ECS 408/608: Operating System

Assignment: Multithreading

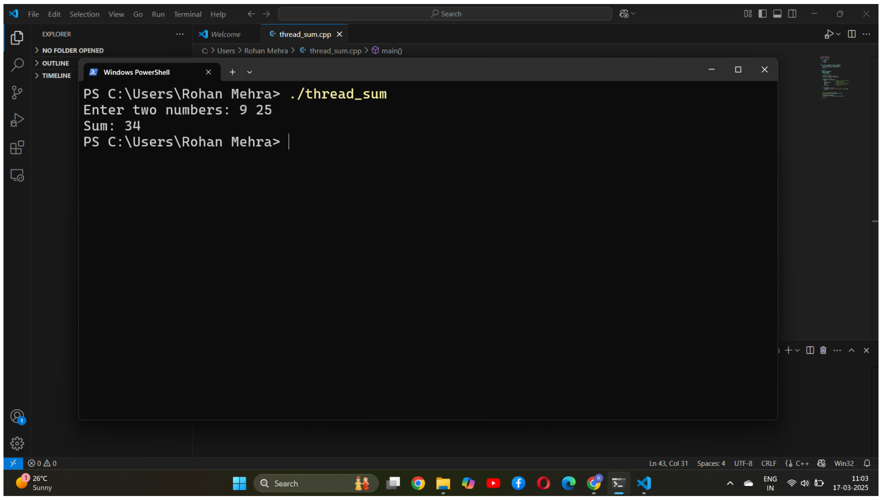
March 17, 2025

Rohan Mehra
(Roll No: 21224)

1. Write a program to create a thread T1. The main process passes two numbers on T1. In result, T1 returns the sum to the parent process for printing.

Answer: To solve this problem, we implemented multithreading using **Windows Threads**. The main process creates a thread $T1$ using the `CreateThread` function. Two numbers are passed to $T1$ via a shared data structure (`struct ThreadData`). $T1$ computes their sum and stores the result in the shared structure. The main process waits for $T1$ to finish using `WaitForSingleObject`, retrieves the result, and prints it.

Below is the scrrenshot of the implementation in C++ using Windows Threads. For the full code, refer to: GitHub Repository.



File Edit Selection View Go Run Terminal Help
Search
EXPLORER
NO FOLDER OPENED
OUTLINE
TIMELINE
Welcome
thread_sum.cpp
C: > Users > Rohan Mehra > thread_sum.cpp > main()
Windows PowerShell
Ln 43, Col 31 Spaces: 4 UTF-8 CRLF C++ Win32
26°C Sunny
Search
ENG IN
11:03
17-03-2025

```
PS C:\Users\Rohan Mehra> ./thread_sum
Enter two numbers: 9 25
Sum: 34
PS C:\Users\Rohan Mehra>
```

Sample Output:

```
Enter two numbers: 5 10
Sum: 15
```

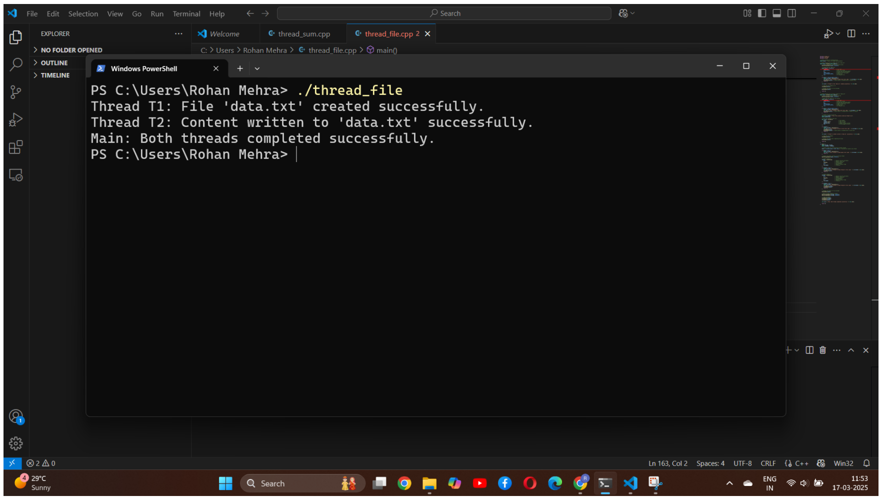
2. Create a program that spawns two threads, T1 and T2. Thread T1 is responsible for generating a file named “data.txt,” and T2 is tasked with writing specific content to the “data.txt” file.
   Hints: You can use an open() system call to create a file.
   read() to collect data to be written.
   write() to put the content in the file.

