Social Media Parallel Data Fetcher

1. Multithreading

1. Create a thread for each GUI so that every social media window operates independently of the others.

```
foreach (var platform in selectedPlatforms)
{
    Thread thread = new Thread(() =>
    {
        Form platformForm = null;
        switch (platform)
        {
            case "Facebook":
                platformForm = new Facebook(fbAccessKey);
                break;

            case "Messenger":
                platformForm = new Messenger(fbAccessKey);
                break;

            case "Instagram":
                platformForm = new Instagram(instaAccessKey);
                break;

            case "Comments":
                platformForm = new Comments(fbAccessKey);
                break;
        }

        if (platformForm != null)
        {
            platformForms.Add(platformForm); // Add form to the list
            Application.Run(platformForm);
        }
    });

    thread.SetApartmentState(ApartmentState.STA);
    platformThreads.Add(thread); // Add thread to the list
    thread.Start();
}
```

2. For each GUI, create a thread to call the fetch data function.
This ensures that the GUI's operation is independent of data fetching. Previously, GUI forms would not open until the data was fetched from the API, causing delays in displaying the forms. With this approach, data fetching happens on a background thread, keeping the UI responsive.

```
List<object> commentsData = await Task.Run(() => DataFetcher.FetchAllComments(accessKey));
```

3. The fetch function is called as a task using async and await functions, enabling parallelism instead of sequential execution. This asynchronous approach ensures that data loading does not freeze the UI.

```
private async Task FetchDataAsync()
```

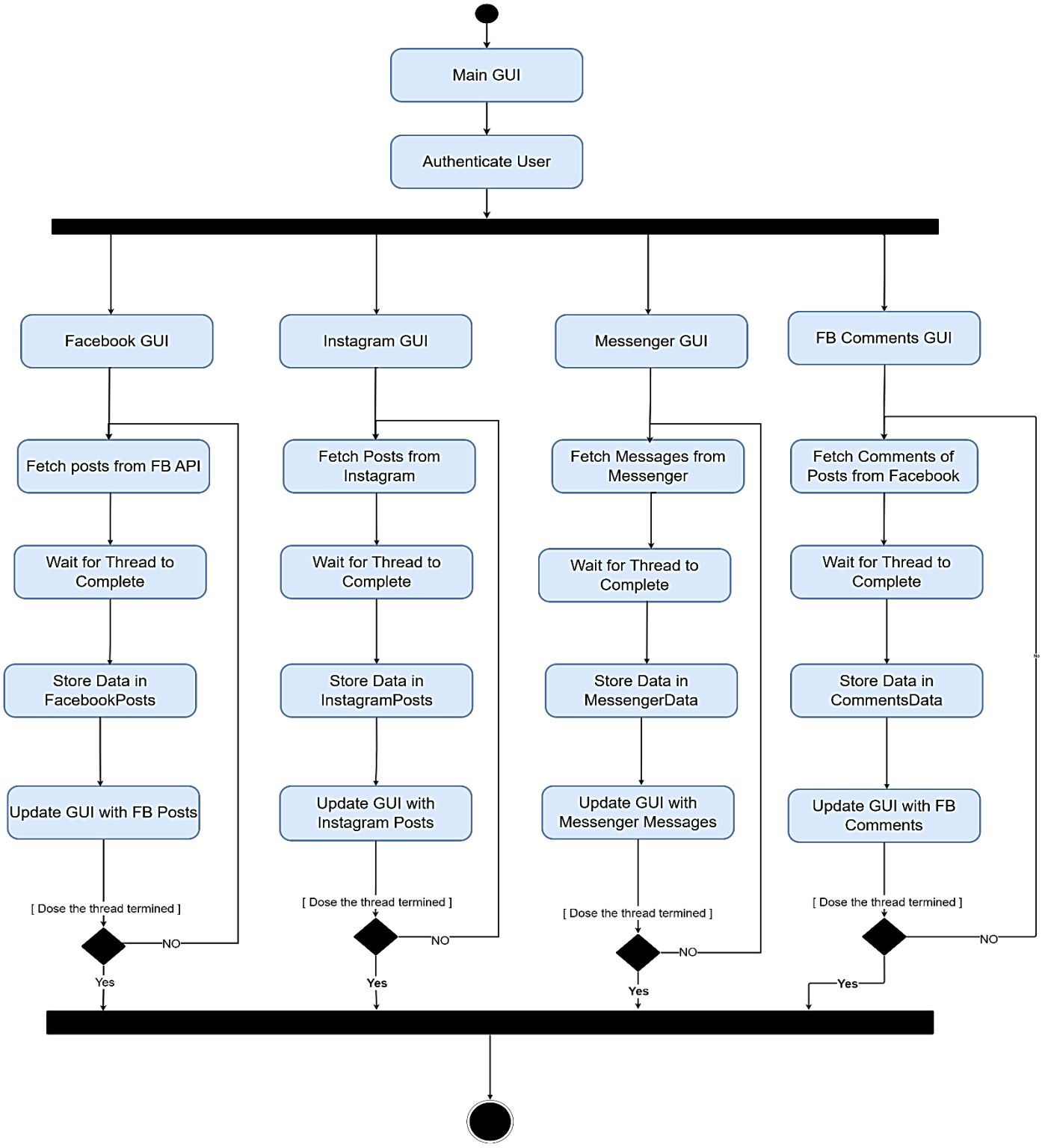
Were issues like race conditions and deadlocks avoided?

