UNIT 1

Operating Systems: Structure and Services

Student Learning Outcomes:

By the end of this chapter, you will be able to:

- Define an operating system and explain its role as an interface between the user and translate user actions into hardware-level instructions.
- Identify the responsibilities of the OS in managing multiple user accounts.
- Differentiate between the kernel and shell, including their specific functions.
- Explain OS layers and how each layer interacts with the others.
- Identify the role of system libraries and device drivers in OS functionality.
- Explain the stages in the process lifecycle (creation, execution, termination) with examples and apply the First-Come, First-Served (FCFS) process scheduling.
- Describe multitasking and concurrency, giving one real-world analogy.
- Distinguish between primary memory (RAM) and virtual memory.
- Define a process and a thread, and explain how threads share resources within a process.
- Give examples of how multithreading improves performance in applications.
- Define a system call and explain its purpose and types
- Define file system, files, folders, and metadata, and explain their roles in organizing data.
- Explain how these components work together to process a user request from device to server and back.

Introduction

In this chapter, we will discuss the core functions and structure of an operating system, the fundamental software that manages computer hardware and facilitates seamless user interaction. The discussion will covers:

- Operating system architecture and its role as the central controller of the computer; process management concepts such as the process lifecycle, multitasking, concurrency, and scheduling .
- Memory management, multithreading, and the use of system calls to securely access hardware resources

- File system organization and the main types of operating systems, including real time, embedded, network, and mobile OS.

1.1 Introduction to Operating System (OS)

Operating System (OS) is a type of system software that manages hardware, runs application software, and provides a user interface like Windows, macOS, Linux, Android, and iOS, along with their basic roles. Here we will go a step further and focus on how the operating system works as the central controller of the computer system, ensuring smooth and secure interaction between the user and the hardware, especially in multi-user environments.

Operating System (OS) as the Central System Controller

An operating system works like a traffic controller for the computer. It decides which task should be done first, how the computer's memory is used, and which devices (like a printer or speakers) should be active at a given time. In short, we can say that the OS makes sure the computer works smoothly even when multiple tasks are running at the same time.

Role in User Hardware Interaction

Computer hardware cannot understand human language directly. The operating system acts as a translator between the user and the hardware system.

Example: When you click an icon or type something, the OS changes those actions into instructions that the hardware can understand. This allows people to use computers easily without learning how the hardware works.

DO YOU KNOW?

Humans speak languages like Urdu or English, whereas computers understand only binary language, or ON and Off switch i.e. 0 and 1.

Example: When you press the letter **A** on a keyboard, the operating system converts it into a special binary code:

- The letter **A** in binary is $(\mathbf{01000001})_2$.
- This code is then sent to the computer screen, which displays **A** on the screen.
- This process happens so quickly that it feels instant to the user.

Responsibilities in Multi-User Environments Tasks

In places like schools, offices, or online systems, many people may use the same computer. The operating system ensures:

- The creation and management of separate **user accounts**.
- Keeps each user's files and information **private**.
- Shares the computer's resources fairly so that no one slows down the systems functionalities.
- Protect shared resources from **unauthorized access**.

Creating and Managing User Accounts

The operating system allows separate accounts for different users, each with its own desktop, files, settings, and passwords.

Example: In computer lab, students login with a username and password provided by the school. This keeps each student's work separate and private. In most modern OS like Windows, a new account can be created through the Settings or Control Panel by selecting User Accounts, choosing Add New User, and entering details like username, password, and account type(Standard, Administrative, Guest).

1.2 Architecture of an Operating System

The architecture of an operating system is the way how its parts are organized and how they work together. Each part has a special role, and together they make the computer work smoothly, just like a school has different departments that perform specific duties but work together for the smooth working of the school.

Kernel vs Shell

I. **Kernel:** The kernel is the core part of the operating system, that directly controls the computer's system software and hardware such as the CPU, memory, and devices, as shown in Figure 1.1. It decides how and when different programs can use these resources.

Example: When you open a file, the kernel manages the process of reading it from the hard drive and sending it to the screen. Like Engine of a car is kernel and accessories like steering wheel, dashboard are shells.

II. **Shell:** The shell is the outer part of the OS that interacts with the user, also depicted in Figure 1.1. It receives commands from the user and passes them to the kernel. Shells can be:
Graphical shells (like Windows desktop) where you click icons and use menus.
Command-line shells (like the Command Prompt or Terminal) are where you type instructions.

