**MINI OS PROJECT REPORT  
  
CREATED BY : ZAINAB ARIF**

**Introduction**

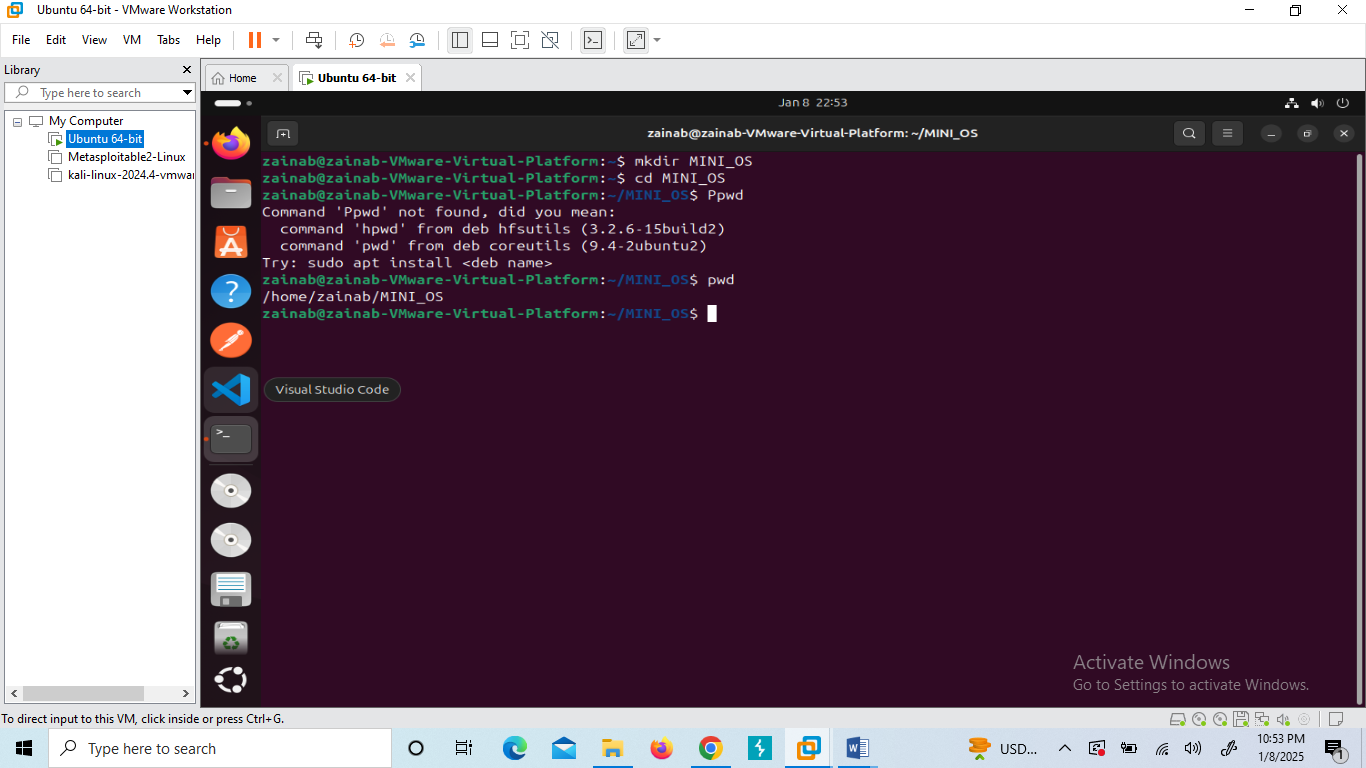
Operating Systems (OS) are essential for managing computer hardware and software resources. This project, **MINI OS**, demonstrates fundamental OS concepts, including **process management, memory management, inter-process communication (IPC), process synchronization, and I/O handling**. The project is implemented in **C programming language** on a Linux-based system.

**Step 1: Setting Up the Workspace**

**Environment Setup**

To begin, set up the working directory in Ubuntu:

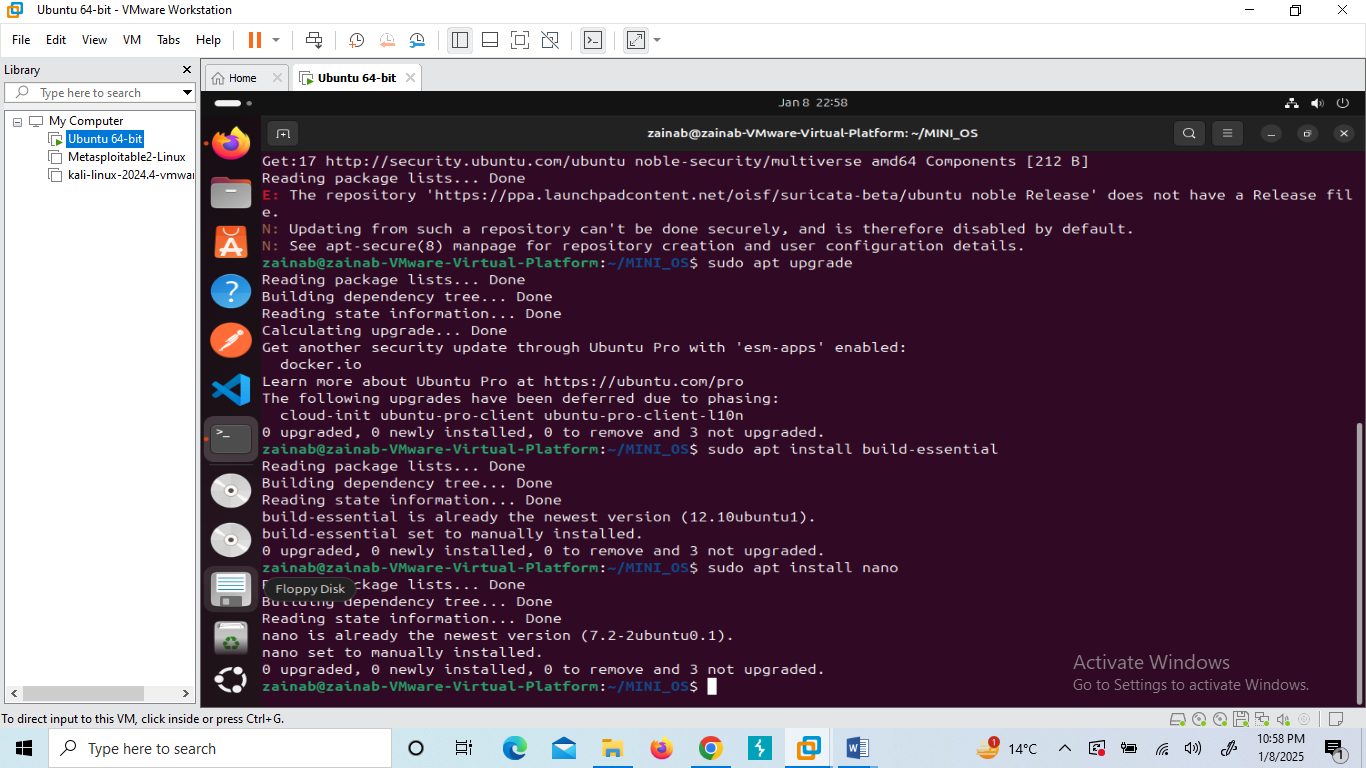
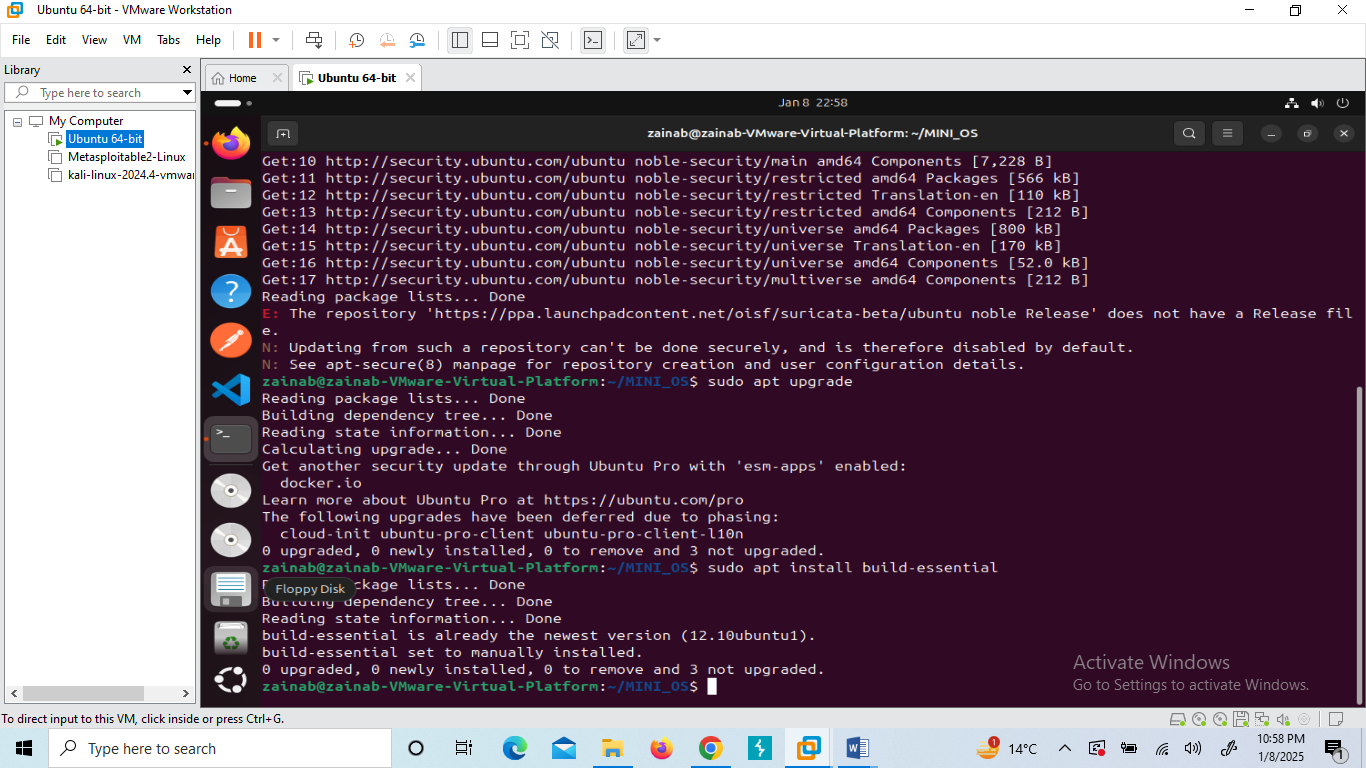
mkdir MINI\_OS

