National University
Of Computer and Emerging Sciences

[CL-2018] OPERATING SYSTEM

PROJECT REPORT

"BASIC OPERATING SYSTEM WITH PROCESS AND MEMORY MANAGEMENT"

Group Members:

Name	Roll No
Shayan Hashmi	22k-4865
Syed Ammar Zulfiqar	22k-4845
Murtaza Hussain	22k-4863

SECTION: A

Instructor: Engr. Muhammad Afnan Malik

DEPARTMENT OF ELECTRICAL ENGINEERING

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES) KARACHI CAMPUS

1. INTRODUCTION:

➢ MOTIVATION:

The spine of a modern computer controls the operating system (OS) processes, memory, hardware resources, and file systems. Essential for industries in system-level programming, the construction of a basic OS simulation allows students to learn fundamental ideas well, including process scheduling, memory management, and file system operations.

➢ PROBLEM STATEMENT WITH TARGETED SDGs:

Modern operating systems are quite complex and challenging to understand. The project wants to recreate a summary OS with fundamental file handling, memory allocation, process scheduling, and command-line interface (CLI) interaction. Through improving students' knowledge in these domains, the initiative fits:

SDG 4 (Effective Education): By developing educational equipment that makes learning concepts easier-
SDG9 (industry, innovation and infrastructure): by promoting innovation in embedded systems and computing.

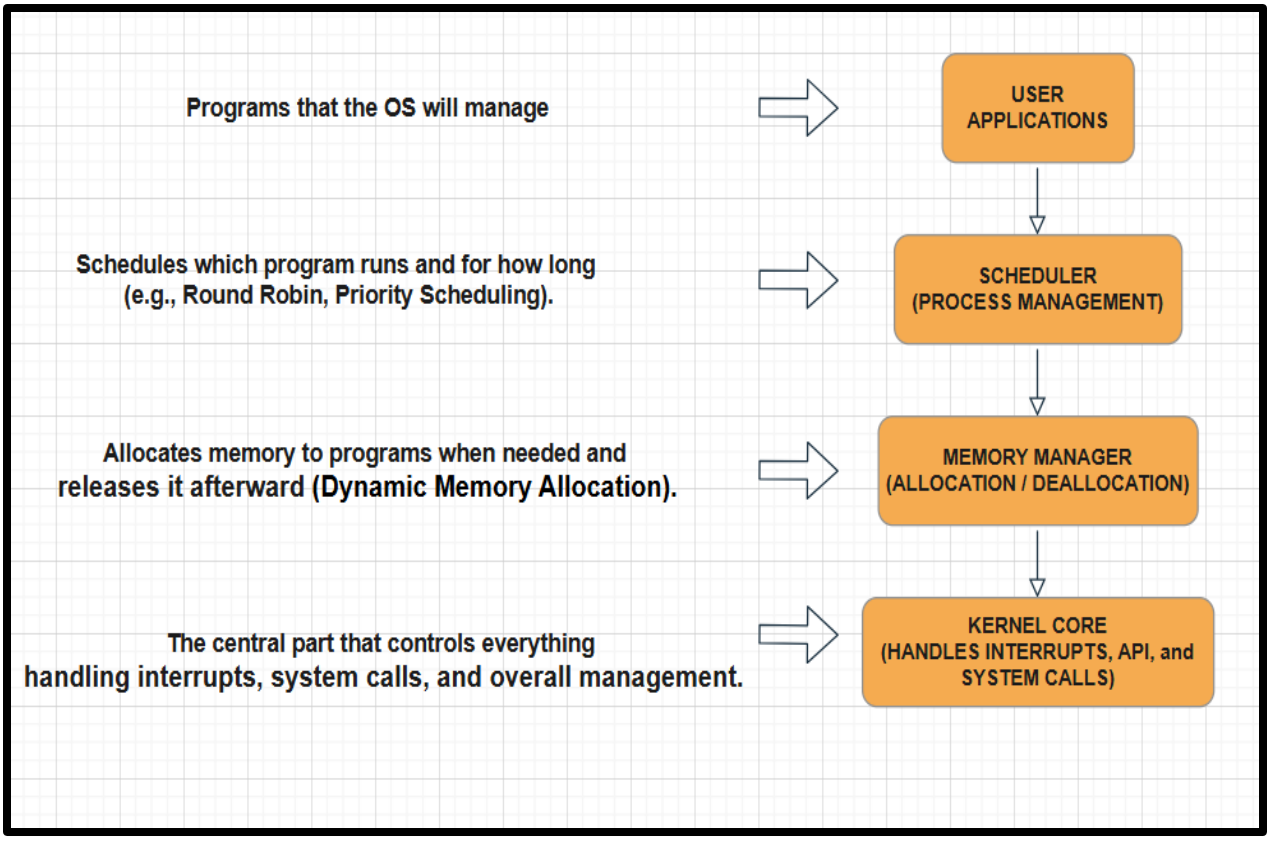
- **Application Interface:**
  Development will target simulation on a virtual machine (VM) or Raspberry Pi platform, demonstrating system operations in a controlled environment.