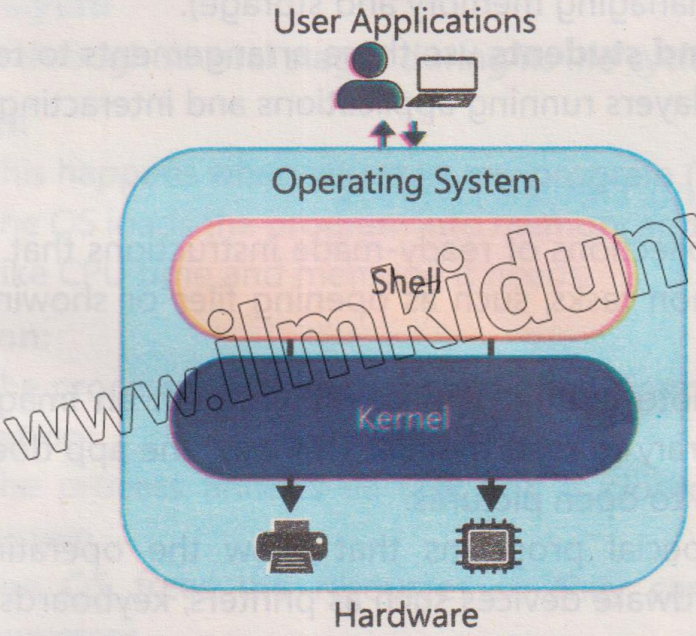


Figure 1.1: Diagram to illustrate the relationship between the user, shell, and kernel in an operating system

OS Layers and Modular Design

In operating systems, the design is divided into **layers**, where each layer has a specific job:

- **Lower layer** work directly with hardware devices like the CPU, RAM, storage and Hard drive.
- **Middle layer** manage these resources and make sure programs can use them when needed.
- **Upper layer** run applications and provide the interface that the user views on the screen.

Each layer depends on the one below it. This design makes the operating system easier to manage, repair, and improve without changing the whole system.

Example: A school system can be used as an example to understand the layered architecture of an operating system:

- The **support staff** (like guards and cleaners) work at the base, keeping the school ready (like the lower layers working with hardware).
- The **administration** manages resources, schedules, and rules (like the middle layers managing memory and storage).
- The **teachers and students** use these arrangements to teach and learn (like the upper layers running applications and interacting with users).

System Libraries and Device Drivers

System libraries are collections of ready-made instructions that programs can use to perform common tasks, such as opening files or showing text on the screen.

Example: When a photo editing app needs to open an image, it uses the operating system's library to read the file. This way, the app does not have to create its own method to open pictures.

Device drivers are special programs that allow the operating system to communicate with hardware devices such as printers, keyboards, and graphics cards.

ACTIVITY

Objective:

Identify the **shell, device driver,** and **system library** used by your computer.

Steps:

1. Locate the shell on your computer (desktop or command-line).
2. Open Device Manager (Windows) or System Information (Mac/Linux).
3. Find one hardware device (e.g., printer, keyboard) and note its driver name.
4. Open a simple program (e.g., Calculator) and identify one task it performs that may use a system library (e.g., displaying numbers).
5. Record your findings in a table with these columns:
   - **OS Component** (Shell / Device Driver / System Library)
   - **Your Example** (e.g., Windows desktop, HP printer driver, Calculator app)
   - **Purpose/Role** (what it does in the system).

1.3 Process Management in Operating System (OS)

The operating system is responsible for managing all the programs that run on a computer. In this context, these running programs are called **processes**. Process management ensures that each process gets the **resources** it needs, even when multiple processes are active. Process management is one of the most important jobs of an operating system because it ensures all processes run smoothly without disturbing each other.

Process Life Cycle

A process goes through several stages during its life cycle:

1. **Creation:**
   - This happens when you start any program (like MS Word).
   - The OS loads the program into memory and gives it the resources (like CPU time and memory) it needs.
2. **Execution:**
   - The process is actively running and performing tasks effectively.
3. **Termination:**
   - The process finishes its task and is closed by the user or the system.
   - The OS frees the resources so they can be used by other processes.

Example: Process Lifecycle of opening a Web Browser on your mobile or computer like Google Chrome.

