Operating System Overview

Dr. Ailbhe Gill

Textbook: Operating Systems: Internals and Design Principles, 9th Edition by William Stallings

Today's Lecture

- Overview of operating systems
- What operating systems do
- Evolution of Operating Systems
- Major Acheivements/ Developments

Learning Objectives

After studying this chapter, you should be able to:

- Summarize, at a top level, the key functions of an operating system (OS).
- Discuss the evolution of operating systems for early simple batch systems to modern complex systems.
- Give a brief explanation of each of the major achievements in OS research.

Operating System Objectives and Functions

Operating System

Operating System

- A program that controls the execution of application programs
- An interface between applications and hardware

Some Services the Operating System Provides

- Access and management of I/O devices including secondary memory
- Controlled access to files
- Controlled system access
- Management of program execution
- Error detection and response
- Accounting i.e. monitors resource usage and performance
- Program development facilities e.g built-in editors/debuggers

Computer Hardware and Software Structure

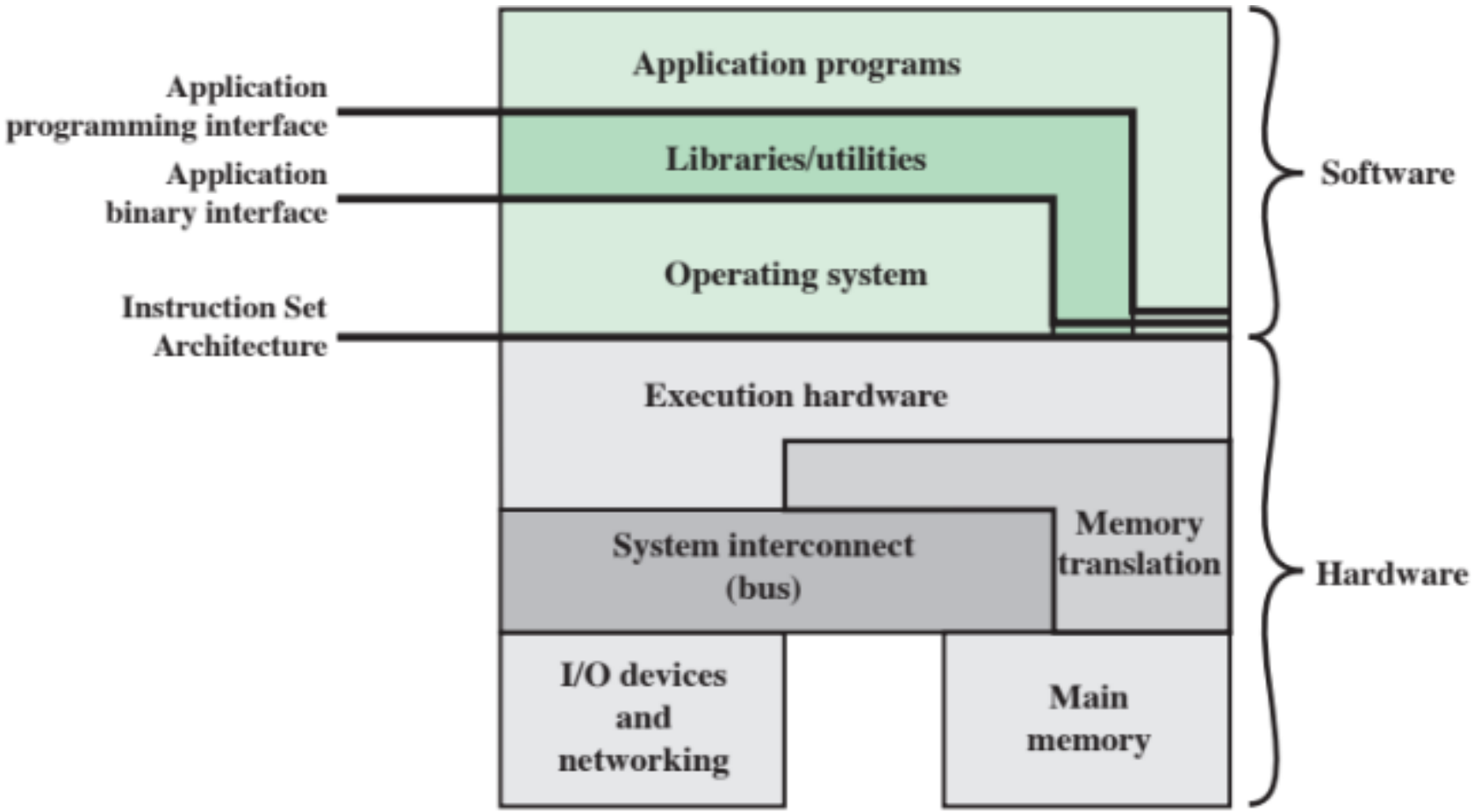

Figure 1: Computer Hardware and Software Structure

Key Interfaces (1 of 2)

Instruction set architecture (ISA)

- Defines the repertoire of machine language instructions that a computer can follow.
- Is the boundary between hardware and software.

