Manual Técnico

MANEJO E IMPLEMENTACIÓN DE ARCHIVOS

José Rafael Morente González | 201801237
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Contenido

Go

Go es un lenguaje de programación concurrente y compilado inspirado en la sintaxis de C, que intenta ser dinámico como Python y con el rendimiento de C o C++. Ha sido desarrollado por Google, y sus diseñadores iniciales fueron Robert Griesemer, Rob Pike y Ken Thompson. Actualmente está disponible en formato binario para los sistemas operativos Windows, GNU/Linux, FreeBSD y Mac OS X, pudiendo también ser instalado en estos y en otros sistemas mediante el código fuente.

Es el lenguaje utilizado para el proyecto de Archivos.

Creación de Archivos Binarios

Para escribir un archivo binario en Golang necesitamos de indicarle un tamaño y esto lo hacemos con el método seek el cual usamos para leer o escribir dentro de una posición en donde le indicamos el desplazamiento hasta donde queremos llegar y también establecemos desde donde queremos llegar, por ejemplo,

Tamanio: nos dice el tamaño que le queremos dar al disco, mientras que 0 nos indica que lo haremos desde la parte inicial.

```
archivo.Seek(tamanio, 0);
```

Posteriormente nosotros escribimos la información dentro de la estructura que hemos creado para ello realizamos las acciones correspondientes y escribimos los bytes para que se almacene la información dentro del archivo binario.

```
//ASIGNAR VALORES A STRUCT
fechaHora := time.Now();
disco := Modelo.MBR{}
disco.Mbr_size = size*sizeUNIT-1;
disco.Mbr_size_disponible = size*sizeUNIT-1;
copy(disco.Mbr_date[:], fechaHora.Format("2006-01-02 15:04:05"))
disco.Mbr_disk_signature = rand.Int63();
disco.Mbr_partition_1 = Modelo.PARTICION{};
disco.Mbr_partition_2 = Modelo.PARTICION{};
disco.Mbr_partition_3 = Modelo.PARTICION{};
disco.Mbr_partition_4 = Modelo.PARTICION{};
s1 := &disco

//ESCRITURA DEL STRUCT
var binarioTemporal3 bytes.Buffer
binary.Write(&binarioTemporal3, binary.BigEndian, s1)
escrituraBytes(archivo, binarioTemporal3.Bytes())
```

```
//ESCRIBIR CERO AL INICIO DEL ARCHIVO
var binarioTemporal bytes.Buffer
binary.Write(&binarioTemporal, binary.BigEndian, s)
escrituraBytes(archivo, binarioTemporal.Bytes())

tamanio := sizeUNIT*size-1;
//TAMAÑO DE ARCHIVO SEGUN UNIT
archivo.Seek(tamanio, 0);

//ESCRIBIR 0 AL FINAL DEL ARCHIVO
var binarioTemporal2 bytes.Buffer
binary.Write(&binarioTemporal2, binary.BigEndian, s)
escrituraBytes(archivo, binarioTemporal2.Bytes())
archivo.Seek(0, 0)

//ASIGNAR VALORES A STRUCT
fechaHora := time.Now();
disco := Modelo.MBR{}
disco.Mbr_size = size*sizeUNIT-1;
disco.Mbr_size_disponible = size*sizeUNIT-1;
copy(disco.Mbr_date[:], fechaHora.Format("2006-01-02 15:04:05"))
disco.Mbr_disk_signature = rand.Int63();
disco.Mbr_partition_1 = Modelo.PARTICION{};
disco.Mbr_partition_2 = Modelo.PARTICION{};
disco.Mbr_partition_3 = Modelo.PARTICION{};
disco.Mbr_partition_4 = Modelo.PARTICION{};
s1 := &disco

//ESCRITURA DEL STRUCT
var binarioTemporal3 bytes.Buffer
binary.Write(&binarioTemporal3, binary.BigEndian, s1)
escrituraBytes(archivo, binarioTemporal3.Bytes())
color.Green("╔═══════════════════════════════════════════════╗")
color.Green("  Disco creado exitosamente :D")
color.Green("╚═══════════════════════════════════════════════╝")
```