The Mutex is used to ensure that only one thread accesses the FetchDataFromApi method at a time, preventing race conditions when multiple threads attempt to fetch data simultaneously. This ensures thread-safe access to shared resources and avoids potential conflicts or data corruption.

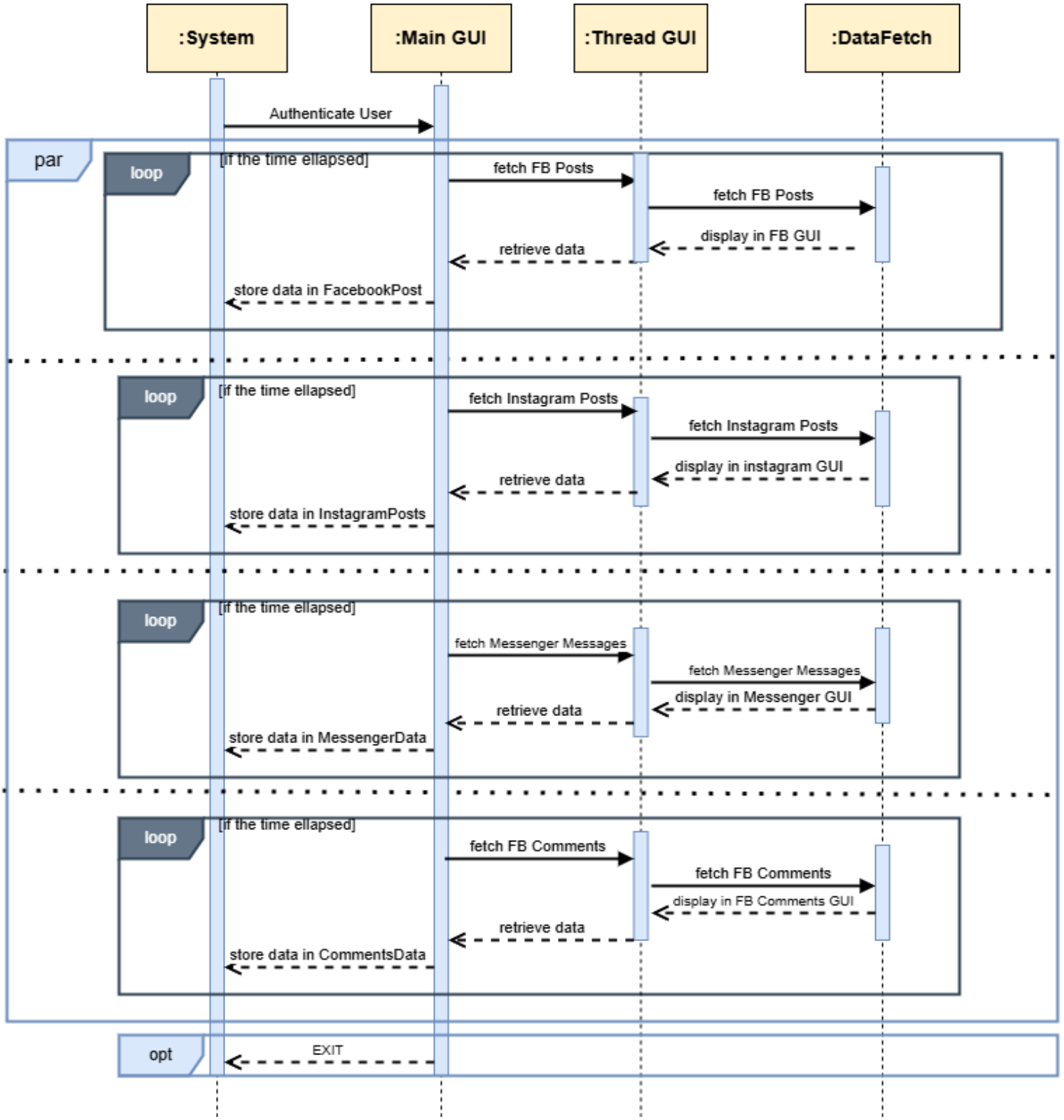
```
private static readonly Mutex mutex = new Mutex(); // Create a Mutex instance

private static string FetchDataFromApi(string url)
{
    mutex.WaitOne(); // Wait until it is safe to enter

    try
    {
        using (HttpClient client = new HttpClient())
        {
            try
            {
                HttpResponseMessage response = client.GetAsync(url).Result;

                // If the response is not successful, show error and return null
                if (!response.IsSuccessStatusCode)
                {
                    Console.WriteLine($"Failed to fetch data from URL: {url}");
                    return null;
                }
                // Return the content of the response as a string
                return response.Content.ReadAsStringAsync().Result;
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error fetching data: {ex.Message}");
                return null;
            }
        }
    }
    finally
    {
        mutex.ReleaseMutex(); // Release the mutex
    }
}
```