Both application programs and utilities may access the ISA directly. For these programs, a subset of the instruction repertoire called the **user ISA** is available.

The OS has access to additional machine language instructions that deal with managing system resources via the **system ISA**.

Key Interfaces (2 of 2)

Application binary interface (ABI)

- Defines a standard for binary portability across programs.
- Defines the system call interface to the operating System
- Defines the hardware resources and services available in the system through the user ISA.

Application programming interface (API)

- Gives a program access to the hardware resources and services available in a system
- Using an API enables application software to be ported easily, through recompilation, to other systems that support the same API.

The Role of an OS

The operating system as **resource manager**

- A computer is a set of resources for the movement, storage, and processing of data
- The OS is responsible for managing these resources such as I/O, main and secondary memory, and processor execution time

Operating System as Resource Manager

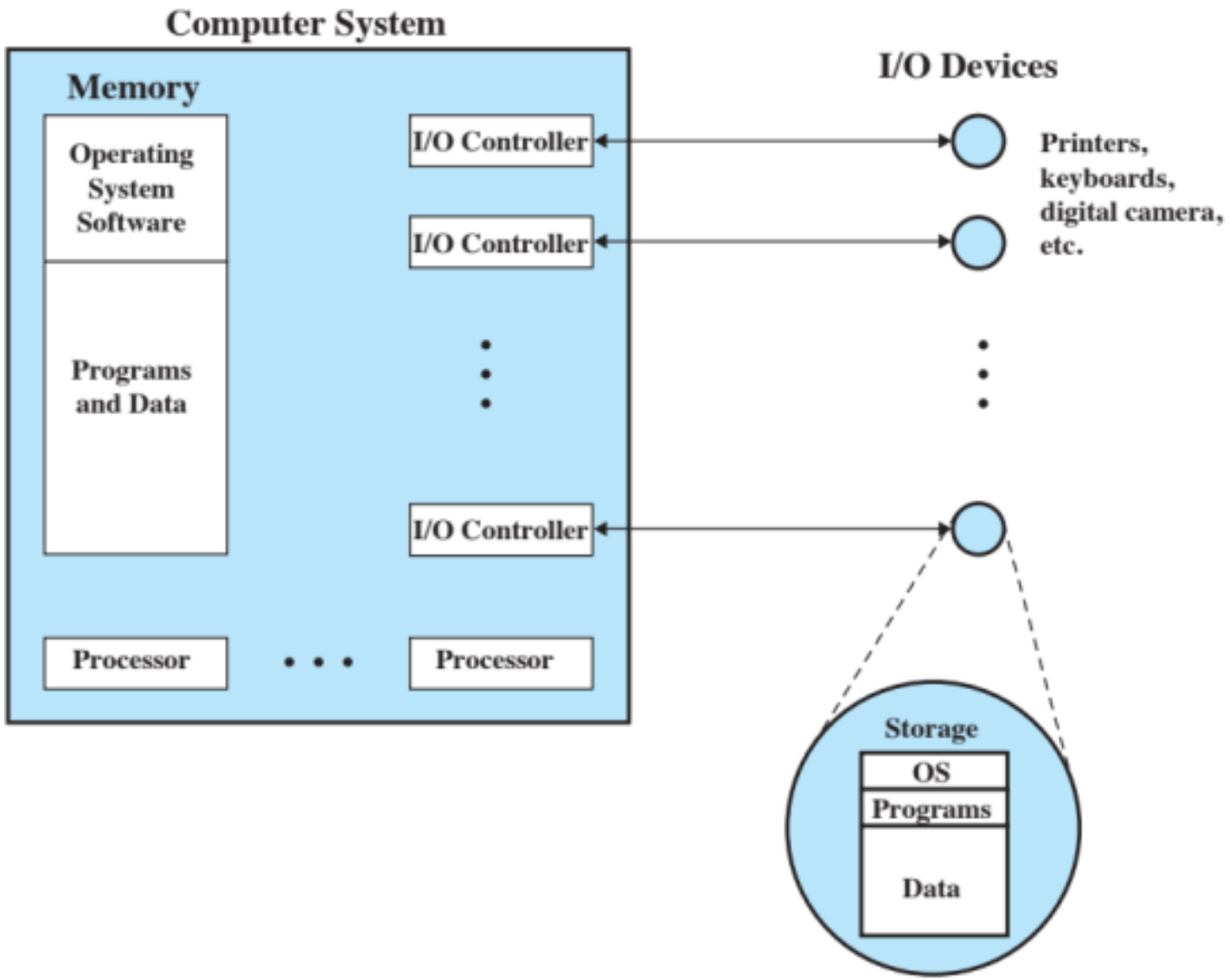

Figure 2: The Operating System as Resource Manager

The Role of an OS

The operating system as **software**:

- Functions in the same way as ordinary computer software
- Program, or suite of programs, executed by the processor
- Frequently relinquishes control and must depend on the processor to allow it to regain control

Evolution of Operating Systems

Evolution of Operating Systems

A major OS will evolve over time for a number of reasons:

- Hardware upgrades
- New types of hardware
- New Services
- Fixes

Stages include:

1. Serial Processing
2. Simple Batch Systems
3. Multiprogrammed Batch Systems
4. Time Sharing Systems

Serial Processing

Earliest Computers (late 1940s to mid-1950s)

- No operating system; programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in series (serial processing)

Problems:

1. Scheduling:
   - Most installations used a hardcopy sign-up sheet to reserve computer time
   - Time allocations could run short or long, resulting in wasted computer time
2. Setup time:
   - A considerable amount of time was spent just on setting up the program to run

Simple Batch Systems

Motivation for simple batch systems:

- Early computers were very expensive
- Important to maximize processor utilization

Introduction of the **monitor**

- User no longer has direct access to processor
- Job is submitted to computer operator who batches them together and places them on an input device
- Program branches back to the monitor when finished

Monitor Point of View

- Monitor controls the sequence of events
- Resident Monitor is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

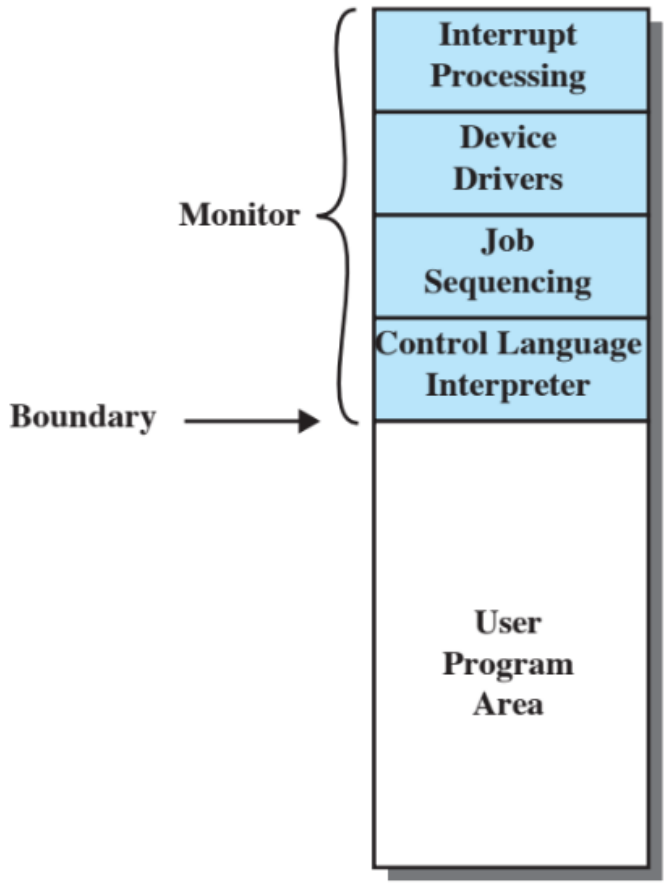

Figure 3: Main Memory Layout for a Resident Monitor

Processor Point of View

- Processor executes instruction from the memory containing the monitor
- Executes the instructions in the user program until it encounters an ending or error condition
- "*Control is passed to a job*" means that the processor is fetching and executing instructions in a user program
- "*Control is returned to the monitor*" means that the processor is fetching and executing instructions from the monitor program

Job Control Language (JCL)

A special type of programming language used to provide instructions to the monitor

Instructions such as:

- What compiler to use
- What data to use

On the right is an example of a user submitting a program written in the programming language FORTRAN plus some data to be used by the program. In addition to FORTRAN and data lines, the job includes job control instructions, which are denoted by the beginning $

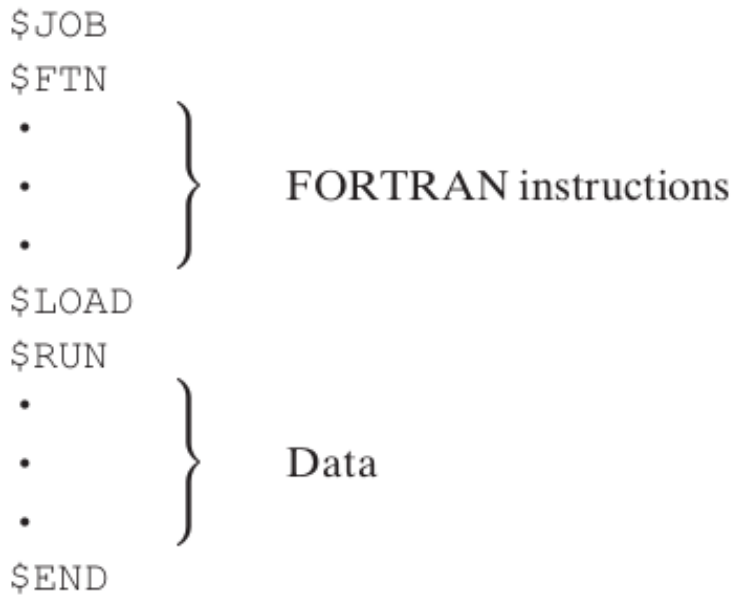

Figure 4: Example format of a job to be read in by a monitor

Desirable Hardware Features

To improve upon early systems other features could be considered.

