simplify programming through high-level abstractions

– Provide users with a stable environment, mask failures

# System design is Complex

- OS are a class of exceptionally complex systems
  - They are large, parallel, very expensive, not understood
    - OS implementation : years, large groups of people
  - Complex systems are the most interesting:
    - Internet, air traffic control, weather forecast, etc.
- How to deal with this complexity?
  - Abstractions and layering
  - Goal: systems trusted with sensitive data and critical roles

# What are the issues in OS / System Design?

- Structure: how is an operating system organized ?
- Sharing: how are resources shared among users ?
- Naming: how are resources named by users or programs ?
- Protection: how is one user/program protected from another ?
- Security: how to authenticate, control access, secure privacy ?
- Performance: why is it so slow ?
- Reliability and fault tolerance: how do we deal with failures ?
- Extensibility: how do we add new features ?

# What are the issues in OS / System Design? ...cont'd

- Communication: how can we exchange information ?
- Concurrency: how are parallel activities created and controlled ?
- Scale, growth: what happens as demands or resources increase ?
- Persistence: how can data outlast processes that created them
- Compatibility: can we ever do anything new ?
- Distribution: accessing the world of information
- Accounting: who pays bills, and how to control resource usage

# Why study Operating Systems?

- Operating systems are a maturing field
  - Most people use a handful of mature OSes
  - Hard to get people to switch operating systems
  - Hard to have impact with a new OS
- High-performance servers are an OS issue
  - Face many of the same issues as OSes
- Resource consumption is an OS issue
  - Battery life, radio spectrum, etc.
- Security is an OS issue
  - Hard to achieve security without a solid foundation
- New “smart” devices need new OSes

**Big Data, Networks, large-scale software systems, whatever --- Key to its success is Operating System Design Principles, its working and optimal use of its policies.**

# What is an OS?

- Tool to make programmer's job easy
- Resource allocator / manager
  - Must be fair; not partial to any process, specially for process in the same class
  - Must discriminate between different class of jobs with different service requirements
  - Do the above efficiently
    - Within the constraints of fairness and efficiency, an OS should attempt to maximize throughput, minimize response time, and accommodate as many users as possible
- Control program
- Tool to facilitate efficient operation of a computer system
- Virtual machine that is easier to understand and program
  - Encapsulates the complexities of hardware

# OS as Resource Manager

- A computer is a set of resources for the movement, storage, and processing of data.
  - Data could be associated with programs, devices, and even the underlying hardware.
- The OS is responsible for managing these resources.



15

# OS as Service Provider

- Program development
  - e.g., editors and debuggers
- Program execution
- Access I/O devices
- Controlled access to files
- System access for shared systems
- Error detection and response
  - e.g., memory error, device failure, division by zero
- Accounting for resources and performance monitoring

**How about Operating System as a Service? - Think about it!**
**We are in the age of Cloud Computing – How OS changes?**



20

# Operating System

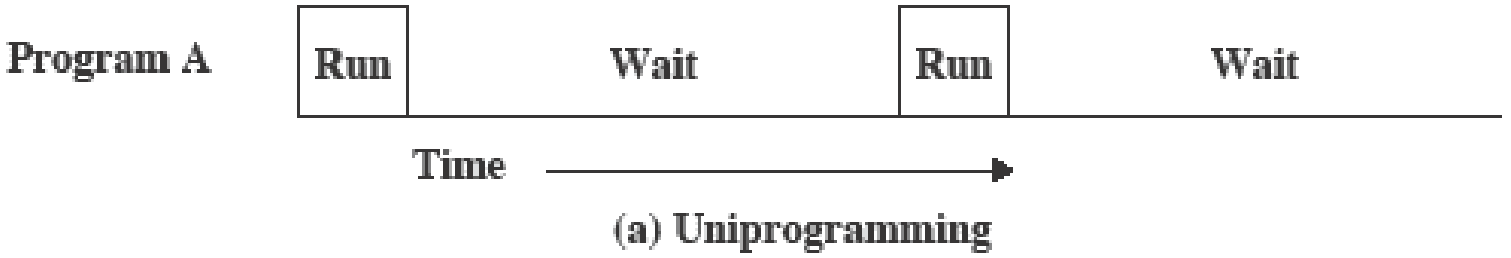
- A program that controls the execution of application programs
- An interface between applications and hardware
- Main objectives of an OS:
  - Convenience
  - Efficiency and Fairness
  - Ability to evolve



