

Textbook:

**Operating Systems – Internals and Design Principles,**  
by ***William Stallings***.

Publisher: Pearson.

# Chapter 2

# Operating System

# Overview

**Part A**



## Important recapitulation

At this point it is presumed that you have basic knowledge from course **COS151: Introduction to Computer Science**.



**Before you continue with this lecture,  
Please press the PAUSE button, and  
familiarize yourself once more with  
Chapter #5 of COS151:  
COMPUTER ORGANISATION**



Thereafter you may press the RESUME  
Button in order to continue with this lecture.

Behrouz  
Forouzan:

Foundations  
of Computer  
Science

CENGAGE  
Publisher

# Contents of Chapter 2: Overview

- Operating system objectives and functions
  - User/computer interface
  - Resource manager
- Evolution of operating systems
  - Serial processing
  - Simple/multiprogrammed/time-sharing batch systems
- Major achievements
- Developments leading to modern operating systems
- Fault tolerance
  - Fundamental concepts
  - Faults
  - OS mechanisms
- OS design considerations for multiprocessor and multi-core
- Microsoft Windows overview
- Traditional Unix systems
  - History/description
- Modern Unix systems
  - System V Release 4 (SVR4)
  - BSD
  - Solaris 10
- Linux
  - History
  - Modular structure
  - Kernel components
- Android
  - Software/system architecture
  - Activities
  - Power management





# Resources and their Management

- In the previous **Chapter** you had seen an overview of the computer's **hardware** and its various **components**.
- From an abstract perspective, **all** those components can be viewed as “**resources**”.
- The main task of the operating system is the **frugal and efficient allocation and management** of the available resources for (and to) the “users” (humans or programs) of the computer.





# Textbook Section 2.1

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

## Main objectives of an OS:

- Convenience
- Efficiency
- Ability to evolve

# The Operating System as “Shield”



User

Operating System

Hard  
ware

The user is shielded from the confusing technical details of the hardware. Instead of seeing the hardware, the user sees the OS.

The operating system provides well-defined functions for accessing the hardware in a well-defined manner.

The hardware's resources are thus shielded from inappropriate access or un-desired utilisation by incompetent or malicious users.



See Figure 2.1 in the book for further details and explanations







# Operating System Support Services for:

- Program development (editors and debuggers)
- Program execution
- Access to I/O devices
- Controlled access to Files
- Concurrent System Access (by multiple users)
- Error detection and response
- Accounting (resource utilization statistics)

Study section 2.1 in the book for further details and explanations



# Operating System as Resource Manager

- Because the **Operating System** itself is software, too, it competes with (or against) the user's application software for the same hardware resources (CPU etc.), and **must thus also manage itself!**
  - **The OS** must thus frequently “relinquish control” (to the user's software) and **must rely on special hardware mechanisms to regain control again.** (While the user's program utilizes the CPU, the OS cannot “do” anything, because “doing something” implies CPU utilization)
- 



# A noteworthy mis-understanding



Some years ago a student wrote in the exam about the most essential services provided by the Operating System:

*“When the CPU is failing, then the Operating Systems takes control and replaces the CPU”*

That student had not understood that **the operating system is Software** which also needs the CPU as a hardware resource!