Desirable Hardware Features:

- **Memory protection**

  While the user program is executing, it must not alter the memory area containing the monitor

- **Timer**

  Prevents a job from monopolizing the system

- **Privileged Instructions**

  Can only be executed by the monitor

- **Interrupts**

  Gives OS more flexibility in controlling flow of user programs

Modes of Operation

Considerations of memory protection and privileged instructions led to the concept of modes of operation.

User Mode

- User program executes in user mode
- Certain areas of memory are protected from user access
- Certain instructions may not be executed

Kernel Mode

- Monitor executes in kernel mode
- Privileged instructions may be executed
- Protected areas of memory may be accessed

Simple Batch System Overhead

- With a batch OS, processor time alternates between execution of user programs and execution of the monitor
- Sacrifices (leading to overhead):
  - Some main memory is now given over to the monitor
  - Some processor time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer

Simple Batch Systems Issue

These were uniprogramming systems (only one processor), which caused the following issue:

- Processor is often idle
- Even with automatic job sequencing
- Major bottleneck: I/O devices are slow compared to processor

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
Total	31 μs

$$\text{Percent CPU utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 5: System Utilization Example

Uniprogramming

- The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

Figure 6: Uniprogramming

Multiprogramming

Also known as multitasking. The act of performing more than one task at the same time. Memory is expanded to hold two or more programs and switch among all of them.

- There must be enough memory to hold the OS (resident monitor) and one user program
- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O

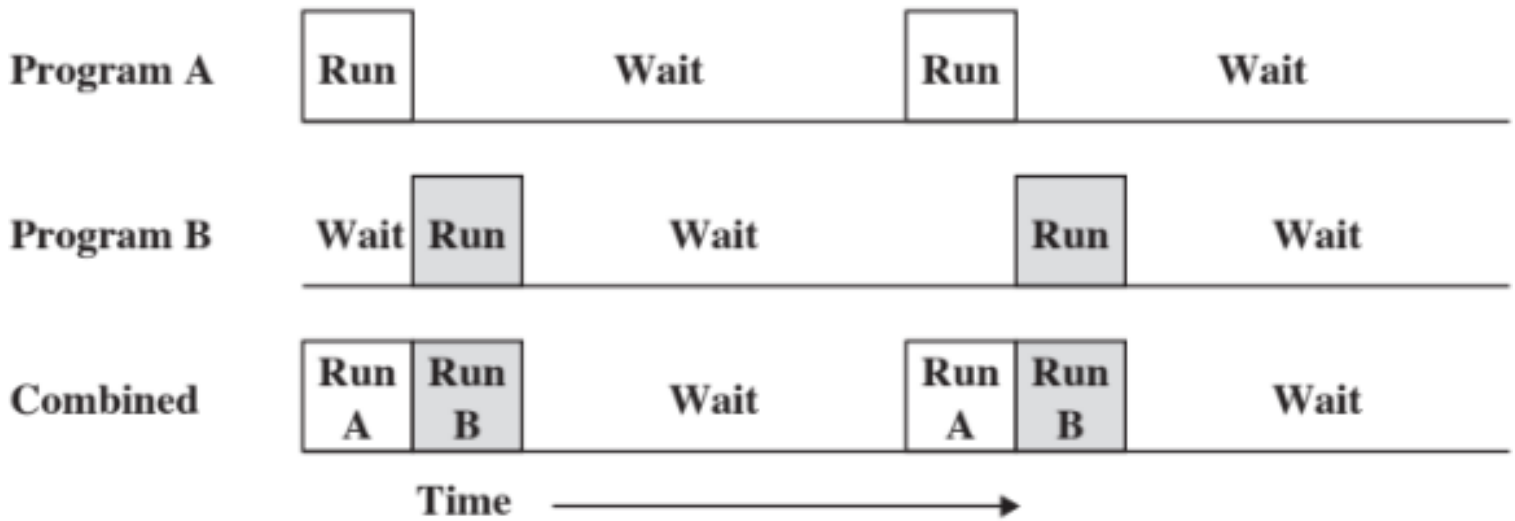

Figure 7: Multiprogramming with two programs

Multiprogramming Example

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Figure 8: Sample Program Execution Attributes