21

# Uniprogramming

- Processor must wait for I/O instruction to complete before preceding



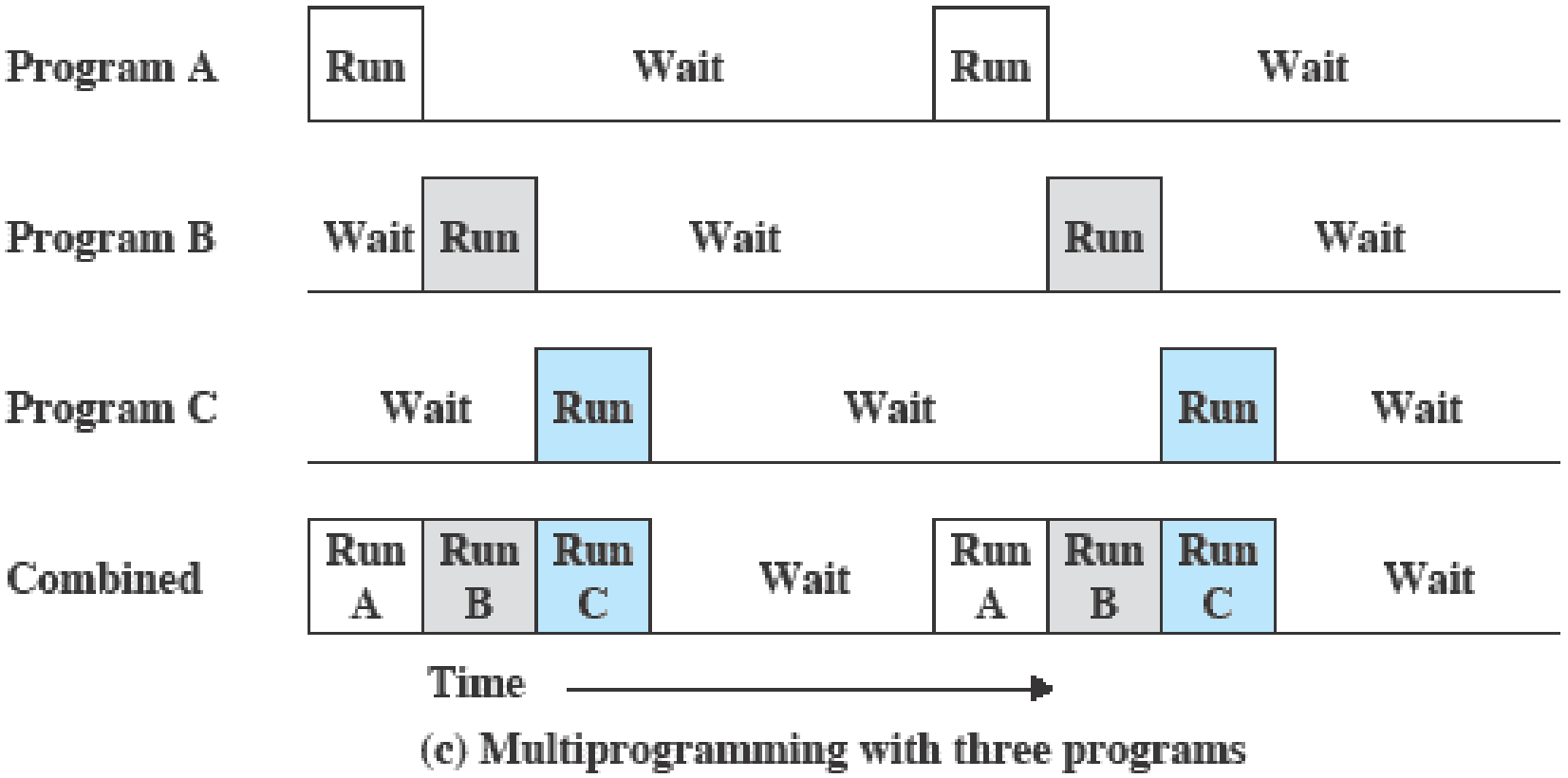

(a) Uniprogramming



31

# Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job




(c) Multiprogramming with three programs

**Idea of degree of multiprogramming – solution to avoiding CPU being idle**
**But what is the effect on Response time?**



32

# Time Sharing Systems

- Using multiprogramming to handle multiple interactive jobs
- Processor’s time is shared among multiple users



33

# OS Concepts

- Kernel
  - Permanently resides in the main memory
  - Controls the execution of processes by allowing their creation, termination or suspension, and commn.
  - Schedules processes fairly for execution on the CPU
  - Allocates main memory for an executing process
  - File system maintenance
  - Allows processes controlled access to peripheral devices such as terminals, tape drives, disk drives, and network devices.
  - Design options – Monolithic kernels, Layered, Micro kernels.

- Processes share the CPU in a time-shared manner
  - CPU executes a process
  - Kernel suspends it when its time quantum elapses
  - Kernel schedules another process to execute
  - Kernel later reschedules the suspended process

- Kernel
  - Permanently re[illegible]e main memory
  - Controls the e[illegible]tion of processes by allowing their creation, ter[illegible]ation or suspension, and commn.
  - Schedules processes fairly for execution on the CPU
  - Allocates main memory for an executing process
  - File system maintenance
  - Allows processes controlled access to peripheral devices such as terminals, tape drives, disk drives, and network devices.
  - Design options – Monolithic kernels, Layered, Micro kernels.

