Universidad De San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
LABORATORIO SISTEMAS OPERATIVOS 2
Sección “A”

“MANUAL TECNICO”

Luis Angel Barrera Velásquez

Carné: 202010223

Índice

Introducción

Este manual técnico tiene como objetivo principal ofrecer una descripción detallada del funcionamiento del código fuente del Proyecto 2. Está diseñado para ser de utilidad a cualquier persona con conocimientos en programación que desee comprender y colaborar en futuras actualizaciones o correcciones durante la ejecución del programa. La comprensión profunda de este artículo resulta crucial para lograr una actualización efectiva de este proyecto.

El Proyecto 2 se basa en un enfoque innovador en Go, que involucra la manipulación de procesos concurrentes para la lectura y escritura de números aleatorios y palabras en un archivo de paginación. Además, el programa responde a la entrada del usuario, lo que lo convierte en una experiencia interactiva única.

Este proyecto se estructura en torno a un conjunto de funciones que gestionan operaciones con números y palabras. Los procesos se ejecutan en paralelo, creando un entorno dinámico y desafiante. La comunicación con el usuario se facilita mediante la detección de entrada de teclado, lo que permite un control total del flujo del programa.

A lo largo de este manual, desglosaremos cada componente del sistema, explicando sus funciones y relaciones, lo que permitirá a los desarrolladores comprender y contribuir al proyecto de manera efectiva. Además, se explorarán aspectos clave del código, como la manipulación de archivos, la generación de números aleatorios y la selección de palabras al azar.

Aplicación en Go

Lenguaje Go

El Lenguaje Go, al igual que C y C++, es un lenguaje compilado y concurrente, o en otras palabras: soporta canales de comunicación basados en el lenguaje CSP. Sin embargo, dentro de las características de Go aparece su concurrencia. En Go es diferente a los criterios de programación basados en bloqueos como pthreads. Los creadores de Go, además, se inspiraron en la versatilidad y las cualidades de otros lenguajes como Python, C++ y Java (entre otros), para conseguir un lenguaje con las siguientes características, algunas únicas, y otras compartidas con otros lenguajes compilados.

Explicación de código fuente

bancoDePalabras

Es una variable que almacena una gran lista de palabras en un slice de cadenas.

```go
var bancoDePalabras = []string{
    "manzana", "naranja", "pera", "uva", "kiwi", "platano",
    "perro", "gato", "elefante", "leon", "jirafa", "tigre",
    "sol", "luna", "estrella", "planeta", "galaxia", "nebulosa",
    "flor", "arbol", "pasto", "montana", "oceano", "rio",
    "coche", "bicicleta", "avion", "tren", "barco", "camion",
    "casa", "apartamento", "villa", "castillo", "cabana", "iglu",
    "computadora", "telefono", "television", "tableta", "raton", "teclado",
    "libro", "revista", "periodico", "cuento", "novela", "poesia",
    "pelota", "raqueta", "balon", "bate", "gorra", "uniforme",
    "piano", "violin", "flauta", "guitarra", "bateria", "trompeta",
    "familia", "amigo", "vecino", "colega", "companero", "profesor",
    "arte", "musica", "danza", "pintura", "escultura", "teatro",
    "amor", "felicidad", "tristeza", "enfado", "paz", "esperanza",
    "viaje", "vacaciones", "aventura", "descanso", "turismo", "excursion",
    "mente", "cuerpo", "alma", "corazon", "espiritu", "mente",
    "comida", "bebida", "postre", "ensalada", "sopa", "hamburguesa",
    "ropa", "zapatos", "sombrero", "abrigo", "vestido", "camiseta",
    "lapiz", "pluma", "cuaderno", "libreta", "papel", "borrador",
    "solucion", "problema", "pregunta", "respuesta", "duda", "certeza",
    "arte", "ciencia", "tecnologia", "cultura", "historia", "educacion",
    "fiesta", "celebracion", "aniversario", "cumpleanos", "boda", "graduacion",
    "aventura", "exploracion", "descubrimiento", "misterio", "leyenda", "mito",
    "moda", "tendencia", "estilo", "diseno", "traje", "modelo",
    "ocasion", "reunion", "encuentro", "evento", "conferencia", "congreso",
    "medicina", "enfermedad", "hospital", "medico", "enfermera", "paciente",
    "deporte", "ejercicio", "competicion", "equipo", "jugador", "entrenador",
    "energía", "electricidad", "combustible", "gasolina", "petróleo", "viento",
    "proyecto", "empresa", "negocio", "inversion", "cliente", "mercado",
    "cine", "pelicula", "director", "actor", "actriz", "escenario",
    "naturaleza", "paisaje", "ecosistema", "clima", "tierra", "cielo",
    "historia", "relato", "leyenda", "mito", "cuento", "novela",
```

crearArchivo()