Utilization Histograms

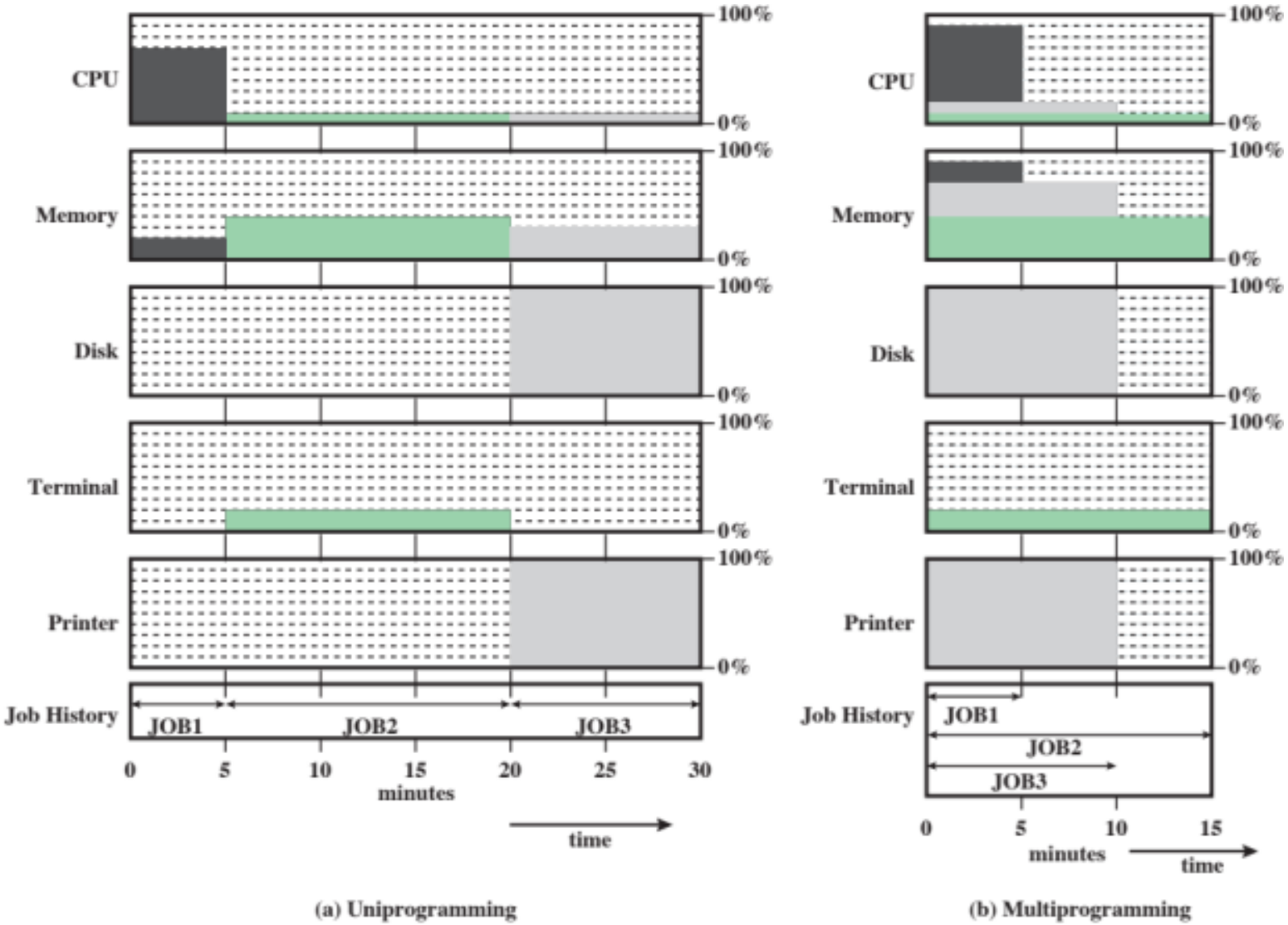

Figure 9: Utilization Histograms

Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Figure 10: Effects of Multiprogramming on Resource Utilization

Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

Batch Multiprogramming versus Time Sharing

Both batch processing and time sharing use multiprogramming. The key differences are listed below:

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Figure 11: Batch Multiprogramming versus Time Sharing

Compatible Time-Sharing Systems (CTSS)

- One of the first time-sharing operating systems
- Developed at MIT by a group known as Project MAC
- The system was first developed for the IBM 709 in 1961
- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that
- When control was to be assigned to an interactive user, the user's program and data were loaded into the remaining 27,000 words of main memory.
- A program was always loaded to start at the location of the 5,000th word; this simplified both the monitor and memory management.
- Utilized a technique known as *time slicing*

Compatible Time-Sharing Systems (CTSS)

Time Slicing in Compatible Time-Sharing Systems

- System clock generates interrupts at a rate of approximately one every 0.2 seconds
- At each clock interrupt the OS regained control and could assign processor to another user
- Thus, at regular time intervals the current user would be preempted and another user loaded in
- To preserve the old user status for later resumption, the old user programs and data were written out to disk before the new user programs and data were read in
- Old user program code and data were restored in main memory when that program was next given a turn
- To minimize disk traffic, user memory was only written out when the incoming program would overwrite it.

CTSS Operation

Assume there are four interactive users with the following memory requirements, in words:
JOB1 (15,000), JOB2 (20,000), JOB3 (5,000), JOB4 (10,000).

Initially, the monitor loads JOB1 and transfers control to it (a).
Later,the monitor decides to transfer control to JOB2. Because JOB2 requires more memory than JOB1, JOB1 must be written out first, and then JOB2 can be loaded (b).
Next, JOB3 is loaded in to be run. However, because JOB3 is smaller than JOB2, a portion of JOB2 can remain in memory, reducing disk write time (c).
Later, the monitor decides to transfer control back to JOB1. An additional portion of JOB2 must be written out when JOB1 is loaded back into memory (d).
When JOB4 is loaded, part of JOB1 and the portion of JOB2 remaining in memory are retained (e).
At this point, if either JOB1 or JOB2 is activated, only a partial load will be required. In this example, it is JOB2 that runs next. This requires that JOB4 and the remaining resident portion of JOB1 be written out, and the missing portion of JOB2 be read in (f).