# OS Concepts

- K

Allows processes to share portions of their address space under certain conditions, but protects the private address space of a process from outside tampering
If system runs low on free memory, the kernel frees memory by writing a process temporarily to secondary memory, or swap device
If kernel writes entire processes to a swap device, then the implementation is called a swapping system; if it writes pages of memory to a swap device, it is called a paging system.
Coordinates with the machine hardware to set up a virtual to physical address that maps the compiler generated addresses to their physical addresses

- Allocates main memory for an executing process
- File system maintenance
- Allows processes controlled access to peripheral devices such as terminals, tape drives, disk drives, and network devices.
- Design options – Monolithic kernels, Layered, Micro kernels.

# OS Concepts

- Kernel
  - Permanently resides in the mai
  - Controls the execution of proce
    creation, termination or suspen
  - Schedules processes fairly for
  - Allocates main memory for
  - File system maintenance
  - Allows processes controlled access to peripheral devices such as terminals, tape drives, disk drives, and network devices.
  - Design options – Monolithic kernels, Layered, Micro kernels.

Allocates secondary memory for efficient storage and retrieval of user data

Allocates secondary storage for user files

Reclaims unused storage

Structures the file system in a well understood manner

Protects user files from illegal access

# OS Concepts - Program

- Program
  - Collection of source code instructions and any associated data kept in a disk file
  - Source program, or a human-readable text file.
  - Machine language translation of the source program, or object file
    - The file is marked as executable.
    - File contents are arranged according to rules established by the kernel
    - Executable program, complete code output by linker / loader, with input from libraries

# Example of Program Execution

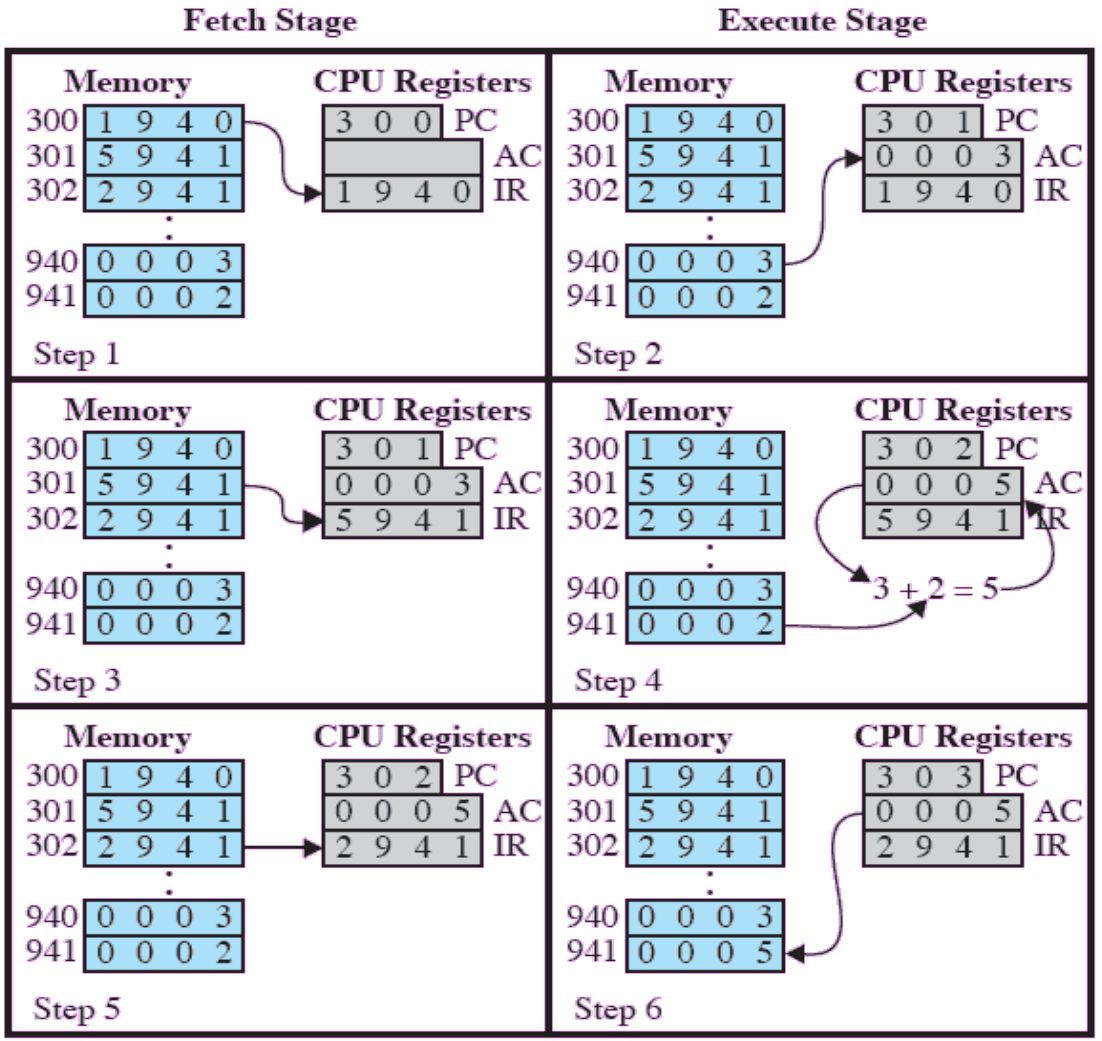



**Figure 1.4 Example of Program Execution**
**(contents of memory and registers in hexadecimal)**



40

# Instruction Execution

- A program consists of a set of instructions stored in memory
- Two steps
  - Processor reads (fetches) instructions from memory
  - Processor executes each instruction

