Go Concurrency Demo Project with Testing

Project Overview

This project demonstrates Go's concurrency features with **goroutines** and **channels** by simulating concurrent tasks. We also utilize Go's built-in **testing** capabilities to verify the function `simulateWork`, which performs simulated tasks with random delays.

Key Features

- **Concurrency**: The project leverages goroutines and channels to perform concurrent tasks.
- **Synchronization**: It uses `sync.WaitGroup` to manage concurrent goroutines.
- **Unit Testing**: Includes a test file to validate that `simulateWork` generates the expected output format.

Requirements

- **Go**: Ensure that Go is installed. You can download it from https://golang.org/dl/.

Check installation by running:
bash
Copy code
```
go version
```

  ○

Project Setup

1. Create a Project Directory

Open a terminal or PowerShell and create a new directory:

bash
Copy code
```
mkdir go_concurrency_demo
cd go_concurrency_demo
```

2. Initialize a Go Module

Initialize a new Go module for dependency management:

bash
Copy code
```
go mod init go_concurrency_demo
```

3. Create the `main.go` Program File

In the project directory, create a file named `main.go` with the following code:

go
Copy code
```
// main.go
package main

import (
    "fmt"
    "math/rand"
    "sync"
    "time"
)

// simulateWork performs a simulated work task with a random delay
func simulateWork(id int, wg *sync.WaitGroup, results chan<- string)
{
    defer wg.Done()
    time.Sleep(time.Duration(rand.Intn(1000)) * time.Millisecond)
    result := fmt.Sprintf("Worker %d finished work", id)
    results <- result
}

func main() {
    rand.Seed(time.Now().UnixNano())
    var wg sync.WaitGroup
    results := make(chan string, 5)

    // Start 5 concurrent workers
    for i := 1; i <= 5; i++ {
        wg.Add(1)
        go simulateWork(i, &wg, results)
    }
    // Close the results channel once all workers are done
    go func() {
        wg.Wait()
        close(results)
    }()

    // Read and print results as they come in
    for result := range results {
        fmt.Println(result)
    }
}
```

4. Create the Test File `main_test.go`

In the same directory, create a file named `main_test.go` with the following content:

go
Copy code

```
// main_test.go
package main

import (
    "testing"
    "sync"
)

// TestSimulateWork verifies that simulateWork sends the correct
result format to the channel.
func TestSimulateWork(t *testing.T) {
    var wg sync.WaitGroup
    results := make(chan string, 1) // Buffered channel to collect
the result

    wg.Add(1)
    go simulateWork(1, &wg, results)

    // Wait for simulateWork to complete
    go func() {
        wg.Wait()
        close(results)
    }()
    // Check the result
    result := <-results
    expected := "Worker 1 finished work"
    if result != expected {
        t.Errorf("Expected %s, but got %s", expected, result)
    }
}
```

Code Explanation

- **simulateWork Function**: Simulates work by sleeping for a random time and sending a message to a channel when complete.
- **Testing with `go test`**: The `main_test.go` file includes a function `TestSimulateWork` that verifies the output of `simulateWork`.
    - The function uses `testing.T` to check if `simulateWork` generates the expected message.
    - If the test fails, it outputs an error message with details.

Running the Program

To Run the Main Program

Navigate to the Project Directory:
bash
Copy code
```
cd "c:\Users\DEEPAK\Desktop\Project GO\go_concurrency_demo"
```

1.

Run the Program:
bash
Copy code
```
go run main.go
```

2.

Expected Output: The program should print messages from each worker as they complete, such as:
plaintext
Copy code
```
Worker 2 finished work
Worker 5 finished work
Worker 3 finished work
Worker 1 finished work
Worker 4 finished work
```

3.

Running the Tests with `go test`

Navigate to the Project Directory:
bash
Copy code
```
cd "c:\Users\DEEPAK\Desktop\Project GO\go_concurrency_demo"
```

1.

Run the Tests:
bash
Copy code
```
go test
```

2.

Verbose Output (Optional): For more detailed output of each test, use the `-v` flag:
bash
Copy code
```
go test -v
```

3.
4. **Expected Test Output**:

If the test passes:
plaintext
Copy code
```
ok    go_concurrency_demo   0.123s
```

  -
  - If the test fails, `go test` will display the error message indicating what went wrong.

Summary

This project demonstrates:

- **Concurrency** in Go using goroutines, channels, and sync primitives.
- **Unit Testing** with Go's `testing` package and `go test` command for reliable code validation.

With `go test`, we can easily verify the correctness of our code and ensure it performs as expected.