➢ PROPOSED SOLUTION WITH BLOCK DIAGRAM:

We propose a mild basic operating system that is capable:

- Determination of many tasks (procedure)
- Dynamically allocate and deal
- Visualizing Process States (Ready, Running, Waiting)

➢ Block Diagram:



2. LITERATURE REVIEW:

The design and implementation of a basic operating system has been a well -researched domain in the last few decades, especially focusing on procedure and memory management. Many scholars and researchers have proposed various light operating systems, real-time schedules, and dynamic memory managers aimed at customizing the performance of the system, especially in a constrained environment such as embedded systems. Tananbam et al. (2015) discussed the architecture of the microcinal-based operating system, which highlights the separation of process management and memory management functions in the user space, leading to increased safety and modularity [1]. Similarly, Silberschhatz et al. (2018) Fundamental procedure such as round robin and priority scheduling is detailed on the scheduling algorithm, which are fundamental for modern operating systems [2]. In terms of embedded operating systems, Dunkles (2004) introduced a mild and flexible operating system for the contextual OS, memory-world equipment, which focuses heavy on efficient memory management [3]. Liu et al. , Dynamic memory allocation was severely studied by Berger et al. (2000), where authors suggested various heap organization strategies to reduce fragmentation, an important factor for operating systems that handle multiple processes simultaneously [5]. Building on Memory Management Concepts, Hanson (1990) discussed the design and implementation of a dynamic storage allocation, which forms a fundamental building block for any memory management unit [6]. In real -time systems, Sha et al. (2004) emphasized the importance of forecastable memory

