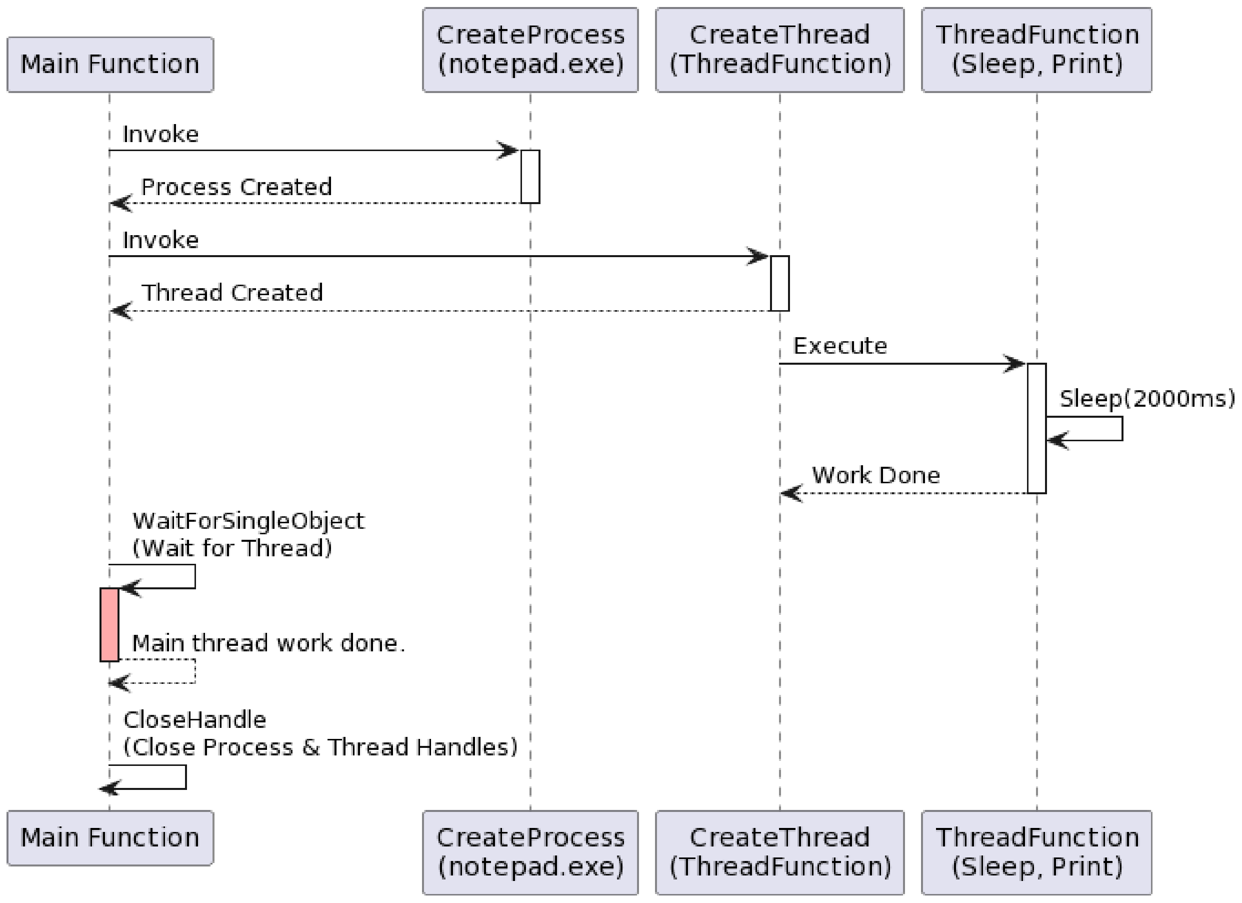
Operating System Project:

Introduction

This project embarked on developing a comprehensive operating system (OS) simulation, focusing on key areas: multi-process and thread management, inter-process communication (IPC), and parallel text processing. The simulation is designed to provide an immersive, hands-on experience in managing OS-level processes and threads, implementing IPC mechanisms, and leveraging parallel computing for efficient text file processing.