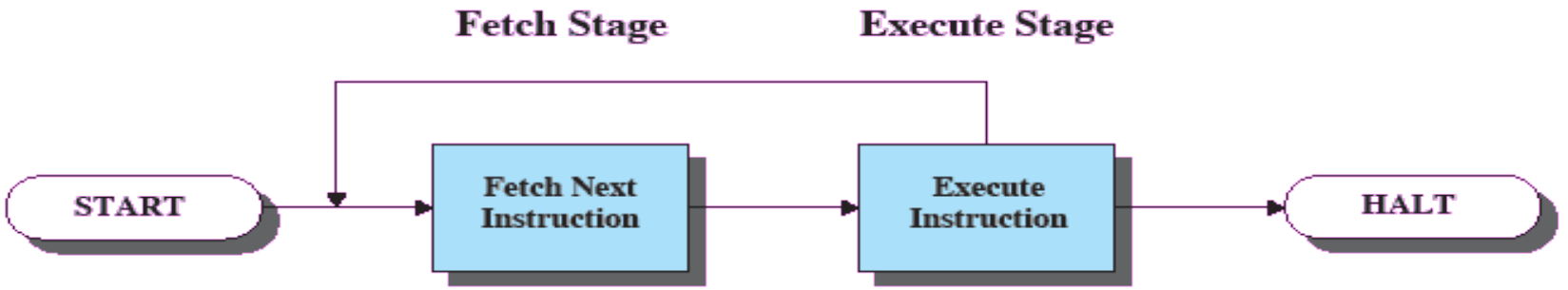



Figure 1.2 Basic Instruction Cycle



41

# OS Concepts

- Memory (Volatility)
  - Memory hierarchy based on storage capacity, speed, and cost
  - Higher the storage capacity, lesser the speed, and lesser the cost
  - Different memory levels, in decreasing cost per byte of storage
  - Use hierarchical memory to transfer data from lower memory to higher memory to be executed
  - Locality of reference
    - Most of the references in the memory are clustered and move from one cluster to the next
  - Cache memory
    - Use of very fast memory (a few kilobytes) designated to contain data for fast access by the CPU
  - Virtual memory or extension of main memory
  - Disk cache
    - Data-intensive applications (generally rotational speed and seek time)

# OS Concepts

- Memory (Volatility)
  - Memory hierar
  - Higher the st
  - Different m
  - Use hierarchical memory to transfer data from lower memory to higher memory to be executed
  - Locality of reference
    - Most of the references in the memory are clustered and move from one cluster to the next
  - Cache memory
    - Use of very fast memory (a few kilobytes) designated to contain data for fast access by the CPU
  - Virtual memory or extension of main memory
  - Disk cache
    - Data-intensive applications (generally rotational speed and seek time)

| | | |
|---|---|---|
| Registers | Few bytes | Almost CPU speed |
| Cache memory | Few kilobytes | Nanoseconds |
| Main memory | Megabytes | Microseconds |
| Magnetic disk | Gigabytes | Milliseconds |
| Magnetic tape/Optical disk | No limit | Offline storage |

# OS Concepts

- Memory (Volatility)
  - Mem
  - High
  - Diffe
  - Use
    high
  - Loca
    -
  - Cache
    - Use of v y (a few kilobytes) designated to contain data for fast ac CPU
  - Virtual me or extension of main memory
  - Disk cache
    - Data-intensive applications (generally rotational speed and seek time)

A disk cache is a mechanism for improving the time it takes to read from or write to a hard disk.

Today, the disk cache is usually included as part of the hard disk.
**(A disk cache is RAM built into your hard disk)**
(Disk specifications : on-board cache)
OR
A disk cache can also be a specified portion of system RAM (disk buffer).

Disk cache holds data that has recently been read and, in some cases, adjacent data areas that are likely to be accessed next.

Write caching (accumulated) is also provided with some disk caches.