3. METHODOLOGY:

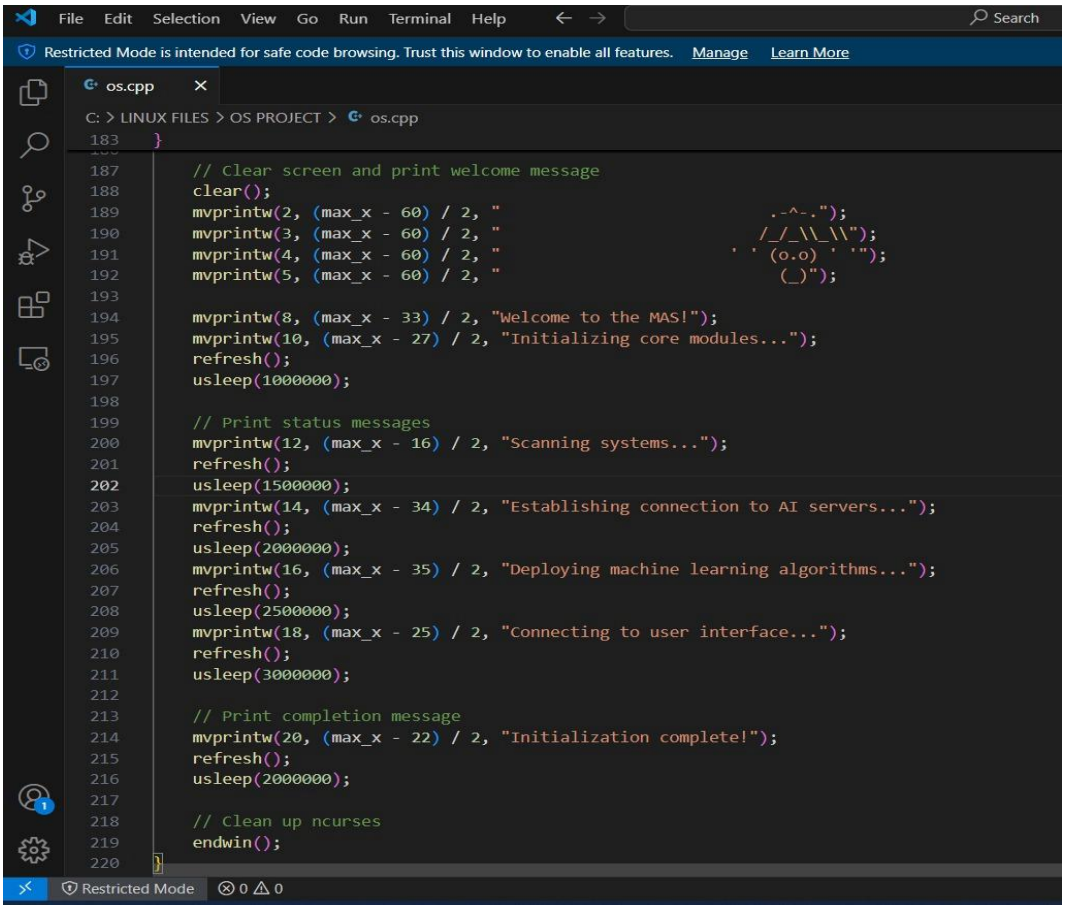
The development was done in several stages, focusing on modular, feature-powered design. All modules were developed in C++ and executed in the Linux environment using the terminal interface.

System Design: Each feature (calculator, calendar, game, file operations, etc.) was separated.

Process Management: Each task (eg, playing a game, performing a file operation) was considered like a process. After each task, the process switching was imitated by returning to the main menu.

Memory Management: Memory was dynamically allocated for files and games. Memory dealing routine was handled in files like Deallocate.cpp.

- **Game and Utility Development:**
  - Tower of Hanoi
  - Tic Tac Toe
  - Hangman
  - Number Guessing
  - Calculator
  - Calendar
  - Clock
  - Notepad
- **Compilation and Execution:**
  Files were compiled individually or together using the Linux g++ compiler.



We implemented different cases of multiple functions:

- **Case 1: Tower of Hanoi**

```
{
case 1:
    pid = fork();
    if (pid == -1)
    {
        cout << "Error in forking\n";
        exit(EXIT_FAILURE);
    }
    else if (pid == 0)
    {
        const char* command = "gnome-terminal";
        const char* arg1 = "--";
        const char* arg2 = "sh";
        const char* arg3 = "-c";
        const char* arg4 = "./tower";

        // Check if the executable file exists
        if (access("tower", F_OK) == -1) {
            // Compile the program
            system("g++ -o tower towerofhanoi.cpp");
        }

        char* args[] = { const_cast<char*>(command), const_cast<char*>(arg1), const_cast<char*>(arg2), const_cast<char*>(arg3), const_cast<char*>(arg4),
        execvp(command, args);
    }
    break;
```

