SISTEMAS OPERATIVOS II

MANUAL TÉCNICO

GRUPO #12

01/03/2022

PRÁCTICA NO.2

Josselyn Vanessa Polanco Gameros 201602676
Oscar Alfredo Llamas Lemus 201602625
Jose Alejandro Santizo Cotto 201709309
Alison Cristina Leiva Paredes 201700378

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

INTRODUCCIÓN

INICIANDO...

Para la elaboración de la aplicación que se encarga de leer páginas web y analizar su contenido, se utilizó el lenguaje de programación Golang, para la lectura de la página web se hizo un scraping y se utilizaron rutinas para ingresar trabajos a la cola y para iniciar el scraper en segundo plano.
A continuación, se muestra más a detalle el código y el manejo del scraper y rutinas.

INICIO...

MENÚ EN CONSOLA

En la función main, existe un método llamado "LimpiarPantalla" que, como su nombre lo indica, sirve para limpiar la consola, luego de ello, se capturan los valores ingresados por el usuario en consola para poder empezar el flujo del proyecto según los datos ingresados por el usuario.

```
func main() {

    LimpiarPantalla()

    fmt.Println(string(getColor("cyan")), "- Ingrese cantidad de monos
    fmt.Print(string(getColor("green")), "SOPES2 ")
    fmt.Print(string(getColor("yellow")), ">> ")
    fmt.Scanln(&Lectura)
    Lectura = strings.TrimSuffix(Lectura, "/")
    Lectura = strings.TrimSpace(Lectura)
    MonkeysAmmount, _ := strconv.ParseInt(Lectura, 10, 64)

    fmt.Println("")
    fmt.Println(string(getColor("cyan")), "- Ingrese tamaño de la cola
    fmt.Print(string(getColor("green")), "SOPES2 ")
    fmt.Print(string(getColor("yellow")), ">> ")
    fmt.Scanln(&Lectura)
    Lectura = strings.TrimSuffix(Lectura, "/")
    Lectura = strings.TrimSpace(Lectura)
    QueueSize, _ := strconv.ParseInt(Lectura, 10, 64)
```

INICIANDO FLUJO

MI PRIMER JOB

Al terminar de capturar los valores ingresados por el usuario, se crea el primer trabajo con los primeros valores que el usuario ingreso, como el Nr, que será el número de enlaces que se utilizarán al inicio, el URL de la página para construir nuestro archivo JSON.
Luego de llenar el struct, se llama al método SetScraper que iniciará con el flujo del programa, este tiene 3 parámetros, el struct, el número de monos y el tamaño de la cola.

```
myFirstJob := Work{
    SHAPadre: "0",
    URL:      EntryPoint,
    NR:       NrValue,
}

SetScraper(&myFirstJob, MonkeysAmmount, QueueSize)

// Realizando un scrapper de prueba, esto se haria iterando en
//RunScrapper(NrValue, EntryPoint, "mono_01", "0")
fmt.Println(string(getColor("green")), "Scraping terminado. Pre
fmt.Scanln(&Lectura)

fmt.Print(string(getColor("yellow")), "Hasta la próxima! :)")
```

SCRAPER

INICIALIZACIÓN DE SCRAPER

En el método que se mencionó anteriormente, se crea un canal donde pasarán los trabajos, con un tamaño de 1000, luego de ello se crea un arreglo de monos con el tamaño que el usuario asignó.
Al inicializar estas dos variables, se crea un for para llenar los structs de monos con su respectivo ID y su disponibilidad, al inicio es true.

```go
// Setear parametros de scrapper
func SetScraper(firstJob *Work, monosValue int64, queueSize int64) {

    // haciendo un canal donde pasaran Works
    jobs := make(chan Work, 1000)

    // Haciendo un arreglo donde estan mis N monos
    myMonkeys = make([]Monkey, monosValue)

    // Seteando ID para cada mono
    for i := 0; i < int(monosValue); i++ {
        myMonkeys[i].ID = "monkey_0" + strconv.Itoa(i+1) //monkey_01, monkey_02
        myMonkeys[i].Disponible = true
    }

    // Creando cola de trabajos
    myQueue = &Cache{Slots: make([]Work, 0, queueSize)}
    myQueue.Mu.Lock()
    // agregando el trabajo inicial a la cola
    myQueue.Slots = append(myQueue.Slots, *firstJob)
    myQueue.Mu.Unlock()

    // Go routine para leer canal y agregar trabajos a la cola
    go CheckJobs(&jobs, myQueue)
```