Esta función se encarga de crear un archivo llamado "paginacion.txt". Si ocurre algún error durante la creación, muestra un mensaje de error y finaliza el programa.

```go
func crearArchivo() {
    file, err := os.Create("paginacion.txt")
    if err != nil {
        fmt.Println("Error al crear el archivo", err)
        os.Exit(1)
    }
    file.Close()
}
```

borrarArchivo()

Elimina el archivo "paginacion.txt". Si hay algún error al eliminar el archivo, muestra un mensaje de error.

```go
func borrarArchivo() {
    err := os.Remove("paginacion.txt")
    if err != nil {
        fmt.Println("Error al eliminar el archivo", err)
    }

    fmt.Println("Archivo de paginación borrado.")
}
```

procesoNumero(tipo int, proceso int)

Esta función realiza dos tipos de operaciones: escritura y lectura de números. El argumento **tipo** indica si se trata de una operación de escritura (1) o lectura (0). Dependiendo del tipo, se realiza una de las siguientes acciones:

- Escritura (tipo = 1): Genera un número aleatorio del 1 al 999, lo escribe en el archivo "paginacion.txt" y muestra un mensaje con el número generado.
- Lectura (tipo = 0): Lee los números previamente escritos en el archivo, selecciona uno al azar y muestra un mensaje con el número seleccionado.

```go
func procesoNumero(tipo int, proceso int) {
    if tipo == 1 {
        numeroAleatorio := rand.Intn(999) + 1

        file, err := os.OpenFile("paginacion.txt", os.O_APPEND|os.O_WRONLY, os.ModeAppend)
        if err != nil {
            fmt.Println("Error al abrir el archivo", err)
            return
        }
        defer file.Close()

        _, err = fmt.Fprintf(file, "%d\n", numeroAleatorio)
        if err != nil {
            fmt.Println("Error al escribir en el archivo", err)
        }
        fmt.Printf("Proceso %d escribiendo numero: %d\n", proceso, numeroAleatorio)
    } else if tipo == 0 {
        numeros := []int{}
        archivo, err := os.Open("paginacion.txt")
        if err != nil {
            fmt.Println("Error al abrir el archivo", err)
            return
        }
        defer archivo.Close()

        scanner := bufio.NewScanner(archivo)
        for scanner.Scan() {
            linea := scanner.Text()
            numero, err := strconv.Atoi(linea)
            if err == nil {
                numeros = append(numeros, numero)
            }
        }

        if len(numeros) == 0 {
```

procesoLetra(tipo int, proceso int)

Similar a **procesoNumero()**, pero en lugar de números, opera con palabras aleatorias. Escritura (tipo = 1) y lectura (tipo = 0).

```go
func procesoLetra(tipo int, proceso int) {
    if tipo == 1 {
        palabraAleatoria := bancoDePalabras[rand.Intn(len(bancoDePalabras))]

        file, err := os.OpenFile("paginacion.txt", os.O_APPEND|os.O_WRONLY, os.ModeAppend)
        if err != nil {
            fmt.Println("Error al abrir el archivo", err)
            return
        }
        defer file.Close()

        _, err = fmt.Fprintf(file, "%s\n", palabraAleatoria)
        if err != nil {
            fmt.Println("Error al escribir en el archivo", err)
        }
        fmt.Printf("Proceso %d escribiendo letra: %s\n", proceso, palabraAleatoria)

    } else if tipo == 0 {
        palabras := []string{}
        archivo, err := os.Open("paginacion.txt")
        if err != nil {
            fmt.Println("Error al abrir el archivo", err)
            return
        }
        defer archivo.Close()

        scanner := bufio.NewScanner(archivo)
        for scanner.Scan() {
            linea := scanner.Text()
            if regexp.MustCompile(`^[a-zA-Z]+$`).MatchString(linea) {
                palabras = append(palabras, linea)
            }
        }

        if len(palabras) == 0 {
            fmt.Printf("Proceso %d leyendo letra: Inexistente\n", proceso)
```

abrirNuevaTerminalYEjecutarComando(comando string)

Esta función ejecuta un comando en una nueva terminal de GNOME. Toma un comando como argumento y lo ejecuta. El resultado de la ejecución se almacena en un archivo temporal y se muestra en una nueva terminal.

```go
func abrirNuevaTerminalYEjecutarComando(comando string) error {
    cmd := exec.Command("bash", "-c", comando+" > /tmp/paginacion_output.txt")
    err := cmd.Run()
    if err != nil {
        return err
    }

    cmd = exec.Command("gnome-terminal", "--", "bash", "-c", "cat /tmp/paginacion_output.txt; read -p 'Presiona Enter para continuar...'")

    return cmd.Run()
}
```

ciclo()