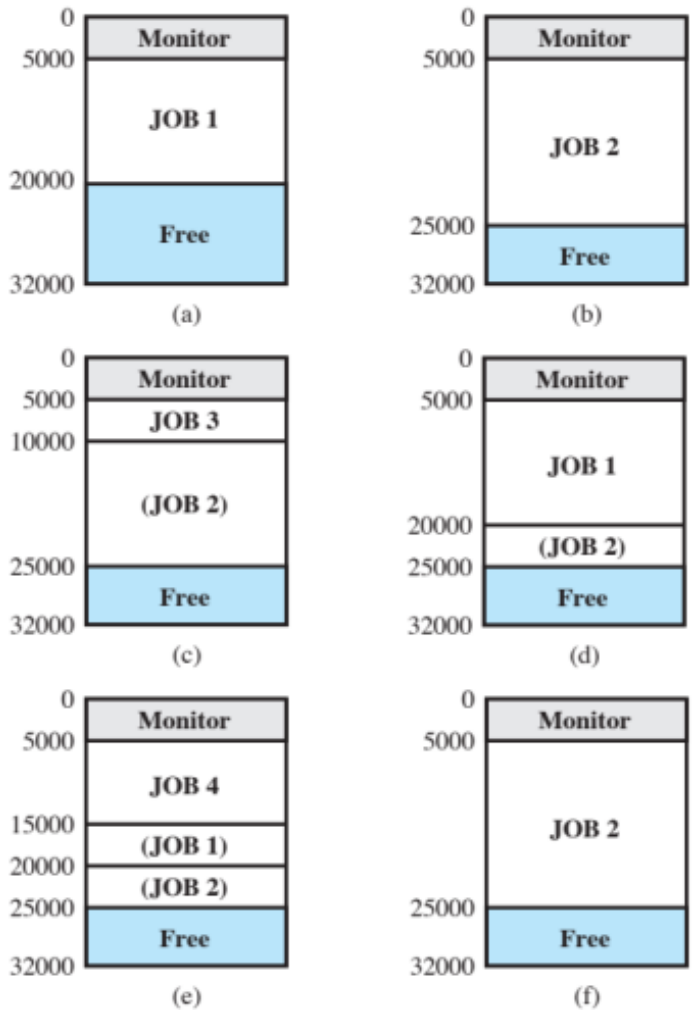

Figure 12: CTSS Operation

Major Achievements

Major Achievements

Operating Systems are among the most complex pieces of software ever developed

Major advances in development include:

- Processes
- Memory management
- Information protection and security
- Scheduling and resource management

Process

Fundamental to the structure of operating systems

A process can be defined as:

- A program in execution
- An instance of a running program
- The entity that can be assigned to, and executed on, a processor
- A unit of activity characterized by a sequential thread(s) of execution, a current state, and an associated set of system resources

where a program is an executable file (contains a set of instructions) on your computer hard drive.

Components of a Process

A process contains three components:

- An executable program
- The associated data needed by the program (variables, work space, buffers, etc.)
- The execution context (or "process state") of the program

The execution context is essential:

- It is the internal data by which the OS is able to supervise and control the process
- Includes the contents of the various process registers
- Includes information such as the priority of the process and whether the process is waiting for the completion of a particular I/O event

Process Management

- The entire state of the process at any instant is contained in its context
- New features can be designed and incorporated into the OS by expanding the context to include any new information needed to support the feature