# OS Concepts – System Calls

Interface between user program and operating system - Set of extended instructions provided by the operating system

- Applied to various software objects like processes and files
- Invoked by user programs to communicate with the kernel and request services
- Access routines in the kernel that do the work
- Library procedure corresponding to each system call

Why modes? Privileges? - indicates power for System control

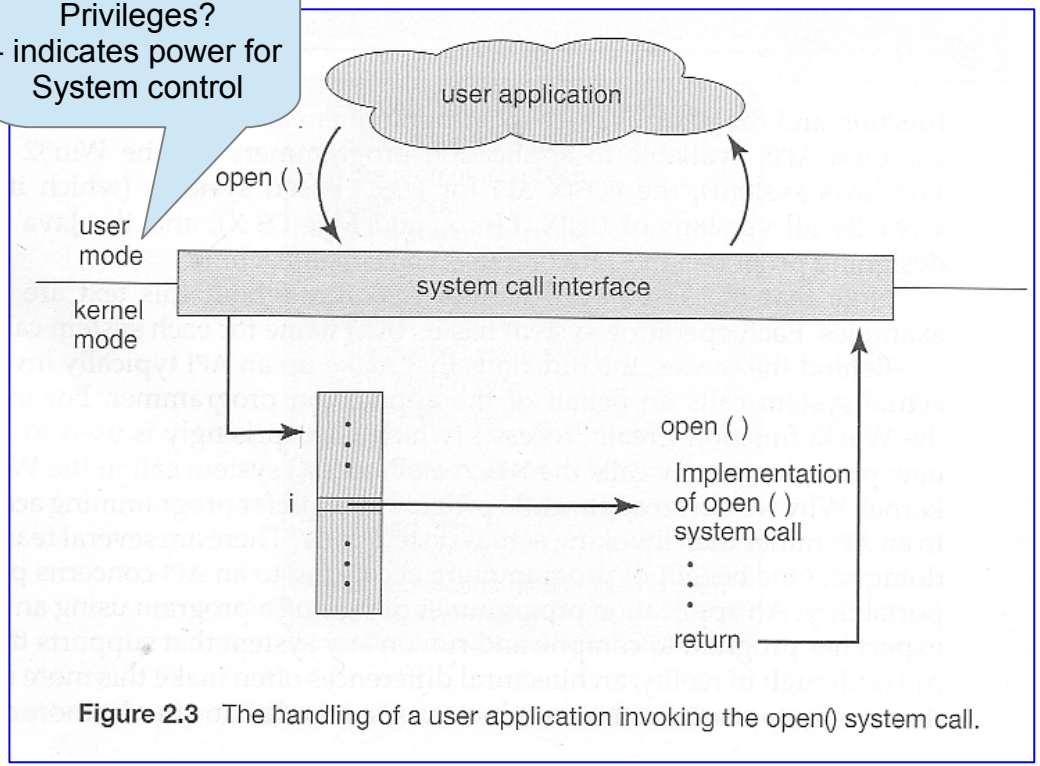



Figure 2.3 The handling of a user application invoking the open() system call.

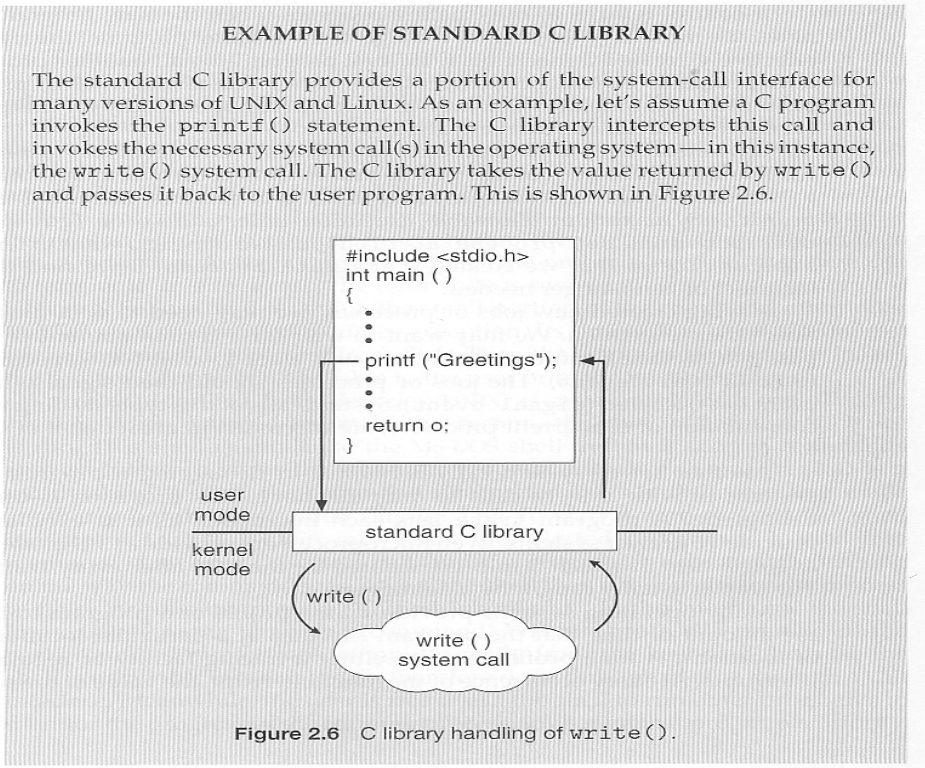



Figure 2.6 C library handling of write().