SCRAPER

UTILIZANDO RUTINAS Y VERIFICANDO LA COLA

Al tener todos los structs de los monos con sus respectivos valores inicializados, se crea la cola de trabajos con el tamaño que el usuario definió, se bloquea la cola para que no haya ninguna alteración mientras se escribe en ella, y luego se inserta el trabajo a dicha cola, al finalizar, se desbloquea la cola para que pueda seguir trabajando.

```
293     // Creando cola de trabajos
294     myQueue = &Cache{Slots: make([]Work, 0, queueSize)}
295     myQueue.Mu.Lock()
296     // agregando el trabajo inicial a la cola
297     myQueue.Slots = append(myQueue.Slots, *firstJob)
298     myQueue.Mu.Unlock()
299
300     // Go routine para leer canal y agregar trabajos a la cola
301     go CheckJobs(&jobs, myQueue)
302
303     for len(myQueue.Slots) > 0 {
304
305         if len(myQueue.Slots) > 0 {
306
307             newJob := DeQueue()
308             searchMonkey := true
309             monkeyIndex := -1
310
311             for searchMonkey {
312
313                 monkeyIndex = FindAvailableMoney(&myMonkeys)
314                 if monkeyIndex != -1 {
315                     searchMonkey = false
316                 }
317
318             }
319
320             // si llego aqui, es porque hay un mono disponible
321             myMonkeys[monkeyIndex].Disponible = false
322             monkeyID := myMonkeys[monkeyIndex].ID
323             go RunScraper(newJob.NR, newJob.URL, monkeyID, newJob.SHAPadre, jobs, monkeyIndex)
```

SCRAPER

COLA DE TRABAJOS Y RUTINAS

Se llama a la rutina CheckJobs para verificar si hay trabajos nuevos para agregarlos a la cola, luego de ello se saca un nuevo trabajo de la cola y se verifica si el mono está disponible llamando al método FindAvailableMoney

```
293     // Creando cola de trabajos
294     myQueue = &Cache{Slots: make([]Work, 0, queueSize)}
295     myQueue.Mu.Lock()
296     // agregando el trabajo inicial a la cola
297     myQueue.Slots = append(myQueue.Slots, *firstJob)
298     myQueue.Mu.Unlock()
299
300     // Go routine para leer canal y agregar trabajos a la cola
301     go CheckJobs(&jobs, myQueue)
302
303     for len(myQueue.Slots) > 0 {
304
305         if len(myQueue.Slots) > 0 {
306
307             newJob := DeQueue()
308             searchMonkey := true
309             monkeyIndex := -1
310
311             for searchMonkey {
312
313                 monkeyIndex = FindAvailableMoney(&myMonkeys)
314                 if monkeyIndex != -1 {
315                     searchMonkey = false
316                 }
317
318             }
319
320             // si llego aqui, es porque hay un mono disponible
321             myMonkeys[monkeyIndex].Disponible = false
322             monkeyID := myMonkeys[monkeyIndex].ID
323             go RunScraper(newJob.NR, newJob.URL, monkeyID, newJob.SHAPadre, jobs, monkeyIndex)
```

COLAS

VERIFICAR SI HAY TRABAJOS NUEVOS

Método que verifica si hay un nuevo trabajo para agregar a la cola, si es así, se llama al método Queue para agregarlo a la cola de trabajo.

```
heckear constantemente si hay trabajos nuevos para agregar a la cola
 CheckJobs(canal *chan Work, cache *Cache) {
for SeguirScrapper {

    myJob := <-*canal
    // fmt.Println("Se recibio un nuevo work en el canal")
    // fmt.Println(myJob)
    Queue(&myJob)
}
```