- **Case 2: Calculator**

```
case 2:  //Calculator
    {
        cout << "Error in forking\n";
        exit(EXIT_FAILURE);
    }
    else if (pid == 0)
    {
        const char* command = "gnome-terminal";
        const char* arg1 = "--";
        const char* arg2 = "sh";
        const char* arg3 = "-c";
        const char* arg4 = "./calculator";

        // Check if the executable file exists
        if (access("calculator.cpp", F_OK) == -1) {
            // Compile the program
                return 0;
            system("g++ -o calculator calculator.cpp");
        }

        char* args[] = { const_cast<char*>(command), const_cast<char*>(arg1), const_cast<char*>(arg2), const_cast<char*>(arg3), const_cast<char*>(arg4), N
        execvp(command, args);
    }
    break;
```

So, like this, we also have more cases for other functions, e.g., calendar, clock, tic tac toe, notepad, etc.

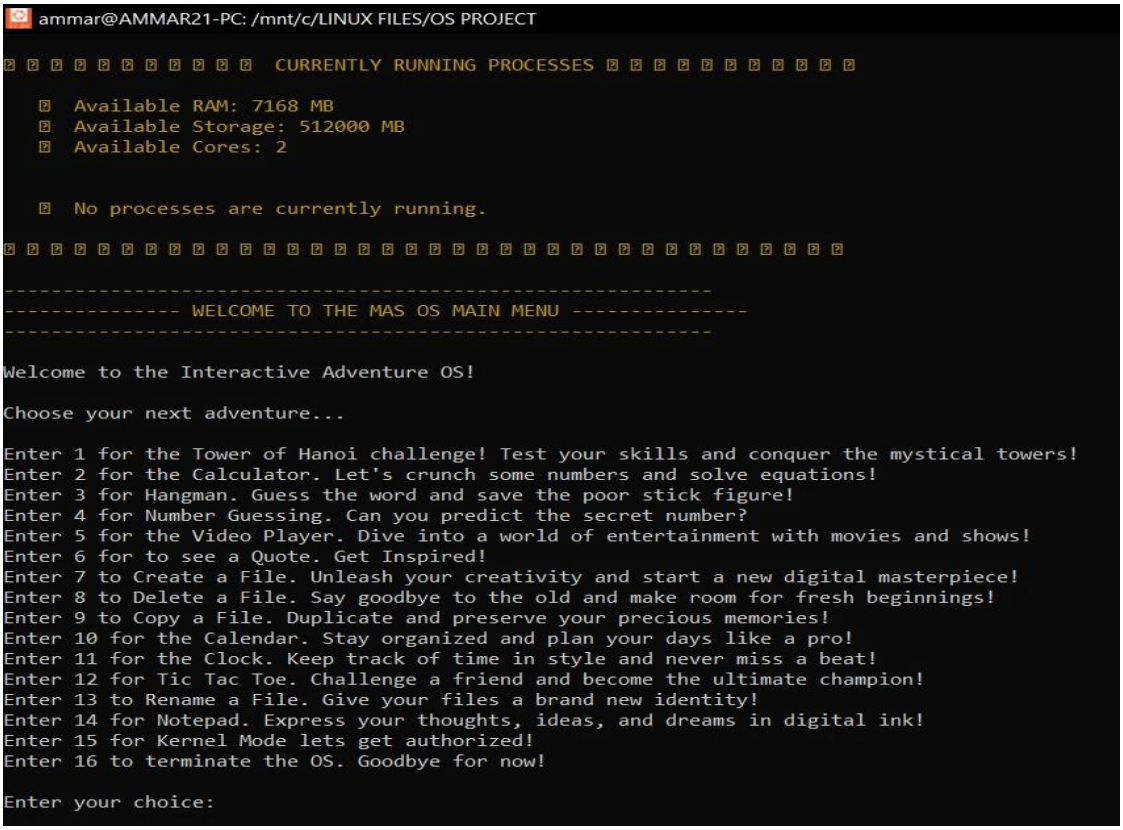
4. RESULTS:

The project successfully met its design goals:

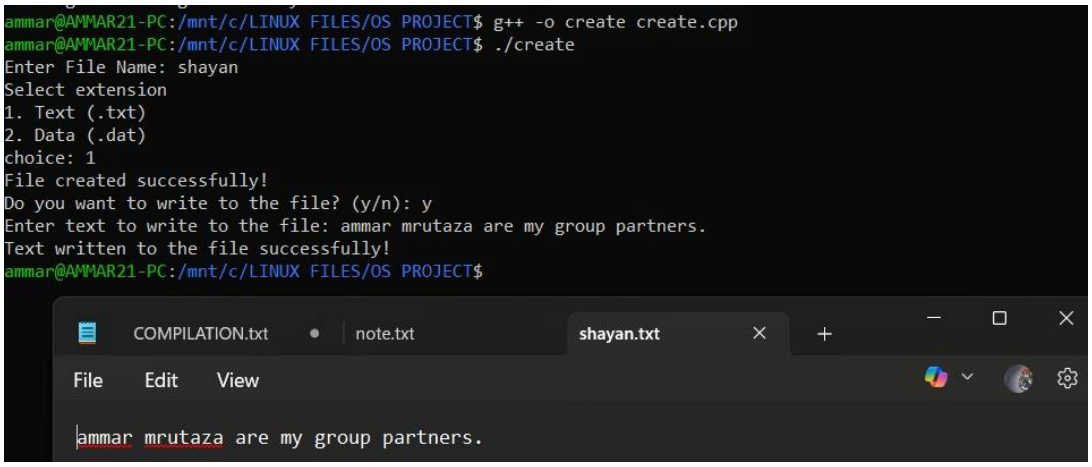
- All major modules function correctly within a Linux terminal.
- The system allows users:
  - Solve puzzles and play games.
  - Operate file (Create, Copy, Delete, Change Name).
  - Use productivity features like calendar, clock, and notepad.
  - Switch between tasks basically through the main menu.

➢ Interface:

➢ Main Menu:



➢ Notepad:



➢ Calculator:

```
ammar@AMMAR21-PC: /mnt/c/LINUX FILES/OS PROJECT
Loading...|/-\|/-\|/-\|/-\|/-\
Please select an operation:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Enter your choice (1-5): 1
Enter two numbers to add:
4
5
Result: 9

Please select an operation:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Enter your choice (1-5): 4
Enter two numbers to divide:
0
6
Result: 0

Please select an operation:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Enter your choice (1-5): 3
Enter two numbers to multiply:
3
6
Result: 18

Please select an operation:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Enter your choice (1-5): 5
Exiting Calculator...
ammar@AMMAR21-PC:/mnt/c/LINUX FILES/OS PROJECT$
```

➢ Calendar:

```
ammar@AMMAR21-PC:/mnt/c/LINUX FILES/OS PROJECT$ g++ -o calendar calender.cpp
ammar@AMMAR21-PC:/mnt/c/LINUX FILES/OS PROJECT$ ./calendar

Enter the year (or 0 to exit): 2025
Enter the month (1-12): 4
----------------------------
         2025 /  4
----------------------------
 Sun Mon Tue Wed Thu Fri Sat
           1   2   3   4   5
   6   7   8   9  10  11  12
  13  14  15  16  17  18  19
  20  21  22  23  24  25  26
  27  28  29  30
----------------------------

Enter the year (or 0 to exit): 0

Exiting Calendar Program. Goodbye!
ammar@AMMAR21-PC:/mnt/c/LINUX FILES/OS PROJECT$
```

- **Clock:**

```
ammar@AMMAR21-PC: /mnt/c/LINUX FILES/OS PROJECT
Current Time: 2025-04-28 22:07:45

Press '0' to exit, or any other key to refresh the clock...
0

Exiting Clock Program. Goodbye!
ammar@AMMAR21-PC:/mnt/c/LINUX FILES/OS PROJECT$
```