Multi Process and Threads:


Main Function
CreateProcess (notepad.exe)
CreateThread (ThreadFunction)
ThreadFunction (Sleep, Print)
Invoke
Process Created
Invoke
Thread Created
Execute
Sleep(2000ms)
Work Done
WaitForSingleObject (Wait for Thread)
Main thread work done.
CloseHandle (Close Process & Thread Handles)
Main Function
CreateProcess (notepad.exe)
CreateThread (ThreadFunction)
ThreadFunction (Sleep, Print)

Objective: To simulate an OS's capability to manage multiple processes and threads, offering functionalities like creation, suspension, resumption, and termination.

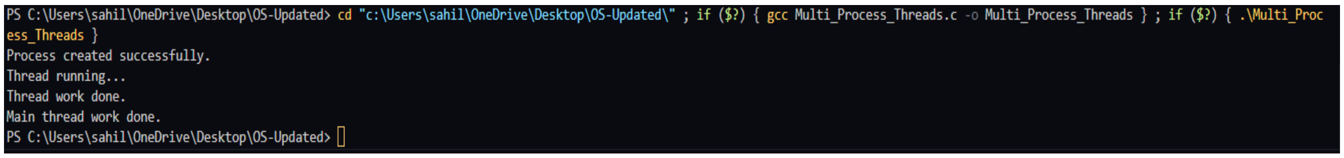
Implementation:

- It includes a Windows-based multi-threaded Code.
- Launched a primary process (notepad.exe) for demonstration.

- Initiated a separate thread to simulate a task.
- Incorporated thread completion management and clean resource deallocation.

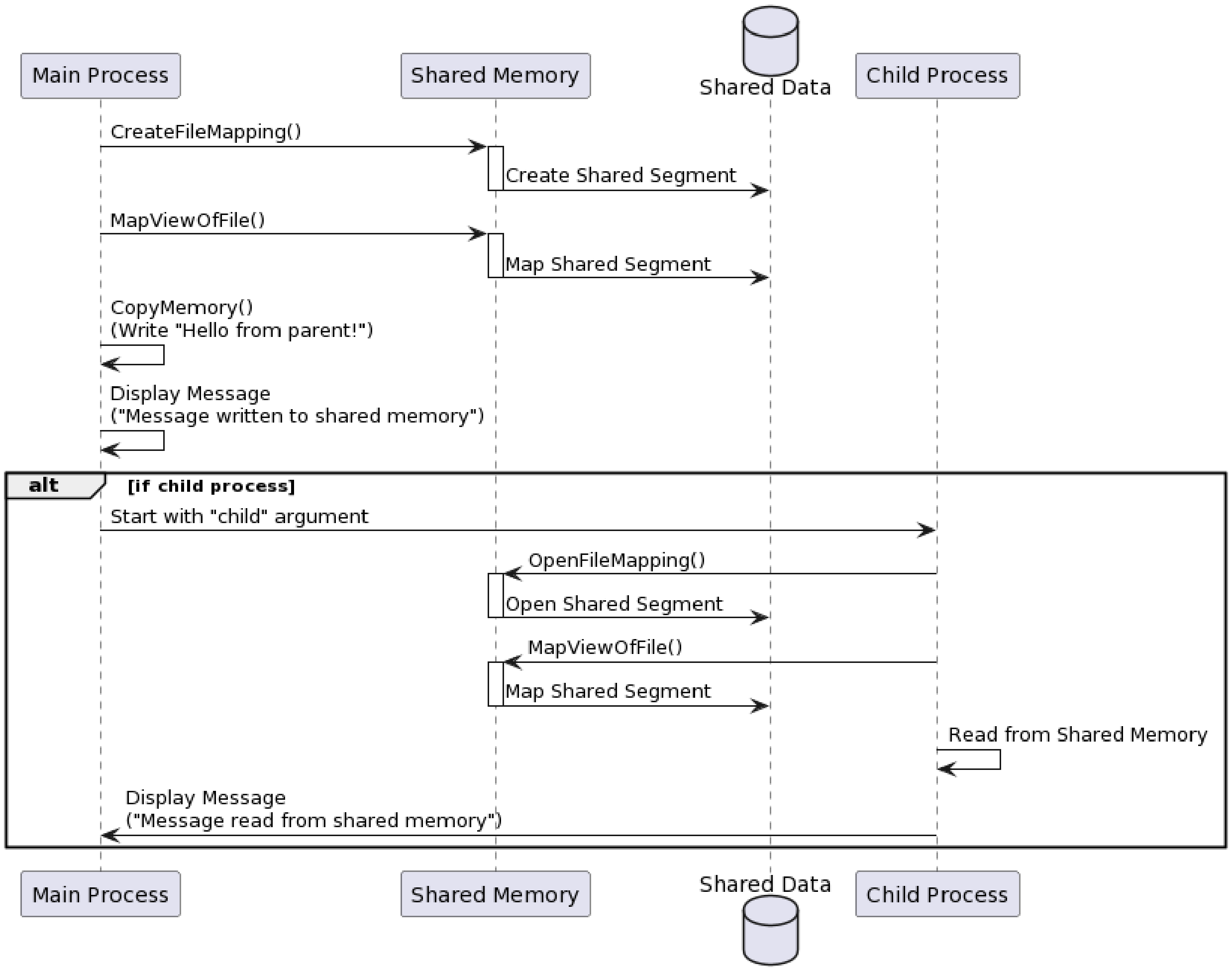
Structure:

- Utilized modular functions for process and thread creation (CreateProcess and CreateThread).
- Implemented synchronization with WaitForSingleObject for orderly execution.
- Integrated error handling via GetLastError for robust operation.
- Encapsulated functionality within a main function for simplicity and scalability.



```
PS C:\Users\sahil\OneDrive\Desktop\OS-Updated> cd "c:\Users\sahil\OneDrive\Desktop\OS-Updated\" ; if ($?) { gcc Multi_Process_Threads.c -o Multi_Process_Threads } ; if ($?) { .\Multi_Proc
ess_Threads }
Process created successfully.
Thread running...
Thread work done.
Main thread work done.
PS C:\Users\sahil\OneDrive\Desktop\OS-Updated>
```

Inter Process Communication (IPC):

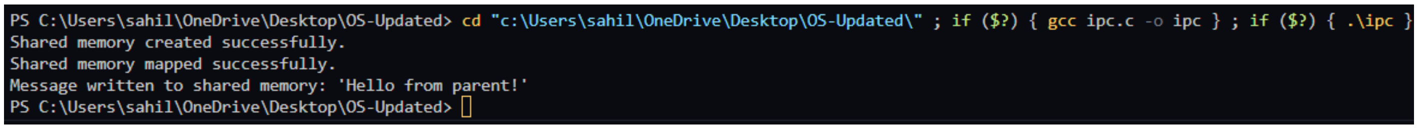


Objective: The code demonstrates the use of shared memory, an essential mechanism for enabling processes to efficiently share data.

Implementation: The **ipc.c** file provides an example of both shared memory and message passing mechanisms. It demonstrates creating a shared memory segment for inter-process data sharing and using message queues for process-to-process messaging.

