Project Title: Parallel File Search Engine

Due to 30th November2023

يعد الغش مخالفة أكاديمية وفقا للوائح والقوانين المعمول بها في جامعة قطر، وقد تصل عقوبة هذه المخالفة في بعض الحالات إلى الفصل النهائي من الجامعة وعلى الطلاب تجنب القيام أو المشاركة في أي عمل يخالف ميثاق النزاهة الاكاديمية وإجراءات الاختبارات المعمول بها بجامعة قطر.

Cheating is an academic violation according to Qatar University rules and regulations, and in some cases, it may result in final dismissal from the University. Students should not under any circumstances commit or participate in any cheating attempt or any act that violates the student code of conduct.

It is the right of the instructor to test the student's undersetting of the project in any way during the demo and discussion session. So, a project that is 100% working might be may be graded (-100%) due to student not being able to explain a functionality they have implemented.

Description:

The objective of this project is to create a robust and efficient file search engine that uses parallel processing to search for a specified keyword across multiple text files concurrently. Upon successful completion of the Parallel File Search Engine project, you are expected to have a solid understanding and practical implementation skills of several advanced computing concepts within Java. These include multi-threading, socket programming, scheduling, and synchronization in Java.

Server:

Upon running the server instance, the server in the Parallel File Search Engine project should perform the following steps:

1. **Identify the Operating System (OS):** The server should first detect the operating system on which it is running. This identification is crucial as it may influence how file paths are handled and how certain system-specific operations are executed.
2. **Determine the Number of Processor Cores:** The server should ascertain the number of available processor cores. This information is essential for optimizing parallel processing, as it allows the server to tailor its threading strategy to the capabilities of the hardware.
3. **Create Threads Based on Core Count:** Leveraging the information about the number of cores, the server should create an equal number of threads. This ensures that the system's processing capabilities are fully utilized, with each core handling a separate thread for balanced load distribution.

Server Scenario: Develop a simple and intuitive user interface (GUI or command-line) that allows users to input search queries and displays the search results.

- The server will start by providing a small menu as shown below and asking the user to choose one of the following options:

> **Welcome to Parallel File Search Server Load Distributor!**
>
> **Choose one of the following options:**
>
> 1. **Equal Distribution.**
> 2. **Round Robin Distribution**
> 3. **Quit**

- The server should divide the task of searching across multiple threads, with each thread being responsible for searching through a subset of the files. The server should implement a strategy for distributing the workload across the threads. Two options are shown:
    - **Option 1 - Equal Distribution:** Here, each thread is assigned an approximately equal portion of the total workload. This approach is straightforward and effective when the workload can be evenly divided.
    - **Option 2 - Round Robin Distribution:** In this method, tasks are distributed to threads in a cyclic manner. As soon as a thread completes its task, it is assigned the next available task. This approach can be more dynamic and efficient, especially in handling varying task sizes.
- Once the server receives the user input, then server should start by indexing a specified directory containing text files.
- The server should work based on the selected searching option **[User input - option1 or option2]**.
- For each file, the server should read its content and store relevant information such as file name, file path, and word count.
    - **Note**: The application should be case-insensitive and search for the keyword regardless of its case in the text files.
- Once the searching is completely done by the server, server should present and send the output results to the client and display them in the following format:
    - **Files Searched by Each Thread:** Generate a detailed list for each thread, indicating the files it has processed. This list should include the index or identifier of each file, providing clear visibility into which thread handled which file.
    - **Identification of Keyword Matches:** Marks the files where the keyword has been found. This makes it easy to identify briefly which files contain the searched keyword.

- **Number of Occurrences:** report the number of times the keyword was found within that file.

Performance Measurements:

- **Total Searching Time:** The server should measure and report the total time taken for the entire search operation across all threads. This metric is crucial for evaluating the overall efficiency of the file search engine.
- **Individual Thread Timing:** In addition to the total time, the server should also record and display the time taken by each individual thread. This breakdown helps in understanding the load distribution and efficiency of each thread in the process.

Multi-Threading and synchronization:

By employing multi-threading, the application aims to improve search speed and efficiency, especially when dealing with many files. Make sure that your Server application employs the following.

- The search results are consistent and complete, even when multiple threads are accessing and updating them simultaneously.
- Implement synchronization mechanisms to prevent race conditions and data inconsistencies.

Client:

- The client must choose one of the options to start searching.
- After choosing the option, the client must enter the search path, and input a keyword that they wish to search for across all indexed files.
- Once the search is done, the client will receive all previous information, display it on screen and terminate the connection.

Important notes:

1. **Programming Language: Java**
2. **Do not use Java's built-in libraries for searching files. Instead, you are required to use the Linux commands that you've learned throughout the course. This approach is key to the learning objectives of the project.**

Instructions & Deliverables: Please read carefully

1- In MS-Word document submit the following:

- Cover page includes necessary details of the group members (Student Name, Student ID)

- In a table format, specify each group member tasks and the contribution percentage into the project.

2- Submit your scripts with the MS-Word document on the Blackboard in one zipped file no later than **the below due date.**

Due to 30th November 2023

Call the Zip file (StuentNAME1_StudentNAME2_StudentNAME3_StudentNAME4)

3- Copying and/or plagiarism (-100%) which includes:
Inappropriate interaction with any other student, outside agency, website, or software that generates assessment responses.
4- Java program should be error free (Errors) (-25%).
5- In case of late submission, (-10%) for each day of delay (Max 3 days delay until 12:00 pm).
6- A group of three to four students can work on the project.
7- Team members are required to meet regularly for discussion and workload distribution.
8- It is the right of the instructor to use any way of testing the student in the discussion and demo session, and according to that in some cases (100%) graded project may be down to (-100%) graded project.
9- Discussion & Demo 50%. + Student Work 50%.