Operating Systems Lab

Final Project

Operating System Simulator

Project Statement

An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs. In this project you will design a simulation of operating system on your own. The simulation should be made using and following all concepts studied in this course.

Key features the operating system should have:

- Your operating system should have a name of its own that will display on starting the program
  Detail: You should make the program in such a way that when it starts it displays the name of your operating system on terminal using sleep command for some seconds like it is loading.
- The RAM hard drive and the number of cores will be given by the user while starting the OS

```
(base) Mateens-MacBook-Pro:desktop mateencheema$ g++ os.cpp -o OS
(base) Mateens-MacBook-Pro:desktop mateencheema$ ./OS 2 256 8
```

  This instance of OS will have 2GB RAM, 256GB Hard drive and 8 cores you can choose your own convention.

- The resources should be managed in you operating system RAM Hard drive and cores should be kept in mind while making new processes or threads.
- You will define basic applications and tasks that your operating system can perform
  - Basic tasks would be like Notepad (writing into file with auto save function), Calculator, Time, creating a file, Move, Copy, Delete file, Check file info, Minigames (minesweeper etc.)
  - You should have minimum of 15 tasks in your operating system

  Detail: Each task should have a defined task its own separate code file and it should make a new process simple function calling is strictly not allowed the once a process is created it will send the creating process message that will contain memory required in Hard drive and RAM if available then the process will reply grantee else the process should be terminated each process will have its own terminal so the output is not messed up.

This will help you to enable multi-tasking in your operating system.

- You can also manually create an interruption to stop the specific task or minimize it.
  Detail: Each task should have a button to close the task in between so to make a simulation of interrupt.
- Your operating system should be able to multitask.
- Your operating system would have user mode and kernel mode (in kernel mode you can access hardware resources)

  Detail: In kernel Mode user can close or delete the processes from memory meaning you can close running programs.

Project Working:

- **Hardware Resources**
  - Hardware resources must be provided for the OS to start.
- **Operating System Boot**
  - OS name appears on the **boot screen**.
  - After booting, **basic tasks** (like time and calendar) should auto-start.
  - You can add more **startup tasks** as needed.
- **Instruction Guide**
  - A guide should show all **available tasks** a user can perform.
- **Task Execution**
  - User selects a task → it starts in a **new terminal** using **EXEC commands**.
  - The task is given a **specific location and size in RAM**.
  - Task waits for its **turn to execute**, managed by **scheduling techniques**.
- **Process Scheduling**

  - If the number of tasks increases, **schedule them** based on available CPU **cores**.
  - A **ready queue** manages and sends tasks from RAM for execution.

- **Handling Interrupts**

  - If an **input or external interrupt** occurs:
    - The task moves to **Blocked State**.
    - Processor can execute another task in the meantime.
    - After the interrupt ends, the original task resumes.

- **Process Lifecycle**

  - Each task executes based on its **turnaround time** (set during coding, **not at runtime**).
  - After a task finishes, it is **removed from RAM**.

- **Data Storage**

  - If the user saves something, it should be stored in the **hard disk**.

Operating system concepts:

Your project should involve the following OS concepts:

- Multitasking
- Context switching
- Resource allocation
- User mode and Kernel mode
- Process creation
- Threads
- EXEC commands
- Scheduling using mutual exclusion, semaphore and condition variable
- Scheduling techniques in ready queue. You should implement multilevel queue with different techniques on each level.

All the above concepts should be used in order to get full marks.

Bonus Task: You will get bonus marks for implementing it using the graphics library.

Project Full Workflow:

Your program starts by requesting system resources and then allows the Operating System to load.

If you're using graphics or Windows Forms:

- Show icons for each available task — just like real OS desktops!

If you're working in the terminal:

- Display a clear list of tasks that users can choose from.

Task Management Made Easy

- When a user selects a task:
  - They should have the option to Close or Minimize it.
  - Close: Ends the task, removes it from RAM, and frees up space.
  - Minimize: Keeps it in RAM, but hides it — allowing the user to do something else.
- There should also be an option to switch to any currently running task in RAM — perfect for multitasking!

Important Rule:

- The number of running tasks should not exceed available RAM size — manage this carefully.