Diagrams

Activity Diagram

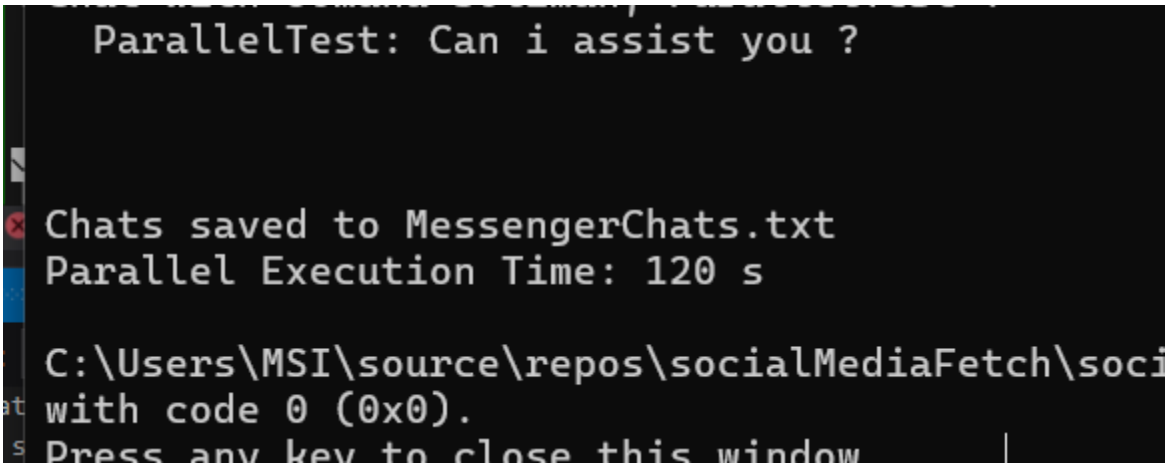


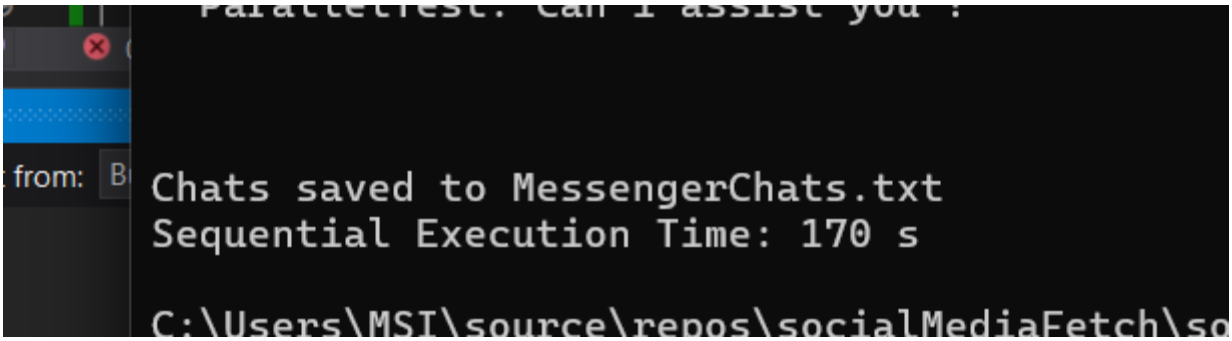
Sequence Diagram



How much faster or more efficient is the system with multithreading compared to a single-threaded version?

The implementation of multithreading significantly improves the efficiency of the system by allowing multiple tasks, such as data fetching, to run simultaneously. Unlike the single-threaded version, where tasks are processed sequentially, multithreading reduces delays and enhances overall performance. The time difference, as shown in the attached screenshot, highlights the performance boost achieved through parallel execution, making the system faster and more efficient.





Steps for Social Media Platform Integration and Data Fetching:

1. **Platform Selection:**
   Users can select specific social media platforms to integrate with, such as:
   - **Facebook**: Posts, comments, and Messenger messages.
   - **Instagram**: Posts.
2. **Access Token Setup**:
   - Obtain an **access token** for Facebook and Messenger.
   - Obtain a separate **access token** for Instagram.
3. **Account and Page Preparation**:
   - Create a **Facebook page**.
   - Link a **professional Instagram account** to the Facebook page.
4. **Data Fetching Process**:
   - Utilize **Meta's Graph API endpoints** with the required parameters to retrieve content from the selected platforms.
   - Provide the necessary access tokens for API access.
5. **Initiate Data Fetching**:
   - Click the **"Start"** button after selecting the platforms and entering the tokens to begin the data fetching process.
6. **Data Presentation**:
   - A **graphical user interface (GUI)** displays the fetched data.
   - The data is organized and presented in a user-friendly format, ensuring seamless integration and efficient visualization of content from multiple channels.