1. **Creation:** Begins when the user clicks the browser icon. The operating system loads the program into memory and allocates the required resources.
2. **Execution:** The browser performs tasks such as loading web pages, displaying media, and responding to user actions.
3. **Termination:** Occurs when the browser is closed. The operating system stops the process and releases its resources for other uses.

Multitasking and Concurrency

Modern operating systems can manage many processes so efficiently that it appears they are all running at the same time.

- **Multitasking:** The operating system allows more than one program to be open and usable by a single user at the same time. You can easily switch between them whenever you need.
  Example: You can listen to music, keep a document open, and browse the internet, moving between them as needed.
- **Concurrency:** More than one process is active at the same time in an OS, but the CPU processes them one by one in extremely fast cycles.
  Example: Like a chef preparing three dishes, working on one for a short time, then moving to the next, and repeating, the CPU switches between processes so rapidly that the user does not notice any delay.

Process Scheduling Concepts

We have learned that many processes can be in progress during the same time period, but the CPU works on them one at a time in very fast turns. Since the CPU cannot run all processes at the same time, the operating system must decide:

- Which process should run first?
- How long each process should run.

This process is called **scheduling**. There are different scheduling methods used by operating systems. But in this unit, we will explore only one of the simplest scheduling techniques, ***First Come, First Served (FCFS)***.

In the ***FCFS*** method, the CPU processes tasks in the exact order they arrive. The first process to arrive is completed first, and the next process starts only after the previous one finishes.

Example: Like a queue at a shop counter, the first customer in line is served first, then the next, and so on.

Numerical Example: Let us use some simple data to see how **FCFS** scheduling works in practice.

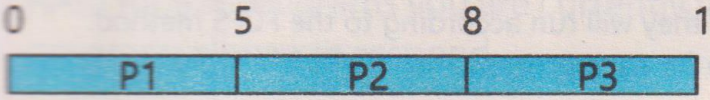
Process ID	Arrival Time	Time Needed
P1	0 seconds	5 seconds
P2	1 second	3 seconds
P3	2 seconds	2 seconds

Step-by-Step Explanation:

1. **At 0 seconds:**
   - **P1** arrives first, so it starts running immediately.
   - It requires **5 seconds** to finish.
   - It runs from **0s to 5s**.
2. **At 1 second:**
   - **P2** arrives while P1 is still running, so **P2** waits.
3. **At 2 seconds:**
   - **P3** arrives, but P1 is still running and P2 is already waiting.
   - In FCFS, P3 will also wait until both P1 and P2 have completed.
4. **At 5 seconds:**
   - P1 finishes.
   - The CPU now starts **P2**, which needs **3 seconds** to complete.
   - P2 runs from **5s to 8s**.
5. **At 8 seconds:**
   - P2 finishes.
   - The CPU now starts **P3**, which needs **2 seconds**.
   - P3 starts and runs from **8s to 10s**.

Execution Timeline:

[P1: 0–5] → [P2: 5–8] → [P3: 8–10]



P1 : 0s – 5s (finish at 5)
P2 : 5s – 8s (finish at 8)
P3 : 8s – 10s (finish at 10)

Observation:

Even though **P3** has the shortest duration (only 2 seconds), it must still wait until **P1** and **P2** have finished because they arrived earlier. This is one of the main characteristics of FCFS; shorter tasks can be delayed if they arrive after longer ones.

Advantages and Disadvantages of FCFS:

Although FCFS is a simple and fair scheduling method, it has both advantages and disadvantages:

Advantages: Easy to understand and implement, as processes are served in the exact order they arrive. No process is skipped.

Disadvantages: Short processes may have to wait a long time if they are queued behind longer processes (known as the "convoy effect"), which can affect the overall efficiency of the system.

DO YOU KNOW?

The CPU in your computer can switch between processes **millions of times** in just one second. This happens so fast that you can play music, browse the web, and download files all at once.

ACTIVITY

Objective:

Use the **First Come, First Served (FCFS)** method to determine the order and completion time of processes.

Steps:

1. Consider the following processes:

Process	Arrival Time	Time Needed
P1	0s	4s
P2	1s	5s
P3	2s	3s

2. Arrange the processes in the order they will run according to the FCFS method.
3. Calculate the **start** and **finish time** for each process.

Answer the questions:

- What is the total time taken for all processes to finish?
- Which process had to wait the longest?