cd MINI\_OS  
  


**Required Tools Installation**

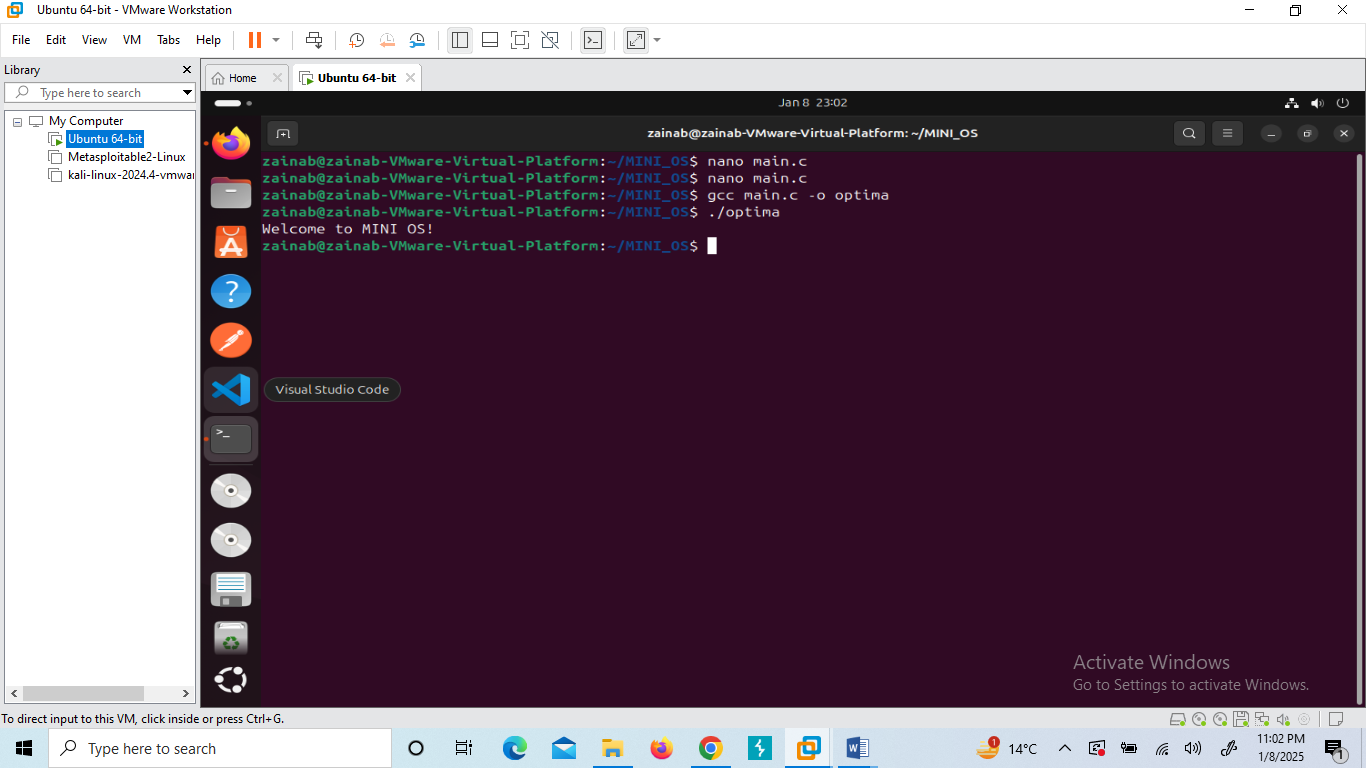
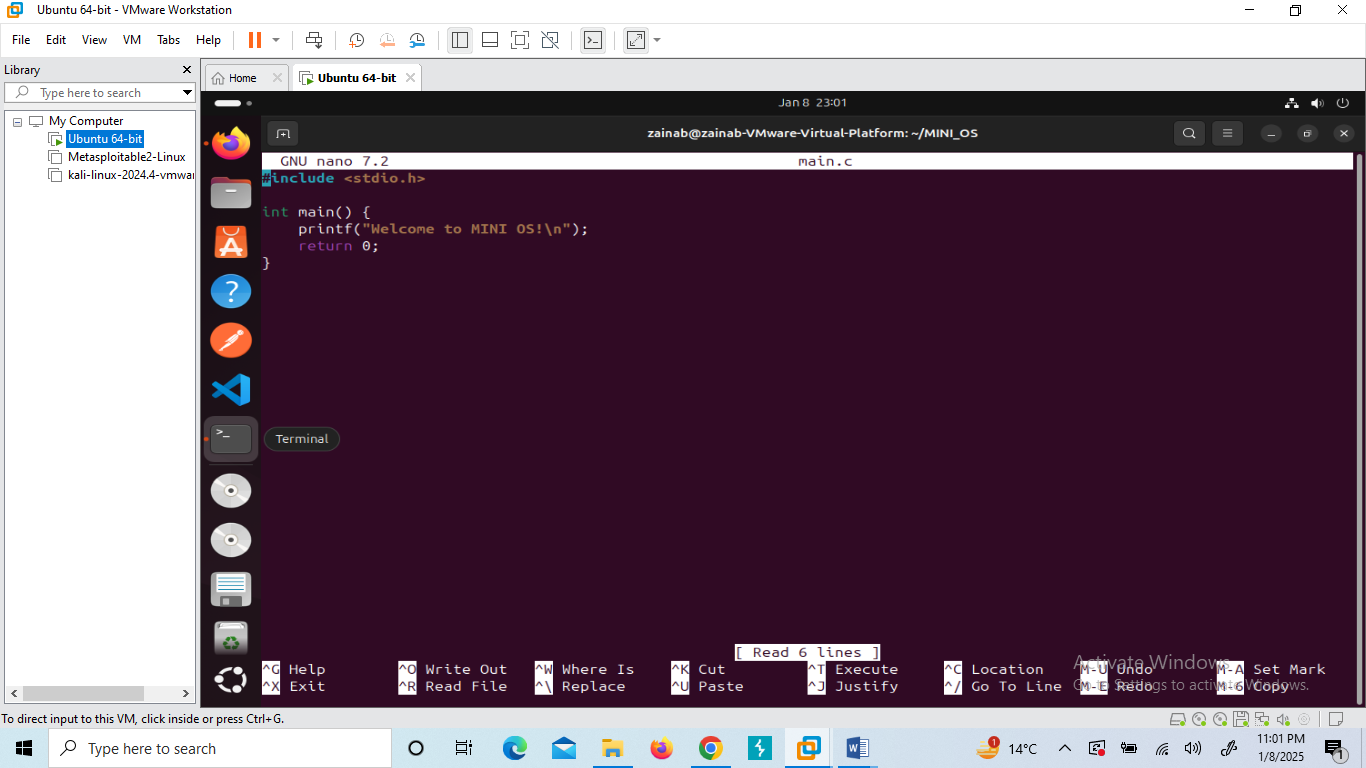
Update system packages and install necessary tools:

sudo apt update && sudo apt upgrade

sudo apt install build-essential nano  


**Step 2: Implementing Features**

The project consists of multiple features, each covering a key OS concept.