See Figure 2.2 in the book for further details and explanations

# Evolution of Operating Systems:

## Textbook Section 2.2

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

Hardware upgrades

New types of hardware

New services

Fixes

}  
External  
Reasons

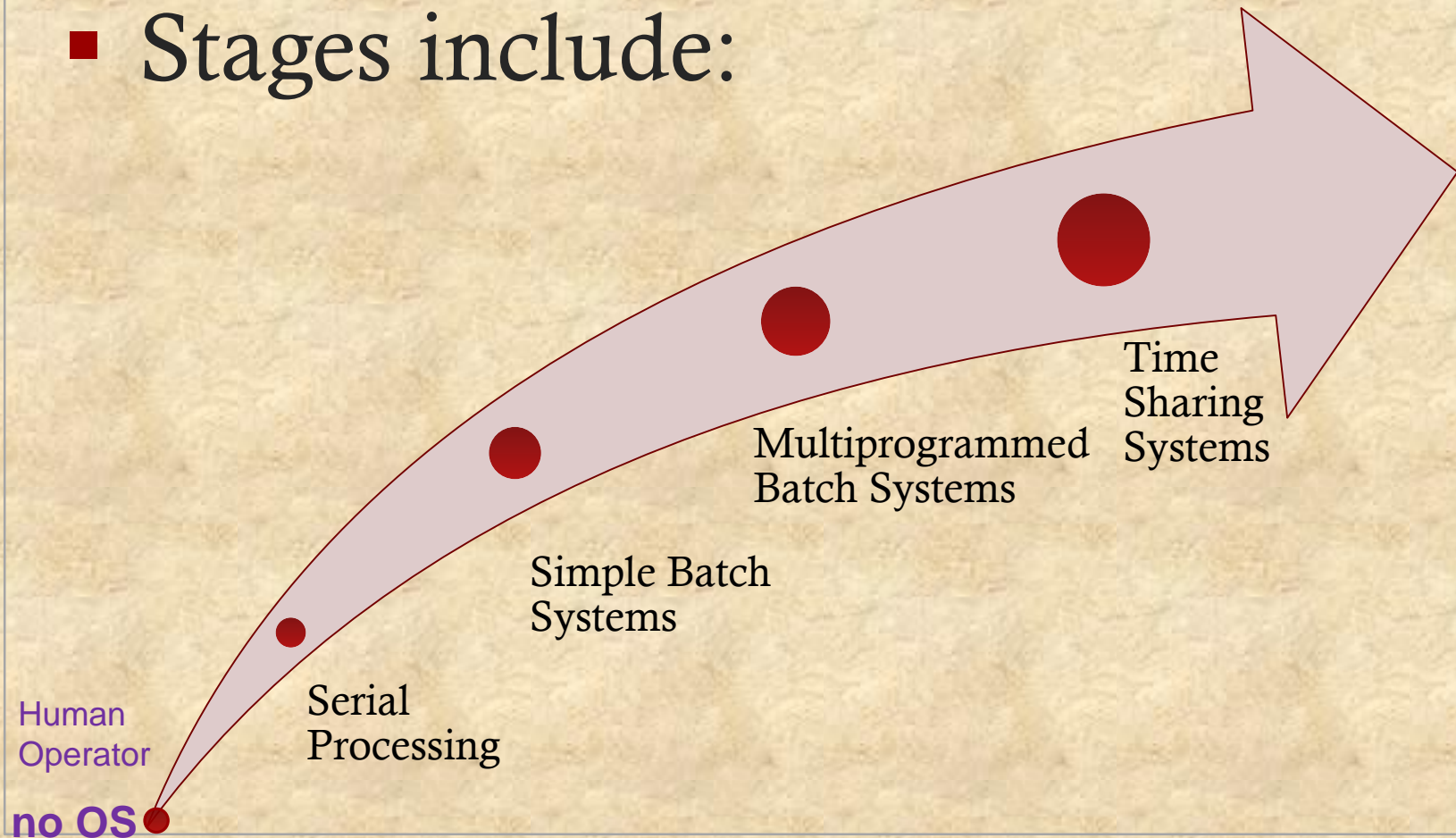
}  
Internal  
Reasons



# Main “path” of Evolution of Operating Systems



- Stages include:





# Interesting Anecdote



When **Bill Gates** programmed his first version of the “Disc Operating System” (**DOS**), he was a schoolboy of **age 13**. The DOS code fitted onto a floppy-disk of less than 2MB storage space.

Nowadays operating systems have **millions of lines of software code** which can no longer be programmed by one person alone.

The need for developing so large operating systems  
By many teams of software developers in an orderly  
and systematic manner gave rise to the discipline of  
**Software Engineering**

# Serial Processing / Simple Batch Systems



- Early Operating Systems *mainly “mimicked” the work-behavior of the human operator.*
- The underlying principle was that a **user-program U had to run entirely to completion** before a subsequent user program U' was allowed to start.
- The “**Monitor**” component had to recognize the completion of a user program, and then to carry out the necessary administrative tasks (code-loading, etc.) to prepare the run of the subsequent user-program, and so on, until the whole “batch” of user-programs was carried out.