- **Tower Of Hanoi:**

```
ammar@AMMAR21-PC:/mnt/c/LINUX FILES/OS PROJECT$ g++ -o towerofhanoi towerofhanoi.cpp
ammar@AMMAR21-PC:/mnt/c/LINUX FILES/OS PROJECT$ ./towerofhanoi
----------------------------
|||||    THE TOWER OF HANOI    |||||
----------------------------
1. Playing game
2. Exit the game
1

first cupboard:
1
2
3
4
5

second cupboard:

third cupboard:

1. Move from 1 to 2
2. Move from 1 to 3
3. Move from 2 to 1
4. Move from 2 to 3
5. Move from 3 to 1
6. Move from 3 to 2
1
Do you want to quit? (y/n): n

first cupboard:
2
3
4
5

second cupboard:
1

third cupboard:

1. Move from 1 to 2
2. Move from 1 to 3
3. Move from 2 to 1
4. Move from 2 to 3
5. Move from 3 to 1
6. Move from 3 to 2
2
```

➢ Process Management:

5. DISCUSSION:

This project was highly successful in displaying:

- **Process Management:** Smooth transition between various tasks, imitating basic multitasking.
- **Memory Management:** Dynamic memory allocation and deallocation were handled safely and efficiently.
- **User Interaction:** A clear and interactive menu system increased the purpose.

Facing challenges: To integrate several .cpp files and ensure smooth compilation. Maintaining memory security in different modules.

How the objectives were obtained: Careful modular coding ensured the separation of functionality. Hard tests using devices such as GDB and Valgrind helped identify and fix issues.

6. CONCLUSION:

The Basic Operating System Project performed a successful implementation of fundamental OS concepts, including process management and memory management in a fully software-simulated Linux environment.

Through this project, Core OS principles such as process states, reference switching, and dynamic memory allocation were deepened and practically implemented. The system showed stability, fair scheduling, and efficient memory use. The use of Linux as a platform allowed effective testing, debugging, and performance analysis.

- **Future recommendations:** Introduce multithreading to allow real-time process performance. Add a user authentication system for better safety. Develop a basic file system for more realistic storage management. Apply a graphical interface (GUI) on top of the command-line system.

7. REFERENCES:

[1] A. S. Tanenbaum, and H. Bos, **Modern Operating Systems**, 4th ed., Pearson, 2015.

[2] A. Silberschatz, P. B. Galvin, and G. Gagne, **Operating System Concepts**, 9th ed., Wiley, 2018.

[3] A. Dunkels, "**The Contiki Operating System**," Swedish Institute of Computer Science, 2004.

[4] J. Liu, W. Wang, and Z. Zhang, "**Research on Task Scheduling Mechanism for Embedded Real-Time Operating Systems**," IEEE International Conference on Embedded Software and Systems, 2010.

[5] E. D. Berger, K. S. McKinley, R. D. Blumofe, and P. R. Wilson, "**Hoard: A Scalable Memory Allocator for Multithreaded Applications**," ACM SIGPLAN Notices, vol. 35, no. 11, pp. 117-128, 2000.

[6] D. R. Hanson, "**Fast Allocation and Deallocation of Memory Based on Object Lifetimes**," Software—Practice & Experience, vol. 20, no. 1, pp. 5-12, 1990.

[7] L. Sha, R. Rajkumar, and J. P. Lehoczky, "**Real Time Operating Systems Support for Predictable Execution**," Real-Time Systems Journal, vol. 5, no. 4, pp. 285-302, 2004.

[8] J. Park and D. Baik, "**Lightweight Semaphore Mechanism for Resource Synchronization in Embedded Systems**," Journal of Systems Architecture, vol. 77, pp. 1-10, 2017.

[9] L. Zhao, H. Wu, and W. Sun, "**A Modular Memory Protection Mechanism for Embedded Operating Systems**," Journal of Software Engineering and Applications, vol. 5, no. 6, pp. 435-441, 2012.

[10] M. Joseph and M. L. Soffa, "**Compiler-Generated Memory Management and Its Impact on Real-Time Systems**," Proceedings of the IEEE Real-Time Systems Symposium, 1991.