**Feature 1: Process Management (First Come, First Serve - FCFS)**

**Implementation:**

* Created a process scheduling mechanism using the **First Come, First Serve (FCFS)** algorithm.
* Each process has a **burst time**, and the scheduler calculates **waiting time and turnaround time**.

#include <stdio.h>

void fcfs(int processes[], int n, int burstTime[]) {

int waitTime[n], turnAroundTime[n], totalWaitTime = 0, totalTurnAroundTime = 0;

waitTime[0] = 0;

for (int i = 1; i < n; i++) {

waitTime[i] = waitTime[i - 1] + burstTime[i - 1];

}

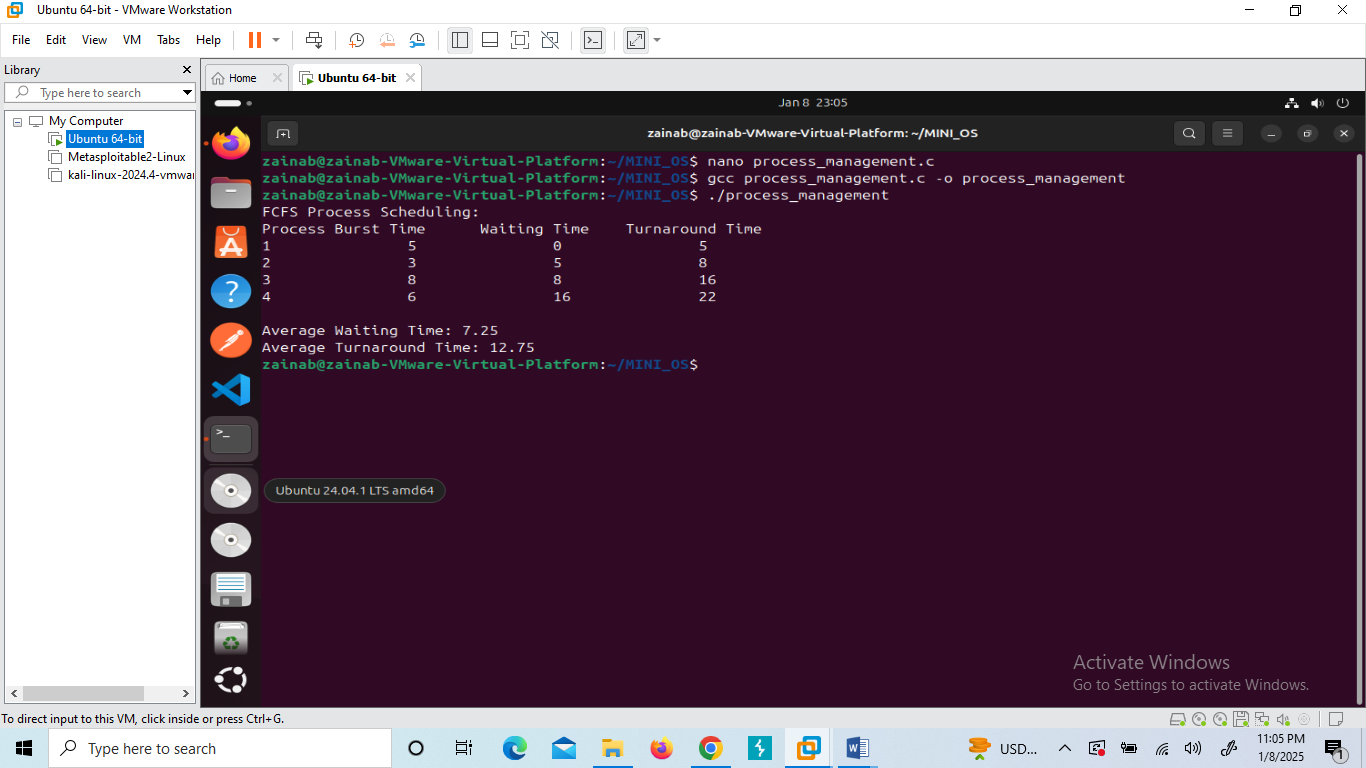
for (int i = 0; i < n; i++) {

turnAroundTime[i] = waitTime[i] + burstTime[i];

totalWaitTime += waitTime[i];

totalTurnAroundTime += turnAroundTime[i];

}

}  
  
  


**Feature 2: Memory Management (Dynamic Allocation)**

**Implementation:**

* Used **malloc()** and **free()** to allocate and deallocate memory dynamically.
* Avoids memory leaks by ensuring proper deallocation.