# Monitor's 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

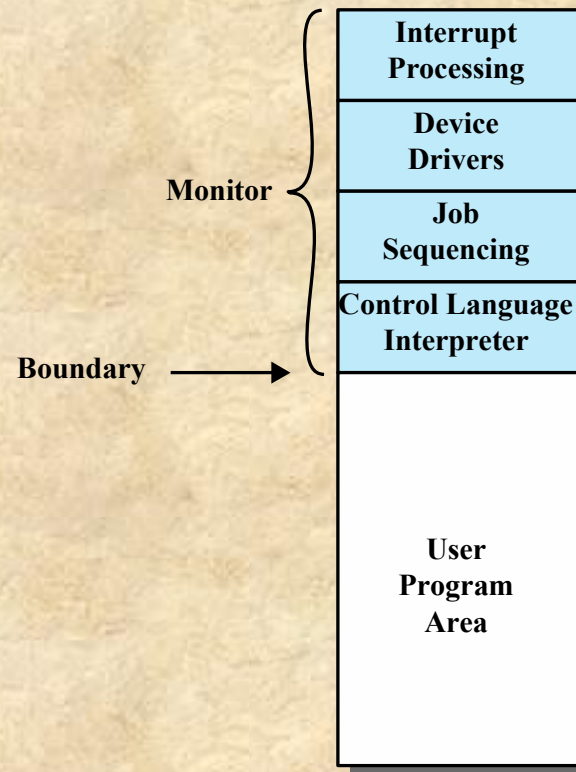


Figure 2.3 Memory Layout for a Resident Monitor





# CPU's Point of View

- The CPU does NOT know whether it processes User Software or Operating System's software (Monitor)!
- CPU executes instruction from the memory containing the monitor  
→ OS is running
- CPU executes the instructions in the user program (until it encounters an ending or error condition) → user software is running
- “*Control is passed to a job*” means: CPU is fetching and executing instructions in a user program
- “*Control is returned to the monitor*” means: CPU is fetching and executing instructions from the monitor program

# 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

**See Textbook, Section 2.2, for the discussion of several further details.**  
For “Instruction Set”, please also recall **Table 5.4** in the book of **COS151**.

# Simple Batch System: Discussion

- Processor time alternates between execution of user programs (P1, P2, P3, ... Pn) and execution of the monitor (M):



M | P1 | M | P2 | M | P3 | M | ... ... | M | Pn | M | *// end of batch*

TIME

- **Sacrifices (Management Costs):**
  - Some main memory is now given over to the monitor
  - Some CPU time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer in comparison against the slow “manual” processing of jobs by a human operator.



# Simple Batch System: Discussion

- However, CPU utilization (efficiency) becomes very poor when the a user-program makes frequent I/O operations, because I/O operations are very slow:

Read one record from file	15 $\mu$ s
Execute 100 instructions	1 $\mu$ s
Write one record to file	<u>15 <math>\mu</math>s</u>
TOTAL	31 $\mu$ s

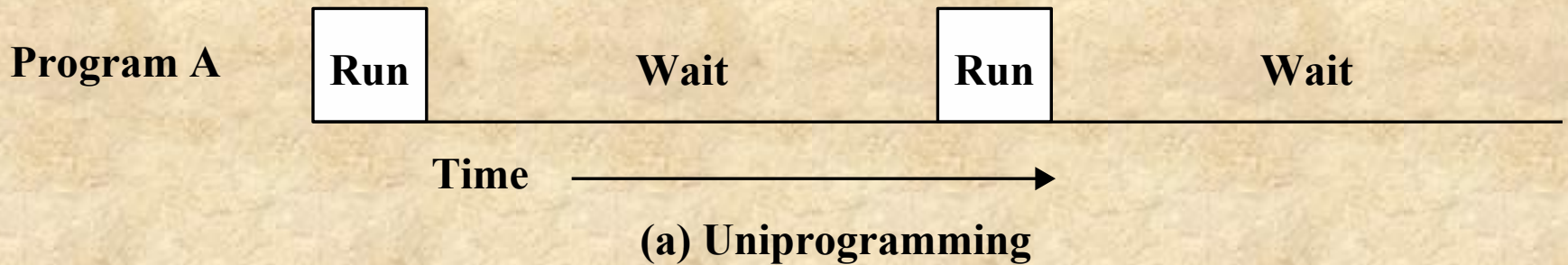
$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$