Esta función inicia un bucle infinito en una goroutine que genera números aleatorios y palabras aleatorias. Luego, selecciona aleatoriamente un proceso (1-6) y realiza una operación de lectura o escritura según el proceso seleccionado. También muestra algunas opciones y duerme durante un tiempo antes de continuar.

```
func ciclo() {
    go func() {
        for {
            rand.Seed(time.Now().UnixNano())

            // Genera un número aleatorio del 1 al 6
            numeroAleatorio := rand.Intn(6) + 1
            fmt.Println("**************************************
            switch numeroAleatorio {
            case 1:
                procesoNumero(1, 1)
                procesoNumero(0, 2)
                procesoNumero(0, 3)
                procesoLetra(0, 4)
                procesoLetra(0, 5)
                procesoLetra(0, 6)
            case 2:
                procesoNumero(0, 1)
                procesoNumero(1, 2)
                procesoNumero(0, 3)
                procesoLetra(0, 4)
                procesoLetra(0, 5)
                procesoLetra(0, 6)
            case 3:
                procesoNumero(0, 1)
                procesoNumero(0, 2)
                procesoNumero(1, 3)
                procesoLetra(0, 4)
                procesoLetra(0, 5)
                procesoLetra(0, 6)
            case 4:
                procesoNumero(0, 1)
                procesoNumero(0, 2)
                procesoNumero(0, 3)
                procesoLetra(1, 4)
                procesoLetra(0, 5)
```

main()

En la función principal, se realiza lo siguiente:

- Se crea el archivo "paginacion.txt" utilizando **crearArchivo()**.
- Se abre una conexión de teclado con **keyboard.Open()** y se cierra al final para garantizar la limpieza.
- Se inicia el ciclo principal de la aplicación con **ciclo()** en una goroutine.
- Se establece un canal de control "quit" para permitir la terminación del programa.
- Se inicia una goroutine para escuchar la entrada del usuario:
- Si el usuario presiona "q" o la tecla Esc, el programa se cierra.
- Si el usuario presiona "p", se llama a **abrirNuevaTerminalYEjecutarComando()** para mostrar

el contenido del archivo "paginacion.txt".

- El programa espera hasta que se reciba una señal de terminación desde el canal "quit".
- Finalmente, se borra el archivo "paginacion.txt" utilizando **borrarArchivo()**.

```go
func main() {
    crearArchivo()

    err := keyboard.Open()
    if err != nil {
        fmt.Println(err)
        os.Exit(1)
    }
    defer func() {
        _ = keyboard.Close()
    }()

    ciclo()
    quit := make(chan bool)
    go func() {
        for {
            select {
            case <-quit:
                return
            default:
                if char, key, _ := keyboard.GetKey(); key == keyboard.KeyEsc || char == 'q' {
                    quit <- true
                    return
                } else if char, key, _ := keyboard.GetKey(); key == keyboard.KeyEsc || char == 'p' {
                    go abrirNuevaTerminalYEjecutarComando("cat paginacion.txt")
                }
            }
        }
    }()

    <-quit

    borrarArchivo()
}
```

Conclusiones

- El núcleo de este proyecto se encuentra en Go y se destaca por su estructura organizada. Las funciones están claramente definidas, y los paquetes específicos se utilizan con eficacia para tareas como la generación de números aleatorios y la escritura/lectura de datos en un archivo. Además, la elección de herramientas estándar de Go y bibliotecas externas, como la biblioteca "keyboard," demuestra un enfoque inteligente para el desarrollo de aplicaciones de consola en Go.
- Este proyecto aprovecha la concurrencia para crear procesos que generan números y palabras aleatorias, lo que añade un nivel de complejidad y emoción a la aplicación. La generación de números aleatorios y la selección de palabras al azar ofrecen una experiencia diversa y dinámica.
- El programa responde a la entrada del usuario, permitiendo acciones como la visualización del contenido del archivo de paginación o la terminación controlada del programa. Esta interacción aporta una dimensión adicional a la experiencia de usuario y fomenta la exploración del código.

Referencias

- López, J. G. (2010). Monitorización de Sistemas GNU/Linux. Todo Linux: la revista mensual para entusiastas de GNU/Linux, (113), 10-15.
- Morales, C. E. G., Ramírez, J. A. V., & Mendoza, N. F. (2015). Inyección de Dependencias en el Lenguaje de Programación Go. RIDE Revista Iberoamericana para la Investigación y el Desarrollo Educativo, 5(10).