#include <stdio.h>

#include <stdlib.h>

void memory\_demo() {

int \*arr, n;

printf("Enter the number of elements: ");

scanf("%d", &n);

arr = (int \*)malloc(n \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed!\n");

return;

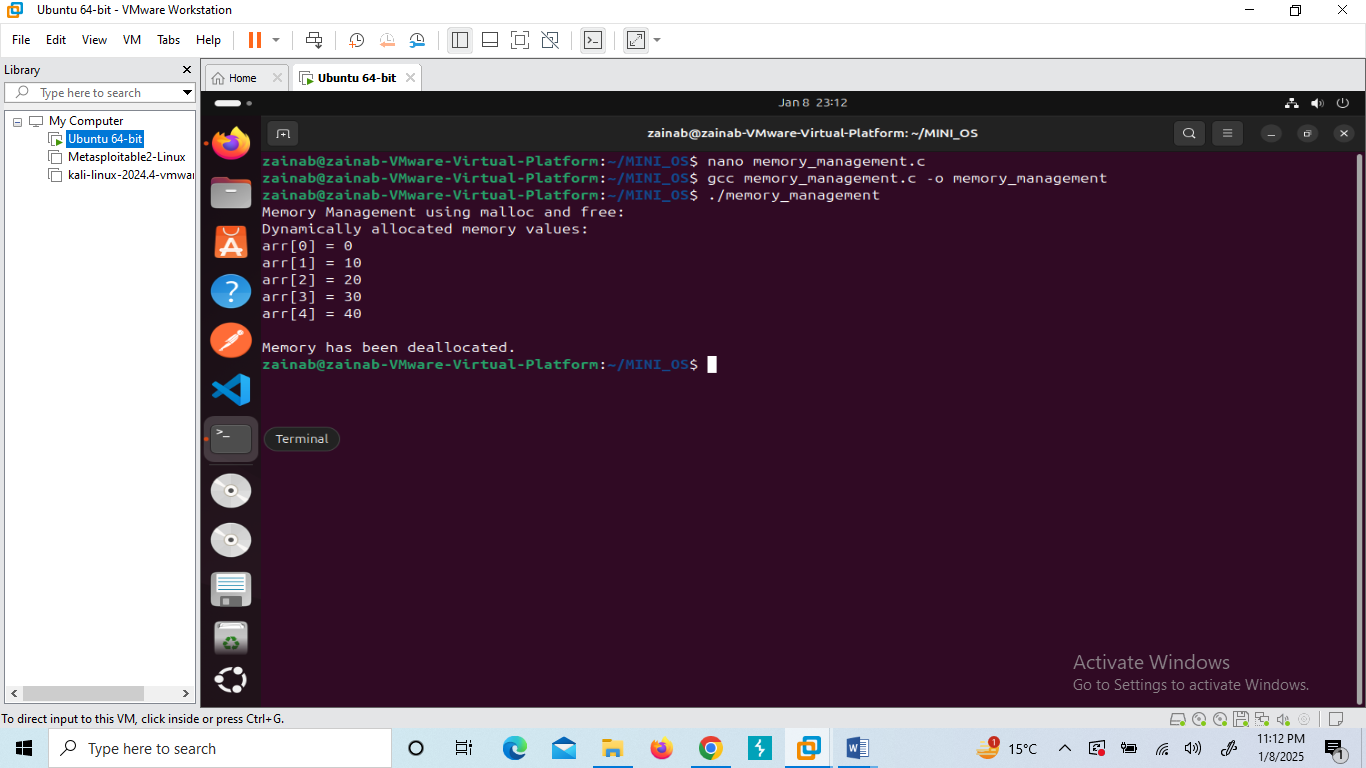
}

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

free(arr);

}  
  


**Feature 3: Inter-Process Communication (IPC - Shared Memory)**

**Implementation:**

* Used **shmget(), shmat(), and shmdt()** for creating and managing shared memory.

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <string.h>

int main() {

key\_t key = 1234;

int shmid = shmget(key, 1024, 0666 | IPC\_CREAT);

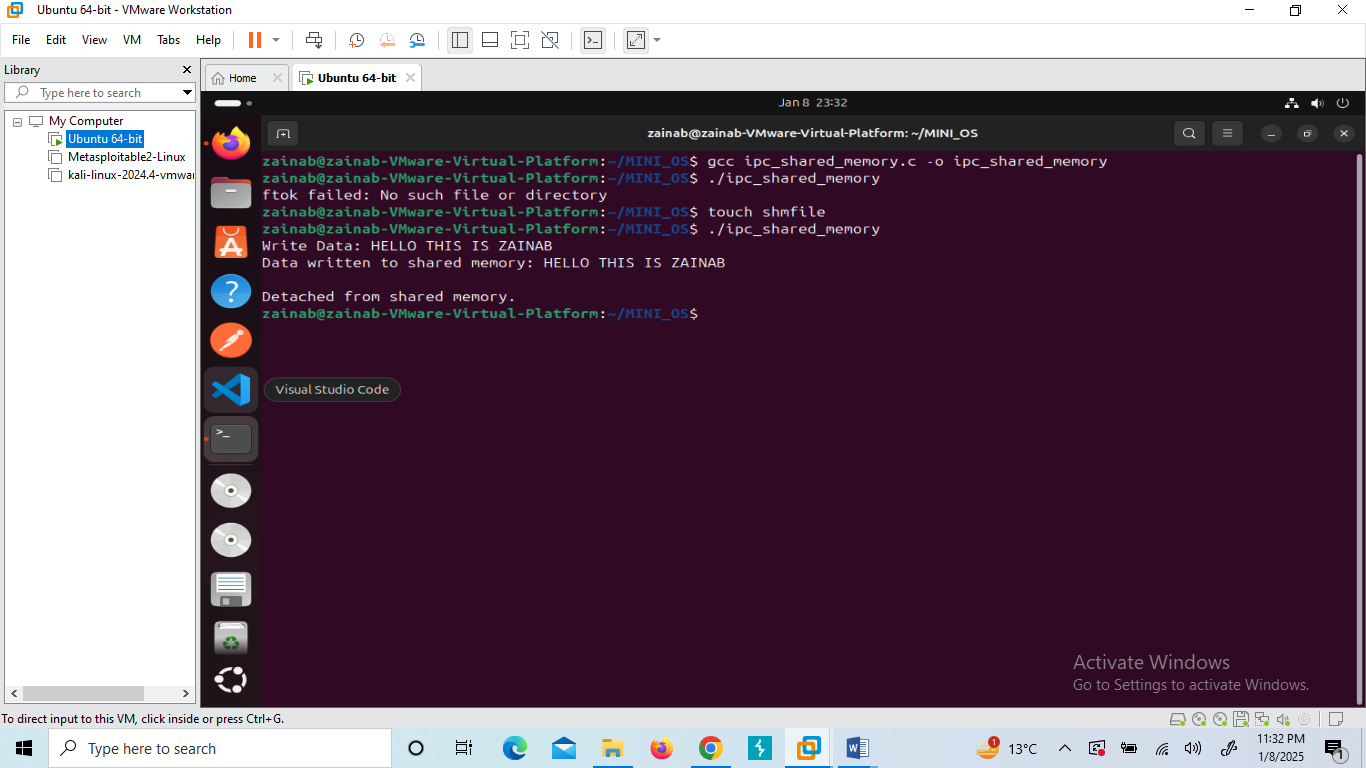
char \*str = (char \*)shmat(shmid, (void \*)0, 0);