Answer:

To solve this problem, we implemented multithreading using **Windows Threads**. The solution involves the following steps:

1. Thread $T1$ creates the file ”data.txt” using the `CreateFile` function.
2. Thread $T2$ writes specific content to the file using the `WriteFile` function.
3. A mutex is used to synchronize the two threads, ensuring $T2$ waits for $T1$ to finish creating the file.

Below is the implementation in C++ using Windows Threads. For the full code, refer to: GitHub Repository.



```
PS C:\Users\Rohan Mehra> ./thread_file
Thread T1: File 'data.txt' created successfully.
Thread T2: Content written to 'data.txt' successfully.
Main: Both threads completed successfully.
PS C:\Users\Rohan Mehra>
```

Sample Output:

```
Thread T1: File ’data.txt’ created successfully.
Thread T2: Content written to ’data.txt’ successfully.
Main: Both threads completed successfully.
```

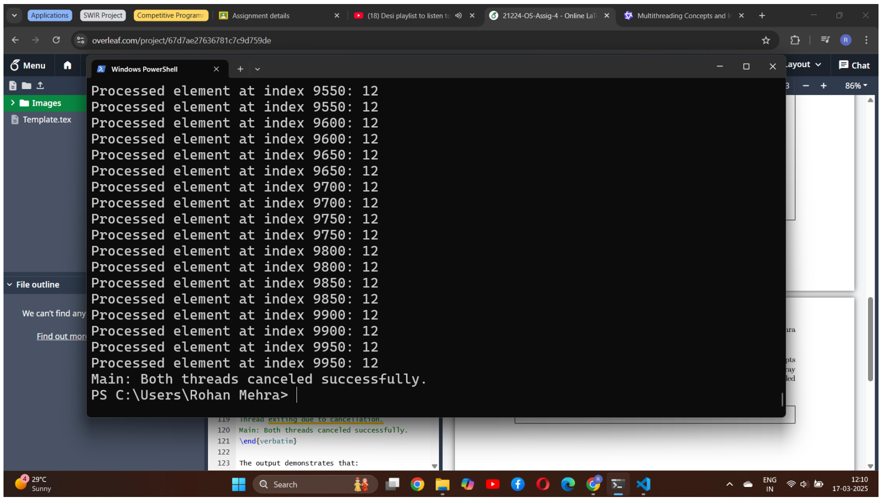
3. Create a program that spawns two threads, T1 and T2. Each of the thread accepts an 10,000 element array as an input, Thread T1 and T2 prints each and every array element by adding an integer 2 and multiplied by 4. Both the threads are canceled after 1 second from the main thread after spawning T1 and T2.

Answer:

To solve this problem, we implemented multithreading using **Windows Threads**. The solution involves the following steps:

1. Two threads, $T1$ and $T2$, are created using the `CreateThread` function.
2. Each thread processes a 10,000-element array by applying the transformation $(x + 2) \times 4$ to each element.
3. A global flag (`volatile bool`) is used to signal the threads to stop execution after 1 second.
4. The main thread cancels both threads after 1 second using a cooperative cancellation approach.

Below is the implementation in C++ using Windows Threads. For the full code, refer to: GitHub Repository.



```
Processed element at index 9550: 12
Processed element at index 9550: 12
Processed element at index 9600: 12
Processed element at index 9600: 12
Processed element at index 9650: 12
Processed element at index 9650: 12
Processed element at index 9700: 12
Processed element at index 9700: 12
Processed element at index 9750: 12
Processed element at index 9750: 12
Processed element at index 9800: 12
Processed element at index 9800: 12
Processed element at index 9850: 12
Processed element at index 9850: 12
Processed element at index 9900: 12
Processed element at index 9900: 12
Processed element at index 9950: 12
Processed element at index 9950: 12
Main: Both threads canceled successfully.
PS C:\Users\Rohan Mehra>
```

```
Thread exiting due to cancellation.
Main: Both threads canceled successfully.
\end{verbatim}

The output demonstrates that:
```

Sample Output:

```
Processed element at index 0: 12
Processed element at index 50: 12
Processed element at index 100: 12
...
Thread exiting due to cancellation.
Thread exiting due to cancellation.
Main: Both threads canceled successfully.
```

The output demonstrates that:

- Both threads process elements concurrently.
- The threads are canceled after 1 second.
- The main thread confirms successful cancellation.