**Figure 2.4 System Utilization Example**

# Uniprogramming

Figure 2.5 a)

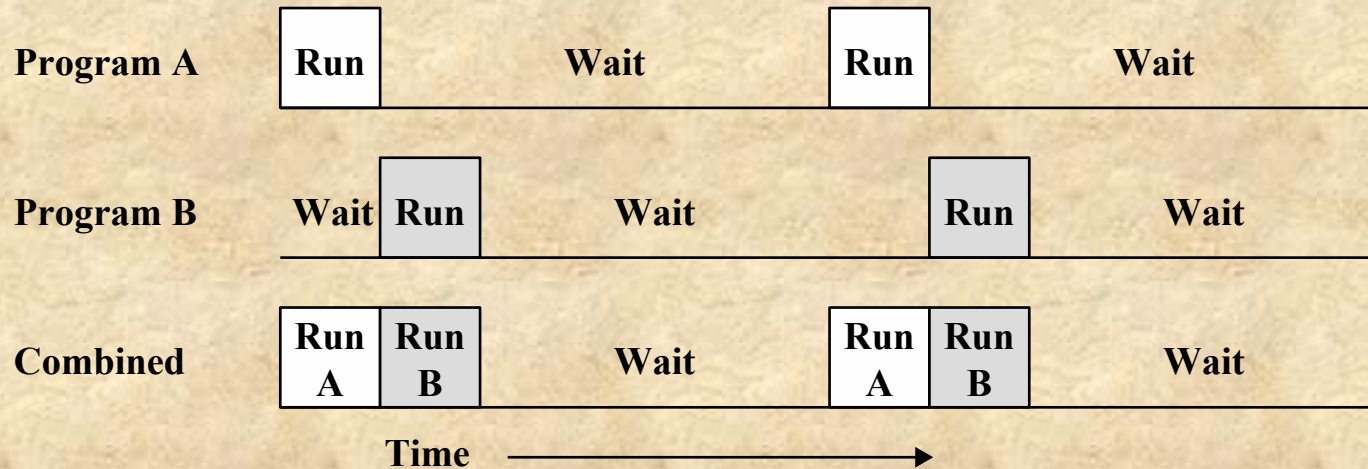


The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding





# Multiprogramming





(b) Multiprogramming with two programs

- There must be enough memory space to hold the OS (resident monitor) and more than one user program (A, B, C...)
- 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





# Time-Sharing Systems

- Whereas “early” user application programs did mostly mathematical calculations to emitted a final result, “modern” user applications frequently communicate or “interact” with their environment: [See, for example, a computer game which users play.](#)
  - Time-Sharing Systems are Operating Systems which combine the idea of Multi-Programming (see above) with additional features that are needed to support especially “interactive” programs.
  - They can be used to **handle multiple interactive jobs**, whereby CPU time is shared among multiple users
  - Multiple users simultaneously access the system through terminals, whereby the OS “interleaves” the execution of all user programs in frequent short “bursts” of computation-time.
- 
- 




	<b>Batch Multiprogramming</b>	<b>Time Sharing</b>
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

**Table 2.3 Batch Multiprogramming versus Time Sharing**



# Compatible Time-Sharing System (CTSS)

- **History:** CTSS was one of the first time-sharing operating systems (developed for the IBM709 in 1961)
- Utilized a technique known as *time slicing*
  - **System clock generated interrupts** at a rate of approximately one every 0.2 seconds
  - At each clock interrupt the OS regained control and could assign another user CPU
  - Thus, at regular time intervals the current user would be preempted and another user loaded in
  - To preserve the old user program 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



**The details of these management activities will be discussed later in Chapters 3 (Processes), 7 (Memory Management), and 9 (Scheduling)**



