C# Final Work

File content indexing using agents and a master process

PLEASE READ WHAT NEED TO SUBMIT BELOW

Objective

Create a distributed system consisting of three separate C# console applications (executables):
Two **Agent (Scanner)** programs responsible for reading and analyzing .txt files.
One **Master** program is responsible for collecting and merging data from both agents via **named pipes**.

Tasks

1. Program responsibilities

Agents (Scanner A and Scanner B):

1. Each agent receives a directory path containing .txt files.
2. It reads the content of each file, indexes words (e.g., filename; count).
3. It sends this information to the **Master** process using a **named pipe** (one pipe per agent).

Master:

1. Waits for connections from both agents via named pipes.
2. Receives and processes the indexed word data.
3. Aggregates the data and displays the final result (filenames, word, count of word).

2. Multithreading Requirements

All applications must use **multiple threads**:

Example: one thread for reading files, another for sending data.
The master process may use separate threads to handle each pipe concurrently.

3. CPU Core Affinity

Each program (Scanner A, Scanner B, Master) must be manually or programmatically configured to run on **a separate CPU core** using ProcessorAffinity.

4. Communication Method

Use **Named Pipes** for inter-process communication:

Each agent connects to the master through a unique named pipe.
Use NamedPipeServerStream (in Master) and NamedPipeClientStream (in Agents).

Input

Agents: a directory path containing .txt files.
Master: the names of the pipes it should listen on (e.g., "agent1", "agent2").

Output
The master process prints a consolidated word index:

fileA.txt:example:3
fileB.txt:home:20

Technology Guidelines
.NET or later
Console Applications or Winforms or WPF
Use of:

- NamedPipeServerStream / NamedPipeClientStream
- Thread or Task
- Processor.ProcessorAffinity

Submit

1. **Upload Code to GitHub**:
   - To upload your project to a **!!!public GitHub repository with each task (or different day, stage or etc.) being committed as a separate version!!!**
   - Each task should have a clear version history (readme.md) and comments to explain what was done at each stage.
   - **Add the GitHUB repository URL to the report's title page**
2. **Upload Test Report (PDF) to Moodle**:
   - Create a UML class diagram for your application
   - After completing the task, create a detailed test report documenting the following:
     - What you have implemented.
     - The functionality of each task.
     - Any challenges faced during the implementation.
     - Screenshots of running the program with example output.
3. **Upload to Moodle final version of program with compiled (debug) folder. Format (*.zip).**