CR1

// Sample program to show how to create goroutines and

// how the scheduler behaves.

package main

import (

"fmt"

"runtime"

"sync"

)

func init() {

// Allocate one logical processor for the scheduler to use.

runtime.GOMAXPROCS(1)

}

func main() {

// wg is used to manage concurrency.

var wg sync.WaitGroup

wg.Add(2)

fmt.Println("Start Goroutines")

// Create a goroutine from the lowercase function.

go func() {

lowercase()

wg.Done()

}()

// Create a goroutine from the uppercase function.

go func() {

uppercase()

wg.Done()

}()

// Wait for the goroutines to finish.

fmt.Println("Waiting To Finish")

wg.Wait()

fmt.Println("\nTerminating Program")

}

// lowercase displays the set of lowercase letters three times.

func lowercase() {

// Display the alphabet three times

for count := 0; count < 3; count++ {

for r := 'a'; r <= 'z'; r++ {

fmt.Printf("%c ", r)

}

}

}

// uppercase displays the set of uppercase letters three times.

func uppercase() {

// Display the alphabet three times

for count := 0; count < 3; count++ {

for r := 'A'; r <= 'Z'; r++ {

fmt.Printf("%c ", r)

}

}

}

CR2

// Sample program to show how the goroutine scheduler

// will time slice goroutines on a single thread.

package main

import (

"crypto/sha1"

"fmt"

"runtime"

"strconv"

"sync"

)

func init() {

// Allocate one logical processor for the scheduler to use.

runtime.GOMAXPROCS(1)

}

func main() {

// wg is used to manage concurrency.

var wg sync.WaitGroup

wg.Add(2)

fmt.Println("Create Goroutines")

// Create the first goroutine and manage its lifecycle here.

go func() {

printHashes("A")

wg.Done()

}()

// Create the second goroutine and manage its lifecycle here.

go func() {

printHashes("B")

wg.Done()

}()

// Wait for the goroutines to finish.

fmt.Println("Waiting To Finish")

wg.Wait()

fmt.Println("Terminating Program")

}

// printHashes calculates the sha1 hash for a range of

// numbers and prints each in hex encoding.

func printHashes(prefix string) {

// print each has from 1 to 10. Change this to 50000 and

// see how the scheduler behaves.

for i := 1; i <= 10; i++ {

// Convert i to a string.

num := strconv.Itoa(i)

// Calculate hash for string num.

sum := sha1.Sum([]byte(num))

// Print prefix: 5-digit-number: hex encoded hash

fmt.Printf("%s: %05d: %x\n", prefix, i, sum)

}

fmt.Println("Completed", prefix)

}

CR3

// Sample program to show how to create goroutines and

// how the goroutine scheduler behaves with two contexts.

package main

import (

"fmt"

"runtime"

"sync"

)

func init() {

// Allocate two logical processors for the scheduler to use.

runtime.GOMAXPROCS(2)

}

func main() {

// wg is used to wait for the program to finish.

// Add a count of two, one for each goroutine.

var wg sync.WaitGroup

wg.Add(2)

fmt.Println("Start Goroutines")

// Declare an anonymous function and create a goroutine.

go func() {

// Display the alphabet three times.

for count := 0; count < 3; count++ {

for r := 'a'; r <= 'z'; r++ {

fmt.Printf("%c ", r)

}

}

// Tell main we are done.

wg.Done()

}()

// Declare an anonymous function and create a goroutine.

go func() {

// Display the alphabet three times.

for count := 0; count < 3; count++ {

for r := 'A'; r <= 'Z'; r++ {

fmt.Printf("%c ", r)

}

}

// Tell main we are done.

wg.Done()

}()

// Wait for the goroutines to finish.

fmt.Println("Waiting To Finish")

wg.Wait()

fmt.Println("\nTerminating Program")

}