Working of a System Call – How?

# Interrupts

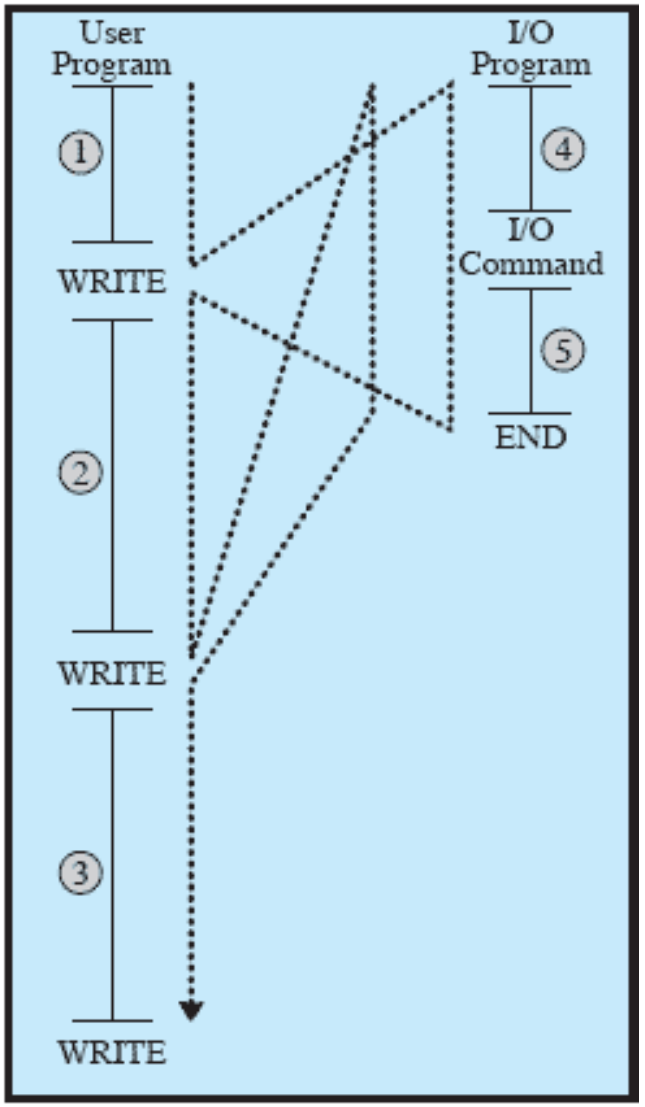
- Interrupt the normal sequencing of the processor
- Provided to improve processor utilization
  - Most I/O devices are slower than the processor
  - Processor must pause to wait for device