printf("Write Data: ");

fgets(str, 1024, stdin);

shmdt(str);

return 0;

}  
  


**Feature 4: Process Synchronization (Mutex Locks)**

**Implementation:**

* Used **pthreads and mutex locks** to prevent race conditions.

#include <stdio.h>

#include <pthread.h>

int counter = 0;

pthread\_mutex\_t lock;

void\* increment\_counter(void\* arg) {

pthread\_mutex\_lock(&lock);

counter++;

pthread\_mutex\_unlock(&lock);

return NULL;

}

int main() {

pthread\_t threads[5];

pthread\_mutex\_init(&lock, NULL);

for (int i = 0; i < 5; i++) {

pthread\_create(&threads[i], NULL, increment\_counter, NULL);

}

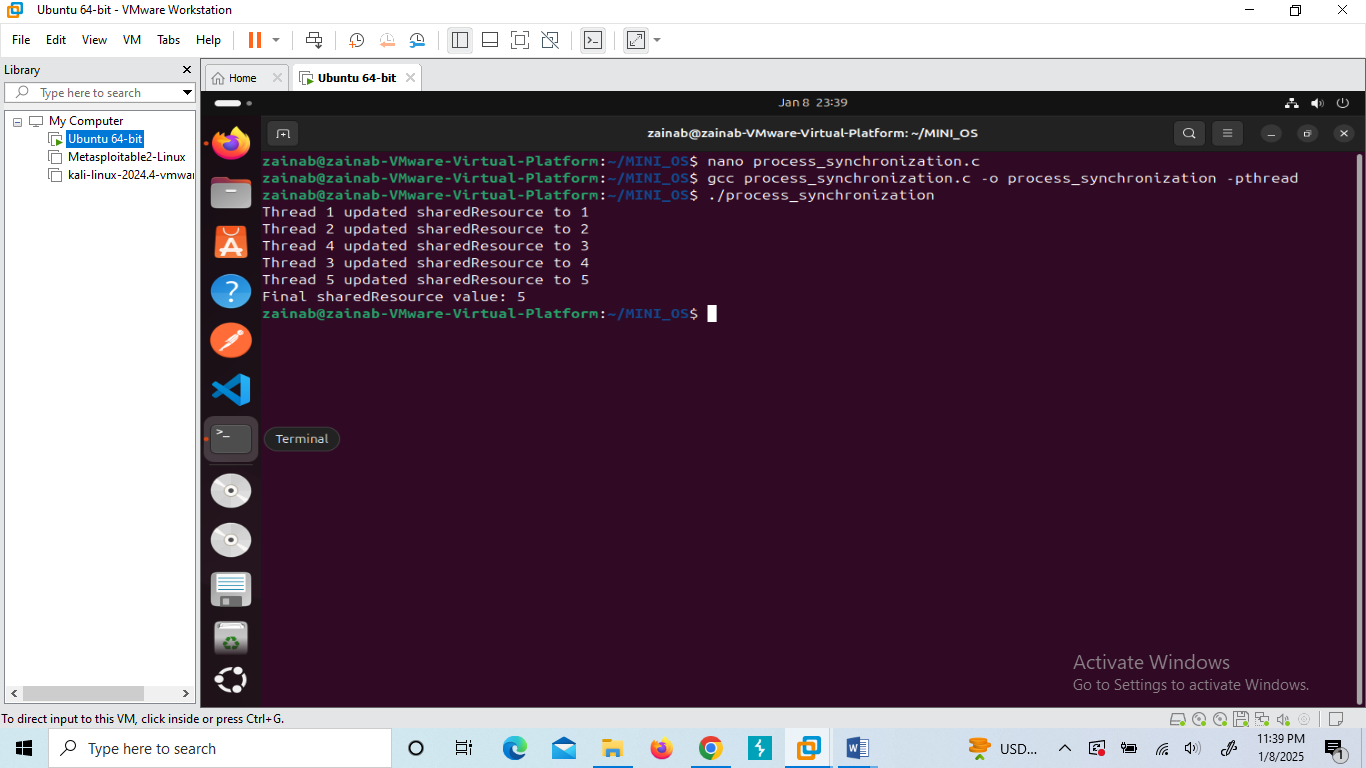
for (int i = 0; i < 5; i++) {

pthread\_join(threads[i], NULL);

}

pthread\_mutex\_destroy(&lock);

return 0;

}  
  


**Feature 5: I/O Handling (System Calls)**

**Implementation:**

* Used **open(), write(), read(), and close()** system calls for file handling.