1.4 Memory

Memory is a fundamental component of a computer system, used to store data and instructions required for processing. It ensures that the CPU can access information quickly during program execution.

Primary Memory (RAM)

Primary memory, also called Random Access Memory (RAM), is the main working area of a computer. It is a fast storage area where the computer keeps

the data and instructions temporarily. Its data is erased when the computer is turned off.

Example: When you open a Word document, the computer quickly places the program and the document into RAM so the CPU can work on them right away.

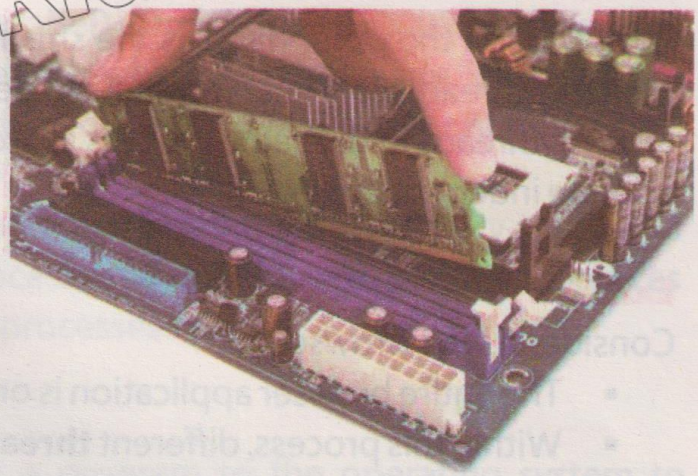


Figure 1.2: RAM

Virtual Memory

When the RAM is full, the operating system uses part of the computer's storage drive (like Hard drives, solid-state drives (SSD), or Non-Volatile Memory Express (NVMe)) as virtual memory. This extra space acts like temporary RAM, allowing more programs to run at the same time. Storage drives operate at lower data transfer speeds and have higher access times as compared to RAM; that's why the use of virtual memory can result in reduced system performance.

When several programs are open simultaneously, the operating system may transfer data from less active programs to virtual memory. This frees up space in RAM for the programs you are currently using, but may make those fewer active programs slower to respond.

DO YOU KNOW?

The first personal computer, the **IBM 5150** (1981), was equipped with only **16 KB** of RAM. In contrast, modern DDR5 RAM can transfer data at speeds exceeding **50 GB** per second, enabling the copying of a full high-definition movie in less than **one** second.

1.5 Processes and Threads

A **process** is an independent program that is currently being executed by the computer. It has its own memory space CPU time, and other resources (such as files or network connections). Processes are isolated from one another to ensure stability and security; a problem in one process generally does not affect another.

A **thread**, on the other hand, is the smallest unit of execution within a process. Multiple threads can exist inside a single process, each performing a different task. All threads in the same process share the same memory and resources, but operate independently.

Example:

Consider a web browser:

- The entire browser application is one **process**.
- Within this process, different **threads** are responsible for:
  - One thread is loading and rendering a webpage.
  - The second thread is playing audio or video content.
  - Another thread is downloading files in the background.

By using the above multiple threads, the browser can continue loading new content while playing a video, without making the user wait for one task to finish before starting another.

Multithreading

Multithreading is an operating system technique that allows a single process to perform multiple tasks at the same time by dividing its work into smaller units called **threads**.

Each thread runs independently but shares the same memory and resources of the process, enabling faster execution, better responsiveness, and efficient use of system resources.

Benefits of Multithreading

Multithreading offers several advantages, and some of the most important are discussed below:

- **Enhanced Performance:** Tasks can be divided into multiple threads and executed in parallel, allowing complex operations to complete more quickly and improving overall system efficiency.
- **Improved Responsiveness:** Applications remain responsive even when one thread is busy with a specific task.
  Example: A word processor enables continuous typing while another thread checks spelling in the background.

- **Support for Concurrent (parallel) Operations:** Multiple tasks can progress during the same period, which is particularly important in applications such as games, video editing, and real-time communication tools.
- **Efficient Use of Resources:** Threads share the same memory space and resources of their parent process, requiring fewer system resources compared to creating separate processes.

1.6 System Calls

A **system call** is a request made by a program to the operating system to perform a specific task that the program cannot do directly. They act as a bridge between **user programs** and the **kernel**, allowing applications to access hardware and core OS functions safely. Without system calls, programs that directly control hardware can be unsafe and complex.