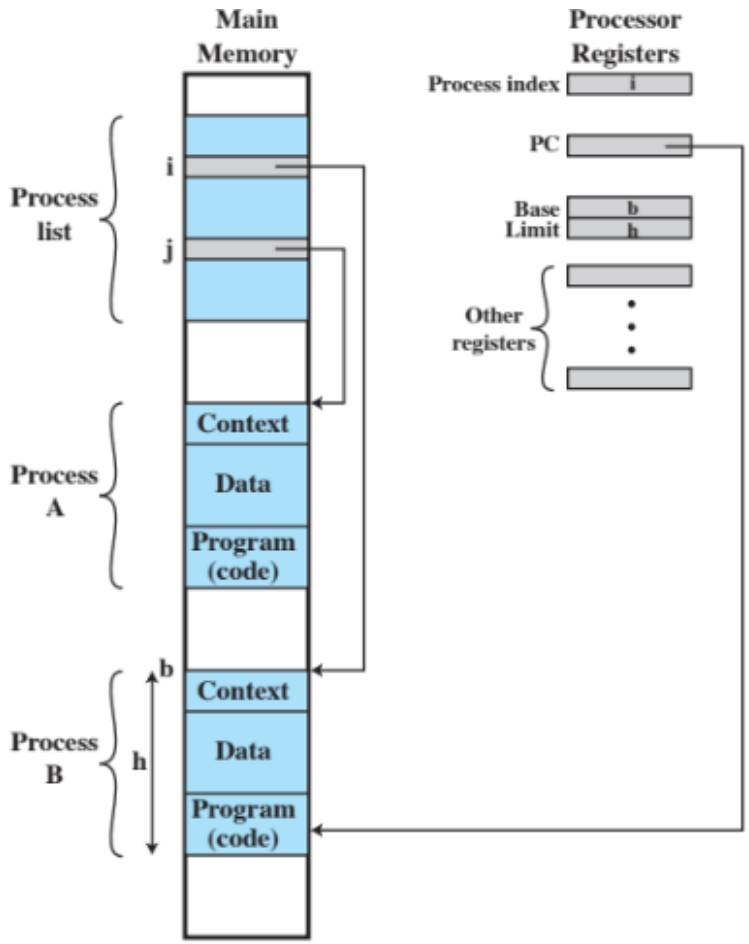

Figure 13: Typical Process Implementation

Memory Management

The OS has five principal storage management responsibilities:

1. Process isolation
2. Automatic allocation and management
3. Support of modular programming
4. Protection and access control
5. Long-term storage

Information Protection and Security

- The nature of the threat that concerns an organization will vary greatly depending on the circumstances
- The problem involves controlling access to computer systems and the information stored in them

Main issues:

- Availability
- Confidentiality
- Data integrity
- Authenticity

Scheduling and Resource Management

Key responsibility of an OS is managing resources

Resource allocation policies must consider:

- Efficiency
- Fairness
- Differential Responsiveness
  - i.e. OS may need to discriminate among different classes of jobs with different service requirements.