## Example: Memory Status at different Time-Points during the Run of CTSS

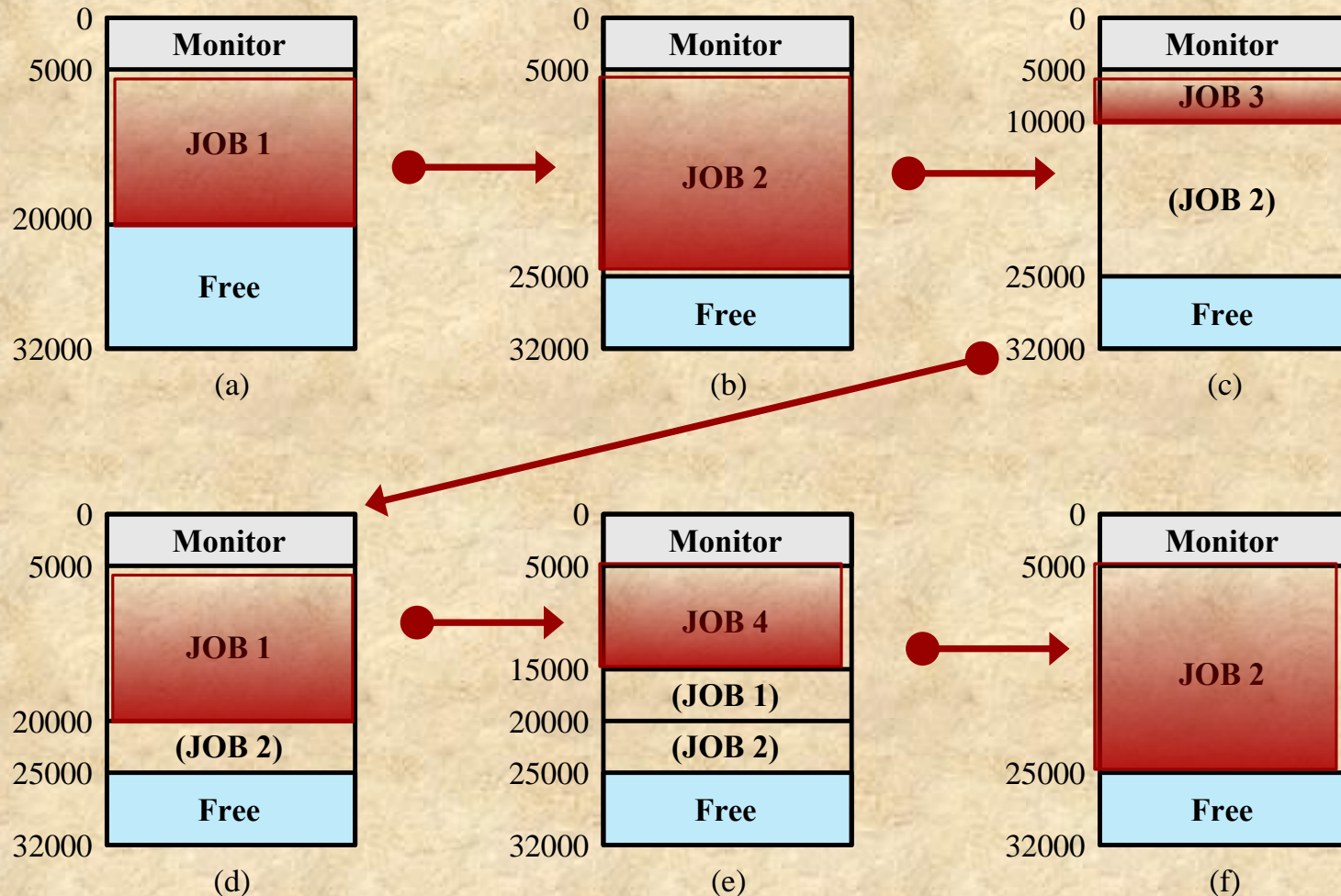


Figure 2.7

The red areas show the memory sections utilized by the various user-jobs. In this example, the user-jobs are serviced in the following **sequence**: J1, J2, J3, J1 (continuation), J4, J2 (continuation)

The Operating-System **overwrites** the memory-section with the **data of the active job**, because memory was a scarce resource in the early 1960.



# Major Achievements

- Operating Systems are among the most complex pieces of software ever developed
- **Major advances in the history of OS development** entail:
  - Multiple User-Processes
  - Memory Management
  - Information Protection and Security
  - Job Scheduling and resource management
  - System Architecture (Software Engineering)

**(End of Part A)**