VILNIUS UNIVERSITY
ŠIAULIAI ACADEMY

BACHELOR PROGRAMME SOFTWARE ENGINEERING

Object-Oriented Programming

C# Final Work (File Indexer System)

Student: Eliyas Ahmed Sadhin

ID: 2431349

Teacher: Prof. Dr. Donatas Dervinis

Date: 7th June, 2025

File Indexer System — Test Report

What Was Implemented

This project consists of three C# console applications working together as a distributed system:

- **ScannerA** and **ScannerB**: Each agent scans a specified directory for .txt files, reads their contents, indexes the frequency of each word in the files, and sends this data to the Master process via named pipes (agent1 and agent2 respectively).
- **Master**: Listens on two named pipes concurrently for incoming data from the two agents, aggregates the word counts received, and displays the consolidated index showing the count of each word per file.

Multithreading and Named Pipes Usage

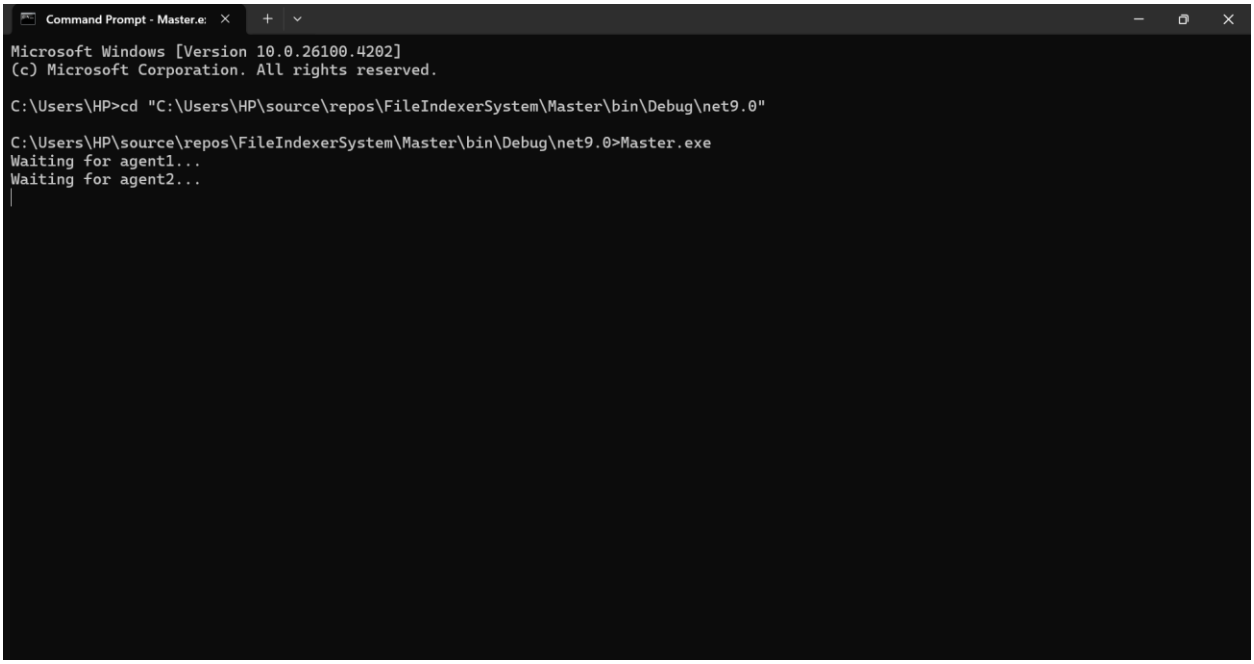
- **Multithreading** was used extensively in all applications to ensure responsiveness and concurrency:
  - In each **Scanner** agent, separate threads handle file reading and data sending operations to the Master process.
  - The **Master** process uses multiple threads to listen to each named pipe simultaneously and process incoming data concurrently.
- **Named Pipes** (NamedPipeClientStream and NamedPipeServerStream) provide the inter-process communication channel between the agents and the master. Each agent connects to the Master via a unique named pipe (agent1 or agent2).

Challenges Faced

- **Synchronizing threads and avoiding race conditions:**
  Ensuring that multiple threads in the master process handle incoming pipe data correctly without overwriting or losing information required careful use of thread synchronization and safe data structures.
- **Setting processor affinity:**
  Configuring each program to run on separate CPU cores programmatically involved platform-specific API usage, which required additional debugging.

- **Handling large files and multiple files concurrently:**
  Efficiently reading and indexing multiple text files in parallel without blocking communication threads was an important challenge.

Screenshots



Running Master

```
Command Prompt

Microsoft Windows [Version 10.0.26100.4202]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>cd "C:\Users\HP\source\repos\FileIndexerSystem\ScannerA\bin\Debug\net9.0"

C:\Users\HP\source\repos\FileIndexerSystem\ScannerA\bin\Debug\net9.0>ScannerA.exe "C:\ScannerAData"
ScannerA completed.

C:\Users\HP\source\repos\FileIndexerSystem\ScannerA\bin\Debug\net9.0>
```

Running ScannerA

```
Command Prompt

Microsoft Windows [Version 10.0.26100.4202]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>cd "C:\Users\HP\source\repos\FileIndexerSystem\ScannerB\bin\Debug\net9.0"

C:\Users\HP\source\repos\FileIndexerSystem\ScannerB\bin\Debug\net9.0>ScannerB.exe "C:\ScannerBData"
ScannerB completed.

C:\Users\HP\source\repos\FileIndexerSystem\ScannerB\bin\Debug\net9.0>
```

Running ScannerB

```
Command Prompt

Microsoft Windows [Version 10.0.26100.4202]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>cd "C:\Users\HP\source\repos\FileIndexerSystem\Master\bin\Debug\net9.0"

C:\Users\HP\source\repos\FileIndexerSystem\Master\bin\Debug\net9.0>Master.exe
Waiting for agent1...
Waiting for agent2...
agent1 connected.
agent2 connected.

Aggregated Word Counts:
b1.txt:again:1
b1.txt:b:1
b1.txt:hello:2
b1.txt:scanner:1
b1.txt:from:1
a1.txt:hello:1
a1.txt:scanner:1
a1.txt:this:1
a1.txt:world:1
a1.txt:is:1
a1.txt:a:1

Master completed.

C:\Users\HP\source\repos\FileIndexerSystem\Master\bin\Debug\net9.0>
```

Final Output from Master

Additional Notes

- The system successfully demonstrates distributed file content indexing using named pipes for communication.
- The use of multithreading improves efficiency by allowing simultaneous file scanning and communication.
- This project solidified understanding of inter-process communication, thread management, and processor affinity in C#.