Example: When you save a file in a text editor, the program uses a system call to tell the operating system to write the data to the storage drive(like a Hard drive).

Types of System Calls

The main types of system calls include:

1. **open** Opens a file for reading or writing.
   Example: Opening a music file to play.
2. **read** Retrieves data from a file or input device.
   Example: Reading text from a document.
3. **write** Sends data to a file or output device.
   Example: Saving an image to the computer.
4. **fork** Creates a new process by duplicating an existing one.
   Example: Opening a new browser tab, where the OS may use fork to create another process.

DO YOU KNOW?

Linux and **macOS** have a few hundred system calls, while **Windows** uses nearly **2,000**. These calls handle everything from opening files to running apps and showing graphics in the background while you work.

1.7 File System Structure and Management

An operating system not only runs programs but also stores and organizes the users' data. This is achieved through the **file system**, which provides a structured way for users and applications to store, locate, and manage information on storage devices. A well-designed file system ensures that data remains organized, secure, and easily retrievable, even when there are thousands of files contain complex data.

Files

A **file** is a collection of related data stored on a computer. It may contain text, images, audio, video, or program instructions.

Folders

A **folder** (also called a directory) is a container used to organize files and other folders in a logical structure, making it easier to locate and manage information.

Metadata

Metadata refers to details about a file, folder, such as its name, type, size, date of creation, and date of last modification. This information helps both the operating system and the user identify and manage files/ folders without opening them.

File Systems

A file system is the way an operating system stores and organizes files on a storage device, such as a hard drive, SSD, or USB. It decides where each file will be kept and save its location so it can be access later. Some widely used file systems include computer-based file systems are including:

- **FAT 32-** Used in USB flash drives
- **NTFS** – Used by Windows OS.
- **APFS / HFS+** – Used by macOS.
- **EXT4** – Commonly used by Linux OS.

The choice of file system affects performance, storage capacity, and security features.

Example: If you save a photo on your computer, the file system makes sure it is stored in the right place and knows exactly where to find it when you open it

again, just like a librarian placing a book in the right section and knowing where to locate it later.

ACTIVITY

Objective:
Demonstrate how system calls work when managing files.
Instructions:

1. **Prepare**:
   - Give each student a small card labeled with a file name (e.g., photo.jpg, homework.docx, song.mp3).
   - Write "**open**", "**read**", "**write**", and "**close**" on the board.
2. **Steps**:
   - Call one student the **"Operating System"**.
   - Another student plays the **"Application"** (e.g., Photo Viewer).
   - The Application requests: "OS, please **open** photo.jpg".
   - The OS responds by selecting the card from the "file list" and handing it over.
   - Repeat with **read**, **write**, and **close** for different files.
3. **Wrap-up Discussion**:
   - Explain that each action represents a **system call**.

1.8 Types of Operating Systems

Operating systems are designed according to the needs of the device and the work it performs. Each type serves a different purpose and is optimized for specific tasks.

Real-Time Operating System (RTOS)

A **Real-Time Operating System** is designed to process data and respond within a strict time limit, known as a **deadline**. It is used where even a tiny delay can cause system failure or serious consequences.

- **Key Feature:** Processes tasks immediately as they arrive, without long waiting times.
- **Used in:** Critical systems such as air traffic control, heart-monitoring devices, or industrial robots.

Embedded Operating System (EOS)

An **Embedded OS** is a small and highly efficient operating system built into a specific device to control only the functions it needs. It is not meant for general-purpose computing but is optimized for one task or a small set of tasks.

- **Key Feature:** Uses very little memory and power, and is often stored permanently inside the device.
- **Used in:** Home appliances (microwaves, washing machines), printers, smart TVs, and ATMs.

Network Operating System (NOS)

A **Network OS** manages and supports multiple computers connected through a network. It enables the sharing of resources like files, printers, and internet connections among users like windows multipoint servers.

- **Key Feature:** Focuses on communication and coordination between computers.
- **Used in:** Offices, schools, and data centers.

Example: In a school computer lab, students use different computers, but all can save files to the same server and print from the same printer, thanks to the Network OS.

Mobile Operating System

A **Mobile OS** is designed for smartphones, tablets, and other handheld devices. It is optimized for **touch-screen use**, **battery saving**, and **mobile apps**.