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int file;

char buffer[100];

file = open("output.txt", O\_WRONLY | O\_CREAT, 0644);

write(file, "Hello, MINI OS!\n", 17);

close(file);

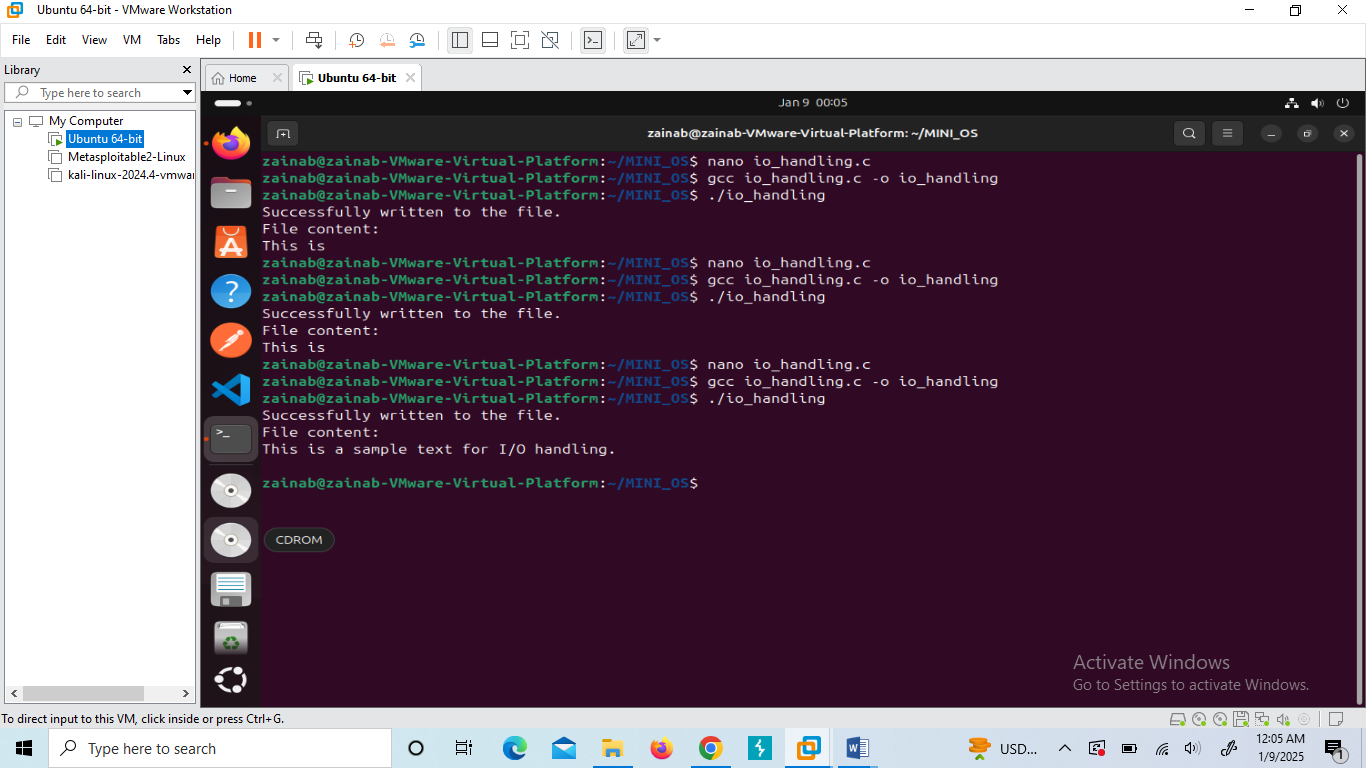
file = open("output.txt", O\_RDONLY);

read(file, buffer, sizeof(buffer));

printf("File content: %s\n", buffer);

close(file);

return 0;

}  
  


**Step 3: Integration of Features**

To make the project user-friendly, all features were integrated into a **menu-driven interface**. Users can select options to execute different functionalities.

int main() {

int choice;

while (1) {

printf("\n1. Process Management\n2. Memory Management\n3. IPC\n4. Synchronization\n5. I/O Handling\n6. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1: process\_management(); break;

case 2: memory\_demo(); break;

case 3: ipc\_function(); break;

case 4: process\_synchronization(); break;