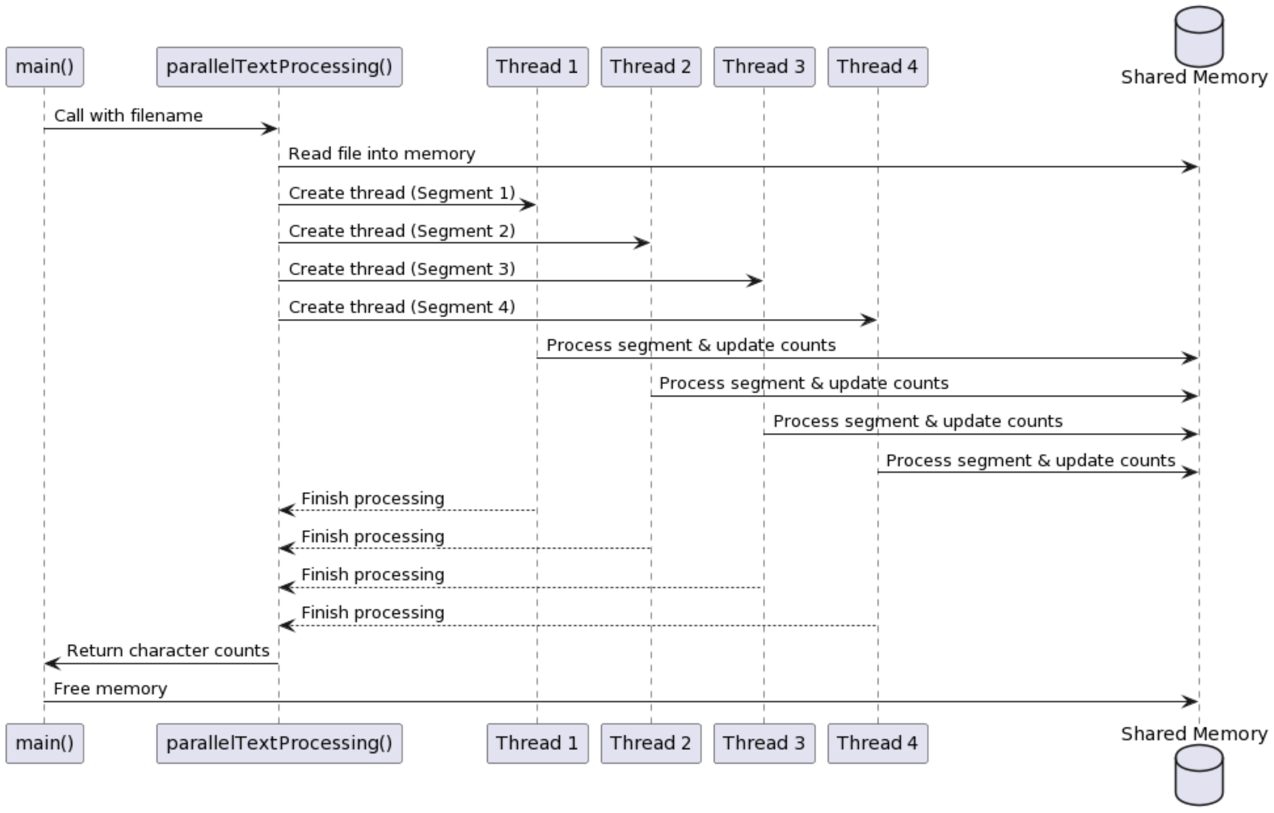
Structure:

- Initializes and maps a specified memory segment into the process's address space for inter-process access.
- Illustrates inter-process communication through direct writing to and reading from shared memory.



```
PS C:\Users\sahil\OneDrive\Desktop\OS-Updated> cd "c:\Users\sahil\OneDrive\Desktop\OS-Updated\" ; if ($?) { gcc ipc.c -o ipc } ; if ($?) { .\ipc }
Shared memory created successfully.
Shared memory mapped successfully.
Message written to shared memory: 'Hello from parent!'
PS C:\Users\sahil\OneDrive\Desktop\OS-Updated> 
```

Parallel Text Processing:

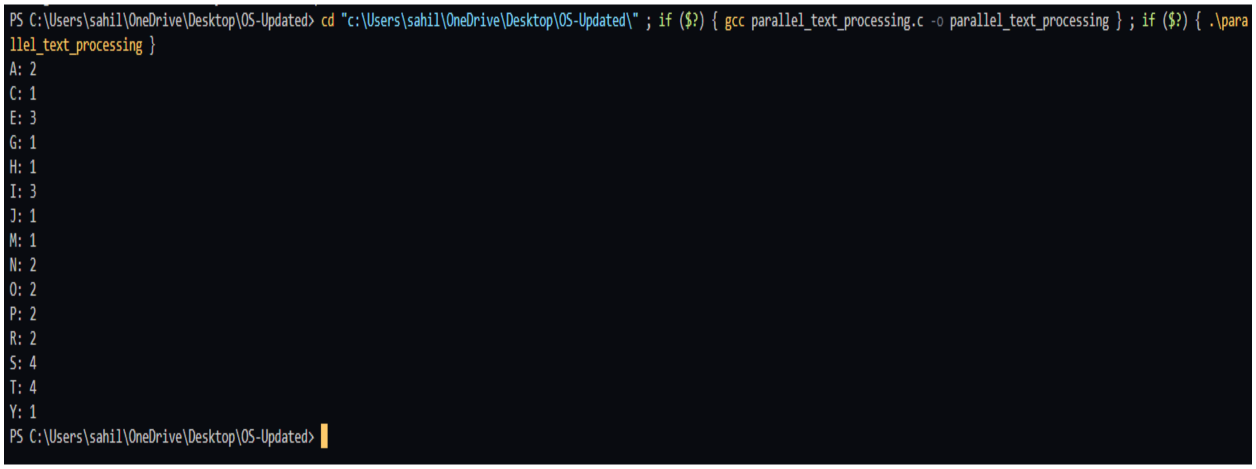

main()
parallelTextProcessing()
Thread 1
Thread 2
Thread 3
Thread 4
Shared Memory
Call with filename
Read file into memory
Create thread (Segment 1)
Create thread (Segment 2)
Create thread (Segment 3)
Create thread (Segment 4)
Process segment & update counts
Process segment & update counts
Process segment & update counts
Process segment & update counts
Finish processing
Finish processing
Finish processing
Finish processing
Return character counts
Free memory

Objective: To implement a system for processing large text files in parallel, demonstrating the effectiveness of parallel computing.

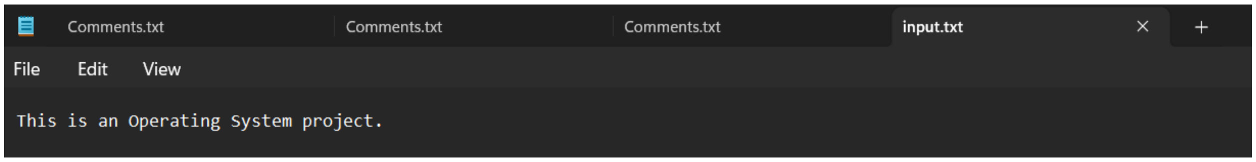
Implementation: The **parallel_text_processing.c** file showcases dividing a large text file into segments processed by individual threads, each converting characters to uppercase and counting character occurrences.

Structure:

- The text file is read into memory, divided into segments.
- Threads are spawned to process each segment in parallel.
- Character occurrences are counted and aggregated.



```
PS C:\Users\sahil\OneDrive\Desktop\OS-Updated> cd "c:\Users\sahil\OneDrive\Desktop\OS-Updated\" ; if ($?) { gcc parallel_text_processing.c -o parallel_text_processing } ; if ($?) { .\para
llel_text_processing }
A: 2
C: 1
E: 3
G: 1
H: 1
I: 3
J: 1
M: 1
N: 2
O: 2
P: 2
R: 2
S: 4
T: 4
Y: 1
PS C:\Users\sahil\OneDrive\Desktop\OS-Updated>
```



```
This is an Operating System project.
```

Verification and Observations

The functionalities were verified through execution, focusing on the correct implementation of process and thread management, IPC mechanisms, and parallel text processing efficiency. Each component behaved as designed, reflecting the core principles intended for demonstration.

Challenges:

- Ensuring accurate synchronization among processes and threads.
- Measuring and comparing IPC mechanism performance under various conditions.

Insights:

- Proper thread and process synchronization are crucial for reliable IPC and parallel processing.
- Performance of IPC mechanisms vary with the data size and operation frequency, highlighting the importance of choosing the right IPC method based on specific needs.

Conclusion

This project successfully simulates crucial aspects of an operating system, offering valuable insights into **process and thread management, IPC, and parallel computing**. While challenges were encountered, particularly in synchronization and performance evaluation, the project provides a solid foundation for understanding and exploring operating system functionalities further. Future work could focus on enhancing user interaction, dynamic thread management, and exploring additional IPC mechanisms.