- **Key Feature:** Supports wireless connectivity, cameras, sensors, and app stores.
- **Used in:** Smartphones, tablets, smartwatches.

DO YOU KNOW?

The very first mobile operating systems, like **Symbian** (used in early Nokia phones), could only run a few small apps. Today's mobile OS platforms like **Android** and **iOS** can handle millions of apps, 3D games, and even professional video editing tools.

Summary

- Operating System (OS) is a type of system software that manages hardware, runs application software, and provides a user interface.
- The kernel is the core part of the operating system, that directly controls the computer's hardware and system software such as the CPU, memory, and devices.
- The shell is the outer part of the OS that interacts with the user.
- System libraries are collections of ready-made instructions that programs can use to perform common tasks, such as opening files or showing text on the screen.
- The operating system is responsible for managing all the programs that run on a computer. In this context, these running programs are called processes.
- In multitasking the operating system allows more than one program to be open and usable by a single user at the same time.
- In concurrency more than one process is active at the same time in an OS, but the CPU processes them one by one in extremely fast cycles.
- In the *FCFS* method, the CPU processes tasks in the exact order they arrive. The first process to arrive is completed first, and the next process starts only after the previous one finishes.
- When the RAM is full, the operating system uses part of the computer's storage drive (like Hard drives, solid-state drives (SSD), and Non-Volatile Memory Express (NVMe)) as virtual memory.
- A process is an independent program that is currently being executed by the computer. A thread, on the other hand, is the smallest unit of execution within a process.
- A system call is a request made by a program to the operating system to perform a specific task that the program cannot do directly.

EXERCISE

Multiple Choice Questions

1. Which is NOT an example of an operating system?

(a) Linux (b) Windows
(c) Photoshop (d) macOS

2. In multi-user environments, the OS ensures:

(a) Equal access and data privacy (b) All users share the same files
(c) Hardware is never used (d) Internet speed is increased

3. In Windows, user accounts can be created via:

(a) Task Manager (b) Control Panel or Settings
(c) Disk Management (d) BIOS settings

4. The core part of the OS that interacts directly with hardware is the:

(a) Shell (b) Kernel
(c) Device Driver (d) System Library

5. A graphical shell allows the user to:

(a) Type commands only (b) Click icons and use menus
(c) Interact only via code (d) Access only the BIOS

6. In the FCFS scheduling method, processes are served:

(a) Randomly (b) Shortest job first
(c) In the order they arrive (d) By priority only

7. Threads in the same process:

(a) Have separate memory spaces (b) Share the same memory and resources
(c) Run on different OS (d) Cannot run simultaneously

8. Which system call is used to create a new process?

(a) Open (b) read
(c) Write (d) fork

Short Questions

1. Why is the operating system referred to as the "central controller"?
2. How does the OS act as a translator between the user and hardware?
3. Define one way to create a new user account in Windows.
4. How is kernel different from shell.
5. Give one example of a system library and its purpose.
6. Explain one advantage and one disadvantage of FCFS scheduling.
7. Why is virtual memory slower than RAM?
8. Define a system call.
9. What is the role of a file system?
10. Define multitasking in computer system.

Long Questions

1. Describe the architecture of an operating system, explaining kernel, shell, and layered design with examples.
2. Explain the process lifecycle in detail with examples.
3. Differentiate between RAM and virtual memory, and explain how multithreading improves performance.
4. Describe system calls, their purpose, and give examples of at least three types.
5. A computer system uses the First-Come, First-Served (FCFS) scheduling method to manage processes. The following table shows the arrival time and the CPU burst time (time needed for execution) of three processes:

Process	Arrival Time (seconds)	Burst Time (seconds)
P1	0	4
P2	1	3
P3	2	1

Requirements:

I. Arrange the processes in the order they will be executed according to the FCFS scheduling method.

II. Show the execution sequence of all processes.
III. Calculate the start time and completion time for each process.
IV. Find the waiting time for each process.
V. Calculate the average waiting time for all processes.

Answer Key for Multiple Choice Questions

1. **c)** - Photoshop
2. **a)** - Equal access and data privacy
3. **b)** - Control Panel or Settings
4. **b)** - Kernel
5. **b)** - Click icons and use menus
6. **c)** - In the order they arrive
7. **b)** - Share the same memory and resources
8. **d)** - fork