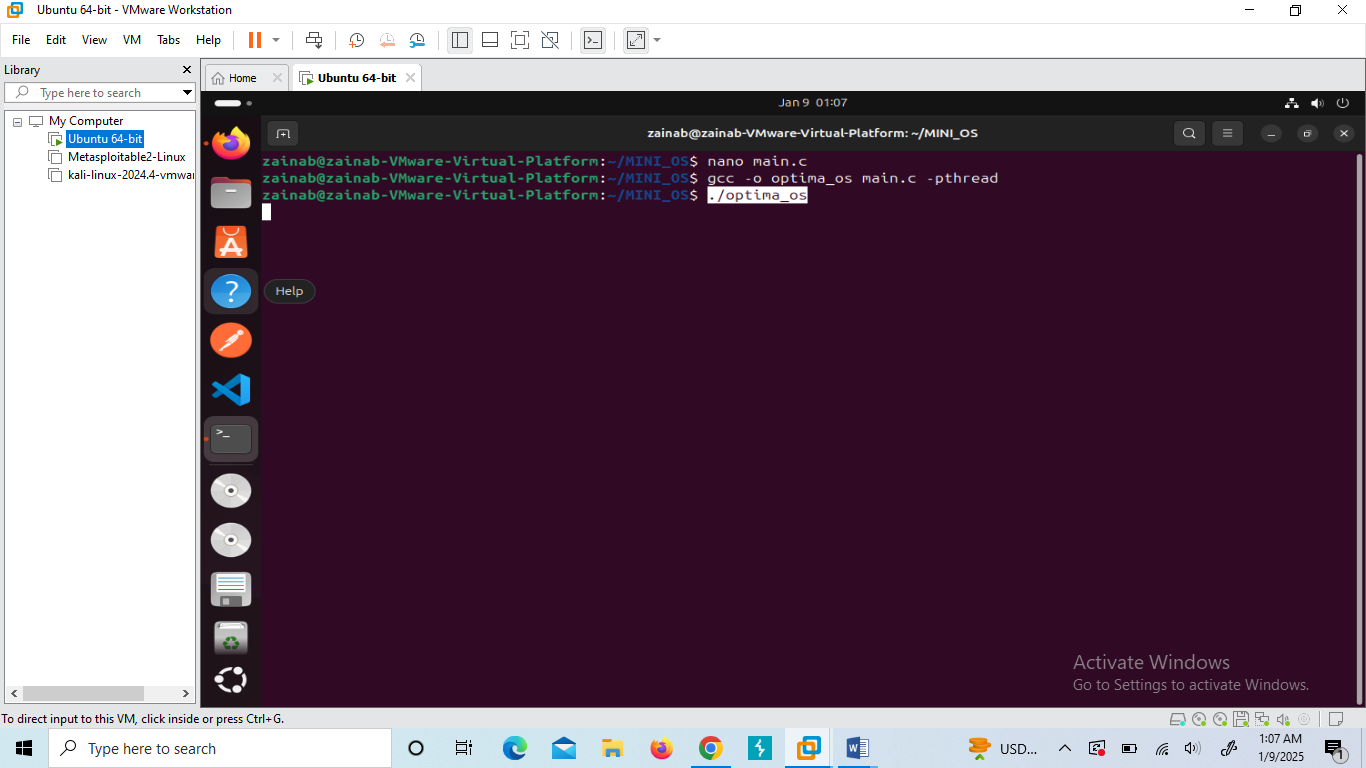
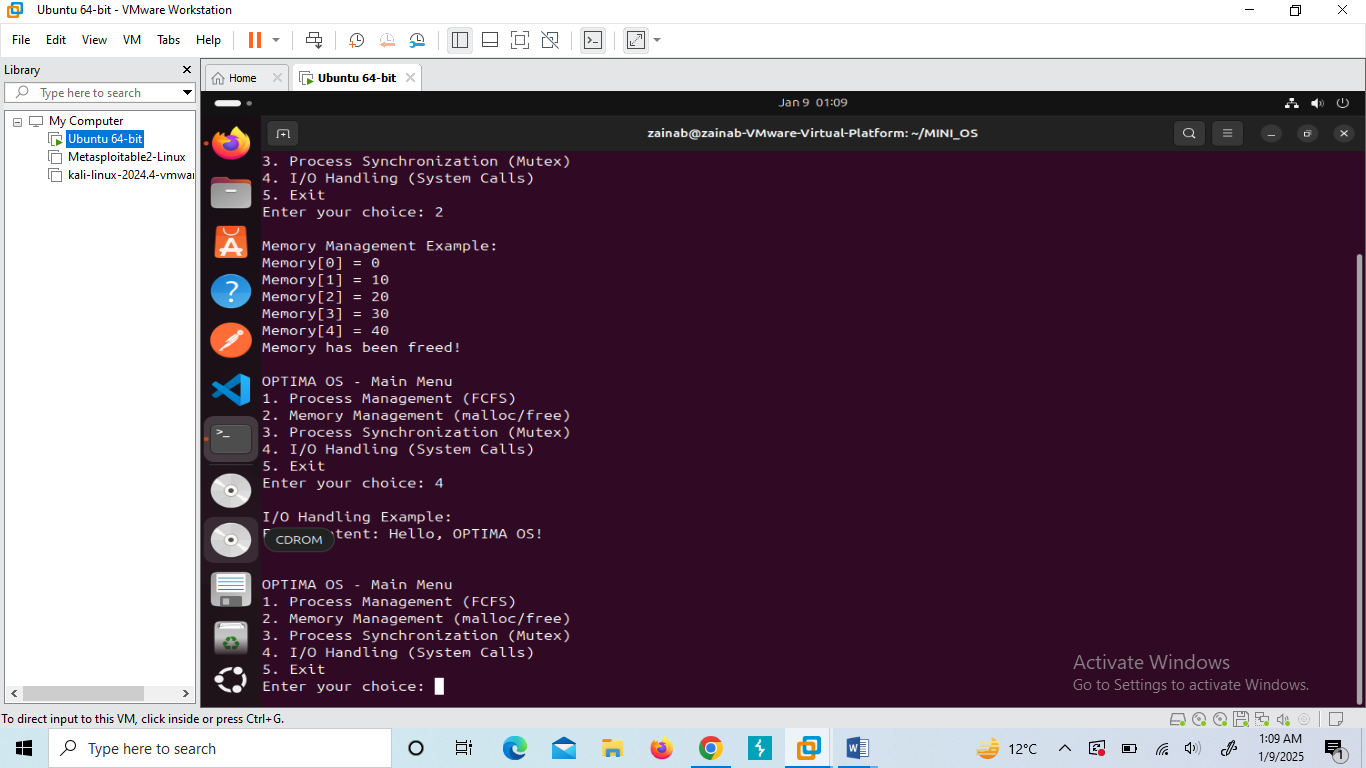
case 5: io\_handling(); break;

case 6: exit(0);

default: printf("Invalid choice!\n");

}

}

}  
  
  


**Conclusion**

This project provided hands-on experience with essential **operating system concepts**, including **process scheduling, memory allocation, inter-process communication, synchronization, and file handling**. The implementation in **C on Linux** strengthened my understanding of **system-level programming**. Future enhancements could include **advanced scheduling algorithms** and **improved memory management techniques**.

**Final Thoughts**

This project was a great learning experience, allowing me to dive deeper into **OS fundamentals**. Excited to work on more advanced OS concepts in the future! 🚀