golang

day #4 assignment

Assignment #1

Concurrent File Word Count:

- **Objective:** Develop a concurrent program in Go to count the occurrences of each word across multiple text files.

Imagine you have a large collection of text files containing articles. Your task is to develop a program that counts the occurrences of each word across all files concurrently using goroutines. Design a solution that reads each file concurrently, processes the text to extract words, and updates a shared data structure (like a map) that stores word counts. Ensure proper synchronization to prevent data races and output the total word count for each unique word at the end.

- **Instructions:**
    - Prompt the user to input the directory path containing the text files.
    - Read the list of files in the specified directory.
    - Create a map to store word counts (`map[string]int`).
    - Launch a goroutine to process each file concurrently.
    - In each goroutine, read the file contents and split it into words.
    - Update the word count map with the occurrences of each word.
    - Use synchronization mechanisms (e.g., `sync.Mutex`) to prevent data races while updating the shared map.
    - Wait for all goroutines to finish processing.
    - Output the word count map with total occurrences for each word.
- **Sample Input:**

```
Enter directory path containing text files: /path/to/text/files
```

- **Sample Output:**

```
Word Count:
"hello": 10
"world": 15
"example": 5
...
```

Assignment #2

Concurrent Download Manager

- **Objective:** Design a concurrent download manager in Go to fetch multiple files concurrently from a list of URLs.

Create a program that acts as a download manager capable of downloading multiple files concurrently from a list of URLs. Design a solution that divides the download tasks among goroutines, each responsible for fetching data from a specific URL. Implement functionality to track download progress, handle errors, and display completion status for each file. Utilize goroutines and channels for efficient concurrent processing and communication.

- **Instructions:**

    - Prompt the user to input a list of URLs to download.
    - Create a goroutine for each URL to handle the download.
    - Inside each goroutine, use `http.Get` to download the file content.
    - Save the downloaded content to a file asynchronously to avoid blocking.
    - Implement error handling to deal with failed downloads.
    - Track download progress and completion status.
    - Wait for all downloads to complete before exiting the program.
- **Sample Input:**

  ```
  Enter URLs to download (separated by spaces):
  https://example.com/file1.txt https://example.com/file2.txt
  ```

- **Sample Output:**

  ```
  Downloading file from https://example.com/file1.txt ... Done.
  Downloading file from https://example.com/file2.txt ... Done.
  ```

Assignment #3

Concurrent Web Server Log Analyzer

- **Objective:** Develop a concurrent web server log analyzer in Go to process log entries concurrently and extract useful statistics.

Build a program that analyzes web server logs concurrently to extract useful information such as the number of requests per endpoint and the most frequent user agents. Assume you have access to a log file containing entries in a common log format (CLF). Design a solution that reads and processes log entries concurrently using goroutines, with each goroutine responsible for analyzing a subset of log entries. Implement data structures to aggregate statistics and ensure proper synchronization to avoid race conditions.

- **Instructions:**

    - Prompt the user to input the path to the web server log file.
    - Read log entries from the log file.
    - Create a goroutine for each log entry to process concurrently.
    - Parse each log entry to extract useful information such as request endpoints and user agents.
    - Update shared data structures (e.g., maps) to aggregate statistics like request counts per endpoint and user agent frequencies.
    - Ensure proper synchronization to prevent data races while updating shared data structures.
    - Wait for all goroutines to finish processing.
    - Output the aggregated statistics.
- **Sample Input:**

  ```
  Enter path to web server log file: /path/to/logfile.log
  ```

- **Sample Output:**

  ```
  Request Count Per Endpoint:
  "/home": 1000
  "/about": 500
  "/contact": 300
  ...
  ```

You may get a sample web server log file from here:
https://raw.githubusercontent.com/elastic/examples/master/Common%20Data%20Formats/apache_logs/apache_logs

---