Key Elements of an Operating System for Multiprogramming

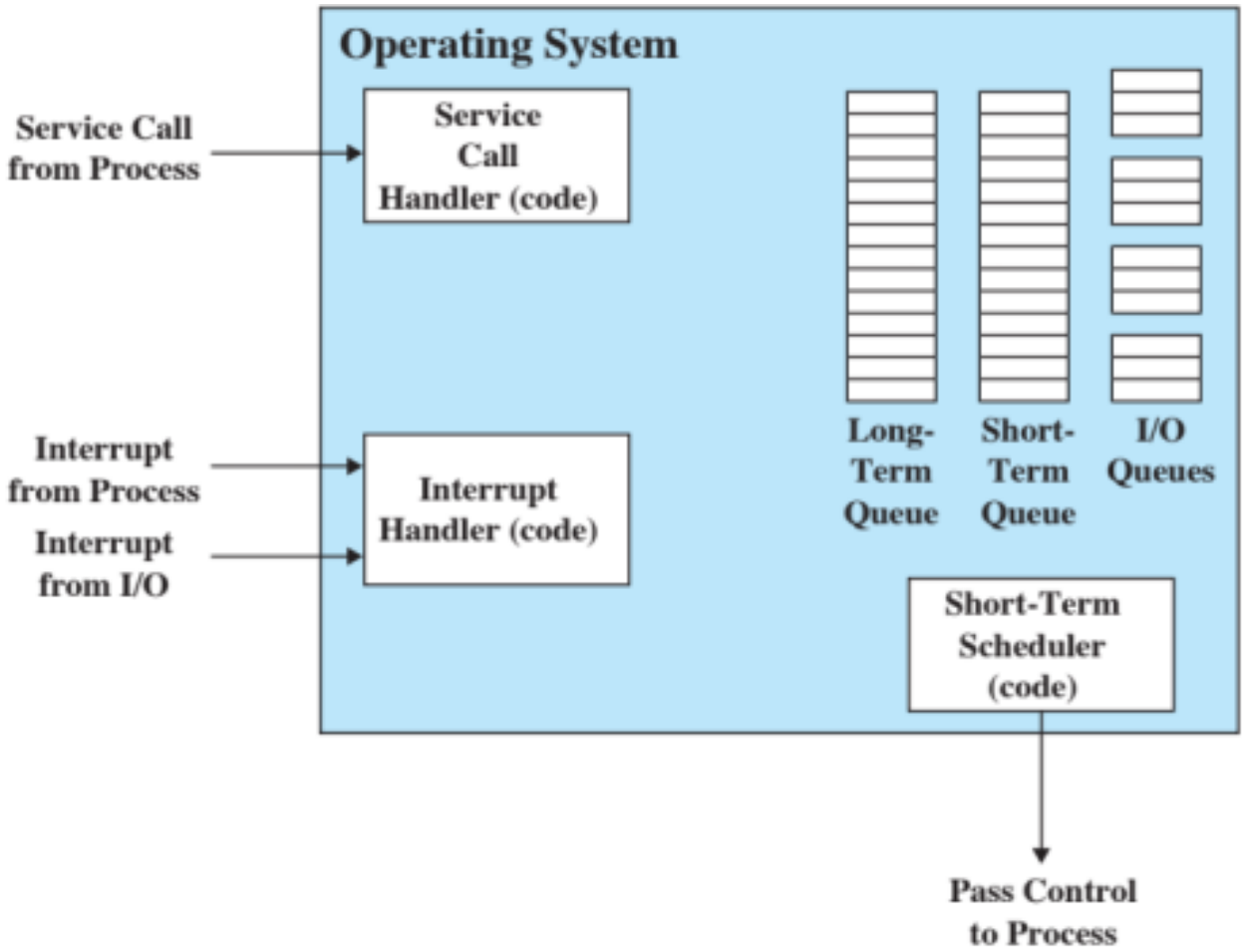

Figure 14: Key Elements of an Operating System for Multiprogramming

Multiprogramming and Multiprocessing

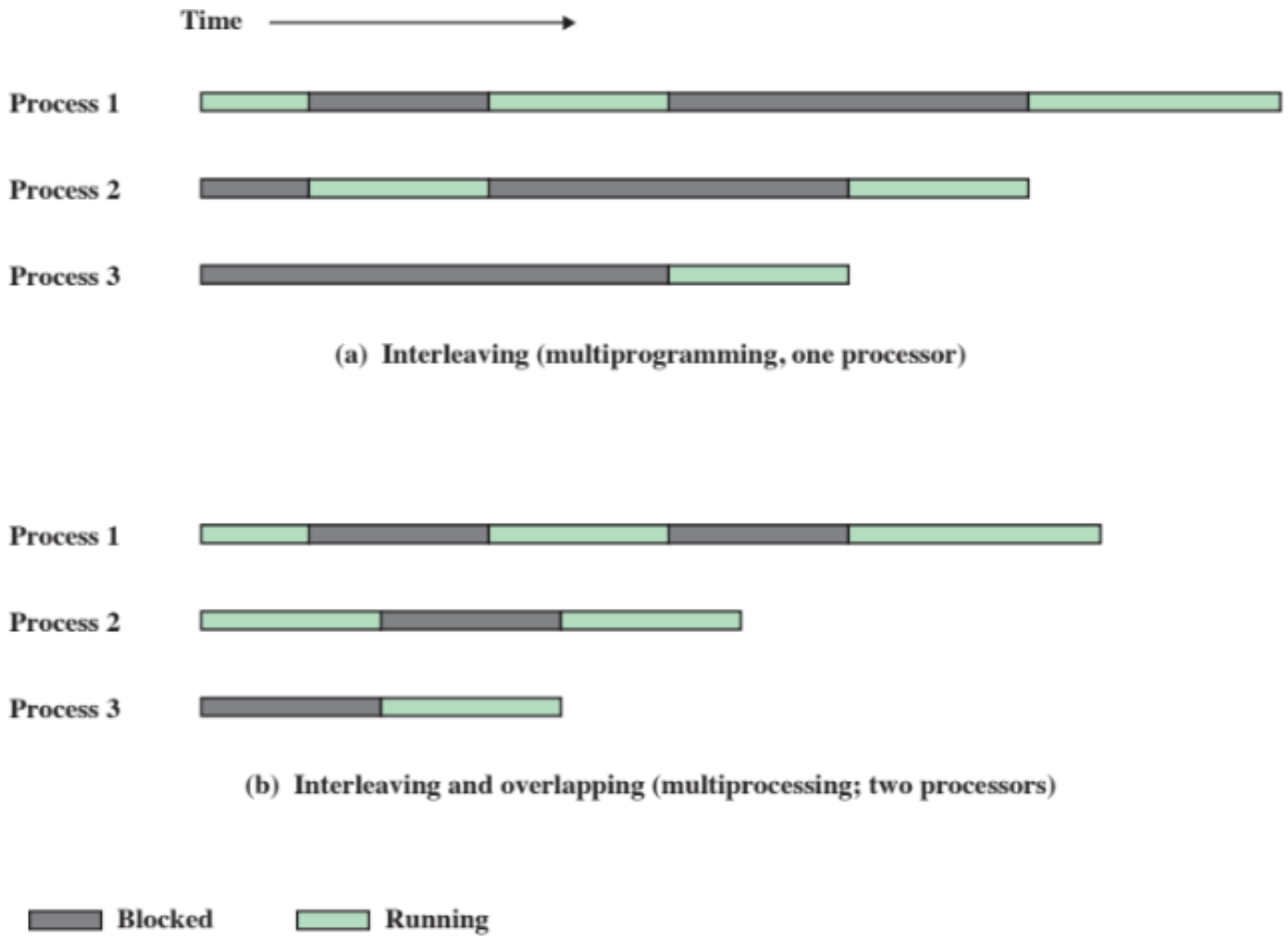

Figure 15: Multiprogramming and Multiprocessing

Multiprogramming and Multiprocessing

Multiprogramming: Involves interleaving of processes

- e.g. while one process is waiting on I/O operation to complete another process can be running on the CPU.

Multiprocessing: Involves overlapping (as well as interleaving) of processes.

- e.g. if there are two processors then two processes can run at the same time

Summary

- Operating system objectives and functions
- Evolution of operating systems
  - Serial processing
  - Simple/multiprogrammed batch systems
  - Time-sharing systems
- Major achievements