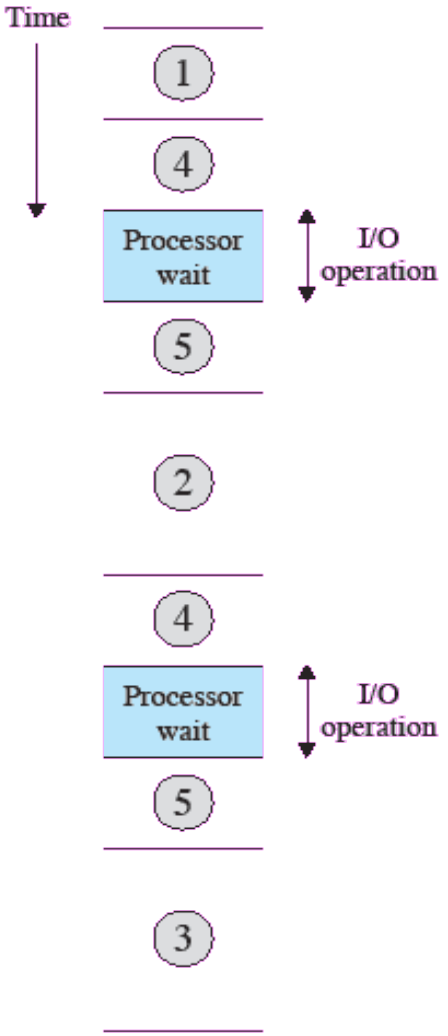
47

# Flow of Control w/o Interrupts and I/O Wait

Figure 1

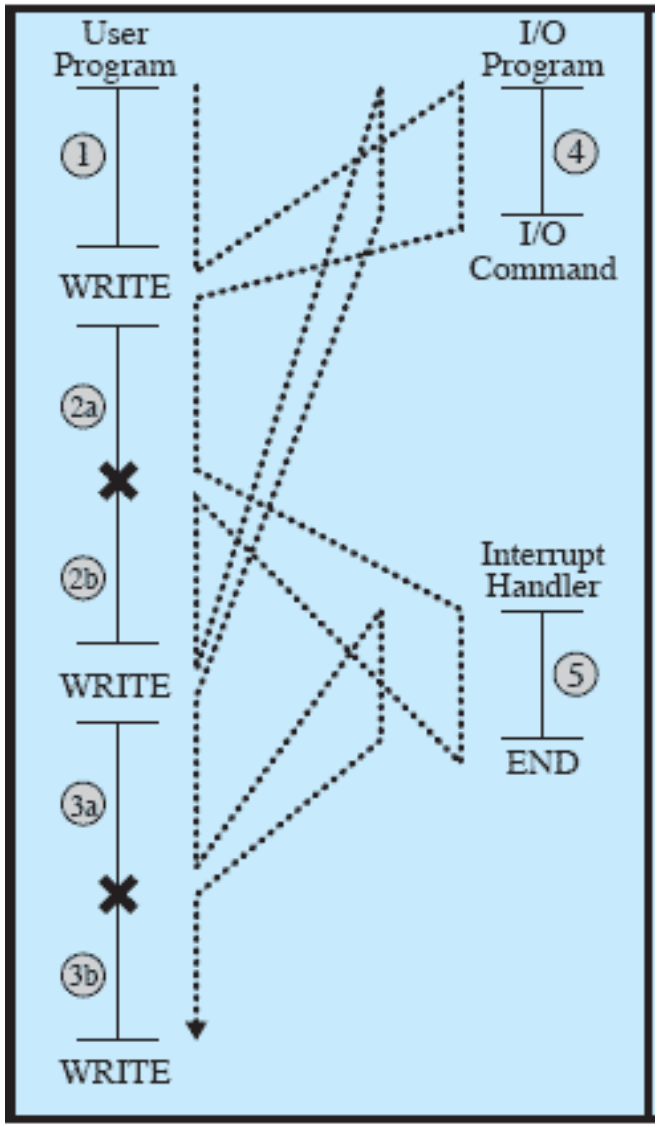



(a) No interrupts

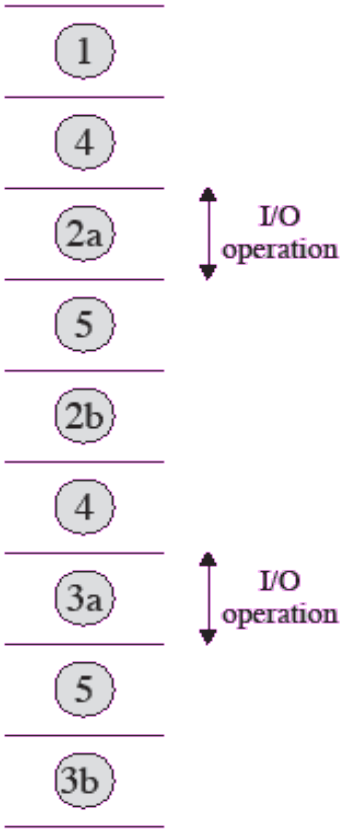


# Flow of Control with Interrupts and I/O Wait

Figure 2




(b) Interrupts; short I/O wait