COLAS Y PALABRAS

FUNCIONES

El primer método agrega un trabajo a la cola, el segundo saca un trabajo de la cola y devuelve ese mismo trabajo, la tercera función verifica si un mono está disponible o no.

```go
// Agregar un trabajo a la cola
func Queue(work *Work) {
    if len(myQueue.Slots) < cap(myQueue.Slots) {
        myQueue.Mu.Lock()
        myQueue.Slots = append(myQueue.Slots, *work)
        myQueue.Mu.Unlock()
    }
}

// Quitar un trabajo de la cola
func DeQueue() *Work {
    myQueue.Mu.Lock()
    auxWork := myQueue.Slots[0] // x|2|3|4|5|...
    myQueue.Slots = myQueue.Slots[1:]
    myQueue.Mu.Unlock()

    return &auxWork
}

// Buscar un mono disponible
func FindAvailableMoney(monos *[]Monkey) int {

    for i, mono := range *monos {

        if mono.Disponible {
            return i
        }
    }

    return -1
}
```

RUN SCRAPER

REALIZACIÓN DEL SCRAPER

Este método es el que extra la información de la página y llama al método de contador de palabras, se cuentan los enlaces que hay y se genera el SHA para cada work.

```
153 // Metodo para realizar el scrapping
154 /*
155 nr = numero de enlaces que debe buscar en la pagina
156 url = direccion de la pagina
157 mono = id del mono que esta haciendo el scrapping
158 origen = es el SHA del contenido de la pagina donde se
159 */
160 func RunScraper(nr int64, url string, mono string, orig
161
162     contenido := ""
163     var cantidadEnlaces int64 = 0
164     var contadorNR int64 = 0
165     var contentSHA string = ""
166     c := colly.NewCollector()
167
168     c.OnHTML("p", func(e *colly.HTMLElement) {
169         contenido += e.Text + "\n"
170     })
```

RUN SCRAPER

REALIZACIÓN DEL SCRAPER

Sección que busca los enlaces dentro de las etiquetas p, también se hace un foreach para buscar los enlaces y hacer un conteo de estos.

```
172     // Buscando todas los enlaces dentro de las etiquetas p
173     c.OnHTML("p", func(e *colly.HTMLElement) {
174
175         e.ForEach("p", func(_ int, text *colly.HTMLElement) {
176             contenido += text.Text + "\n"
177         })
178
179         h := sha1.New()
180         h.Write([]byte(contenido))
181         // variable con el SHA para el atributo SHAPadre de los nuevos works
182         contentSHA = hex.EncodeToString(h.Sum(nil))
183
184         e.ForEach("a[href]", func(_ int, elem *colly.HTMLElement) {
185             link := elem.Attr("href")
186             if elem.Request.AbsoluteURL(link) != "" {
187
188                 cantidadEnlaces++
189
190                 if nr > 0 && contadorNR < nr {
191
192                     // variable con el enlace para el nuevo work que se ingresara en la cola
193                     linkNuevoWork := elem.Request.AbsoluteURL(link)
194
195                     // variable con el valor de nr - 1
196                     newNR := nr - 1
197
198                     contadorNR++
199
200                     /* AQUI SE MANDARIAN LOS NUEVOS WORKS A LA COLA*/
201                     // fmt.Printf("Se encontro el enlace #%v: %s - nuevo Nr = %v \n ", contadorN
202
```

ARCHIVO JSON

INSERTANDO ITEM EN JSON

Se llena el struct para ingresar un nuevo item al archivo JSON, luego de ello, se hace un conteo de palabras y se hace una breve pausa hasta que el mono esté disponible de nuevo.

```
    JsonResult := Resultado{
        Origen:   origen,
        Palabras: int64(WordCount(contenido)),
        Enlaces:  cantidadEnlaces,
        SHA:      contentSHA,
        URL:      url,
        Mono:     mono,
    }

    myResults = append(myResults, JsonResult)
    contadorEscrituras++
    numberOfMiliseconds := WordCount(contenido)

    mensaje := fmt.Sprintf("El mono %s esta descanzando", myMonkeys[mo
    fmt.Println(string(getColor("yellow")), mensaje)
    time.Sleep(time.Millisecond * time.Duration(numberOfMiliseconds) *
    myMonkeys[monkeyIndex].Disponible = true
    mensaje = fmt.Sprintf("El mono %s ya esta disponible", myMonkeys[m
    fmt.Println(string(getColor("cyan")), mensaje)
}
```