# Transfer of Control via Interrupts

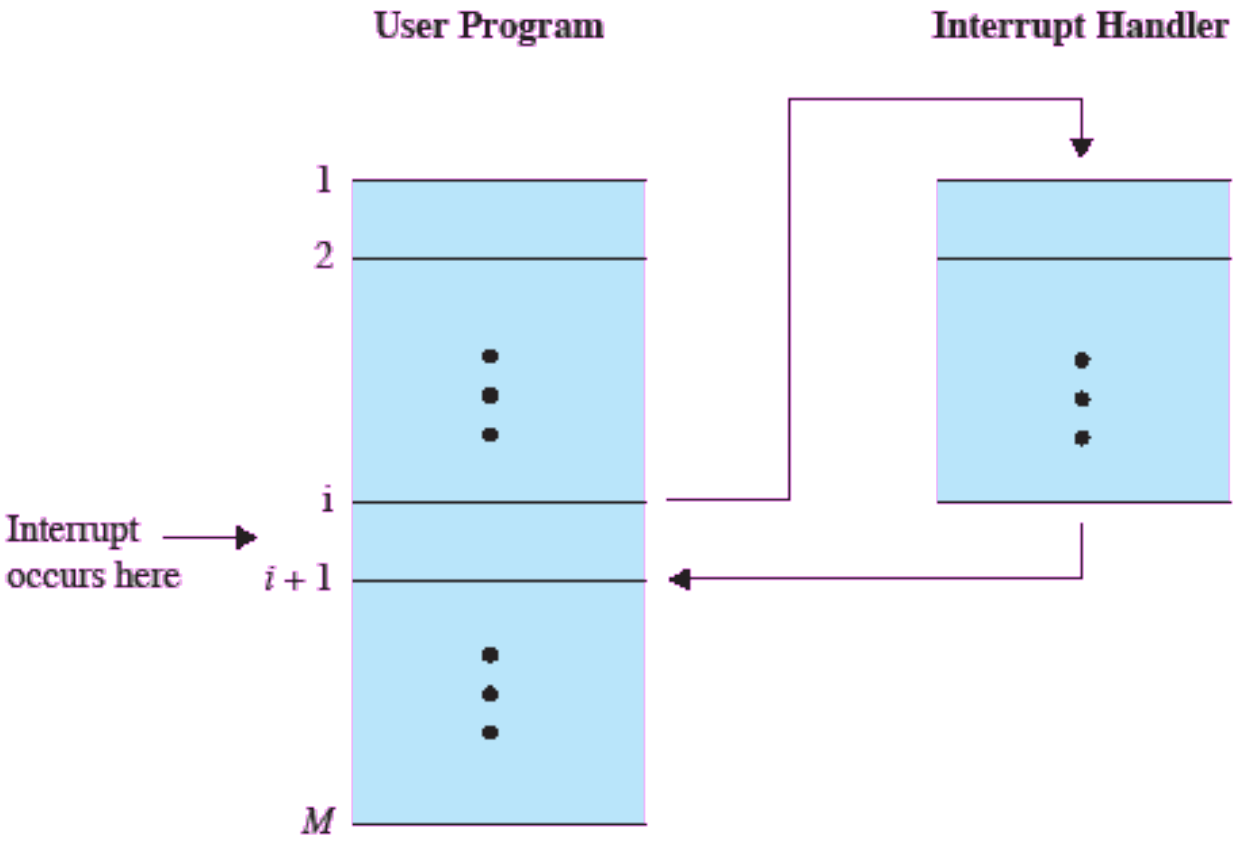



**Figure 1.6 Transfer of Control via Interrupts**



52

# Instruction Cycle with Interrupts

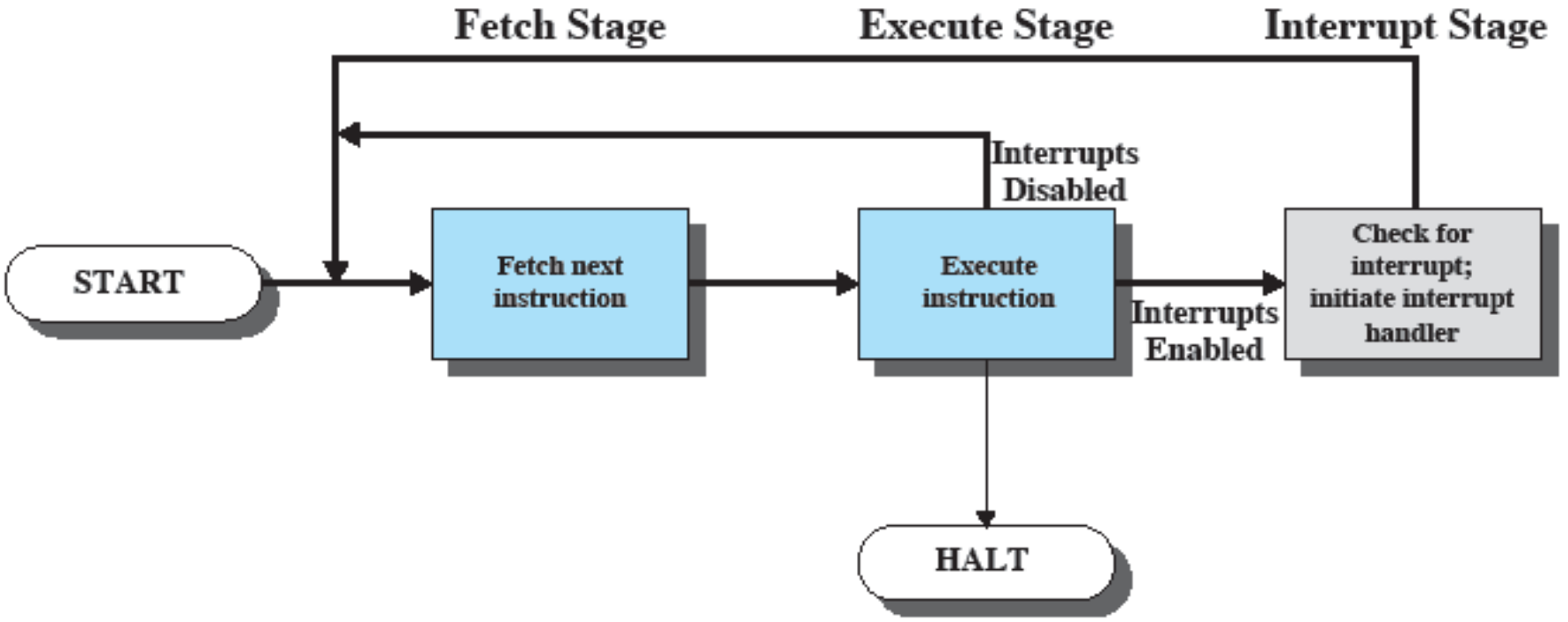



**Figure 1.7 Instruction Cycle with Interrupts**



53