Lectura de Archivos Binarios

Para la lectura de archivos lo que realizamos es declarar la estructura en primer lugar, luego de eso leemos la cantidad de bites del tamaño de la estructura que declaramos y el archivo que buscamos, entonces decodificamos esos datos para posteriormente ingresarlos dentro de la estructura declarada se realiza esto para que al momento de que necesitemos acceder a las propiedades del archivo binario lo podamos realizar sin problemas.

```
//DECLARAR MBR Y OBTENER TAMAÑO
m := Modelo.MBR{}
var size int = int(unsafe.Sizeof(m))

//LEER CANTIDAD DE BYTES
data := leerBytes(file, size)
//DECODIFICACION EN BINARIO
buffer := bytes.NewBuffer(data)

//GUARDAR VARIABLE EN M
err = binary.Read(buffer, binary.BigEndian, &m)
if err != nil {
    log.Fatal("binary.Read failed", err)
}
```

Formateo MKFS

Para realizar el formateo de MKFS se calcula el tamaño de las estructuras, y para ello utilizamos la formula para calcular el numero de estructuras que puede almacenar el sistema de archivos LWS.

$$NE = \frac{SizePartición - 2 * SizeSuperbloque}{27 + SizeAVD + SizeDD + (5 * SizeIN + (20 * SizeB) + SizeBitacora)}$$

De ahí sabemos por teoría que la cantidad de estructuras del Sistema de Archivo LWS es la siguiente:

$No.\ AVD = NE$

$No.\ DD = NE$

$No.\ iNodos = NE * 5$

$No.\ Bloque = NE * 20$

$No.\ Bitácora = NE$

```
var sizeSB int64 = int64(unsafe.Sizeof(Modelo.SUPERBOOT{}))
var sizeAVD int64 = int64(unsafe.Sizeof(Modelo.AVD{}))
var sizeDD int64 = int64(unsafe.Sizeof(Modelo.DD{}))
var sizeInodos int64 = int64(unsafe.Sizeof(Modelo.INODO{}))
var sizeBloques int64 = int64(unsafe.Sizeof(Modelo.BLOQUE{}))
var sizeBitacora int64 = int64(unsafe.Sizeof(Modelo.BITACORA{}))

var cantidadEstructuras int64 = (particion.Part_size - (2 * int64(sizeSB))) / (27 + in
var cantidadAVD int64 = cantidadEstructuras
var cantidadDD int64 = cantidadEstructuras
var cantidadInodos int64 = 5 * cantidadEstructuras
var cantidadBloques int64 = 20 * cantidadEstructuras
var cantidadBitacoras int64 = cantidadEstructuras
```

Ya sabiendo esto entonces averiguamos donde inicia cada estructura para poder posicionarlas dentro de nuestro sistema de archivos LWS.

```
Inicio_bitmapAVD := particion.Part_start + sizeSB
Inicio_AVD := Inicio_bitmapAVD + cantidadAVD
Inicio_bitmapDD := Inicio_AVD + (sizeAVD * cantidadAVD)
Inicio_DD := Inicio_bitmapDD + cantidadDD
Inicio_bitmapInodo := Inicio_DD + (sizeDD * cantidadDD)
Inicio_Inodos := Inicio_bitmapInodo + cantidadInodos
Inicio_bitmapBloque := Inicio_Inodos + (sizeInodos * cantidadInodos)
Inicio_Bloque := Inicio_bitmapBloque + cantidadBloques
Inicio_Bitacora := Inicio_Bloque + (sizeBloques * cantidadBloques)
Inicio_SBCopia := Inicio_Bitacora + (sizeBitacora * cantidadBitacoras)
```

```
/**
* ESCRITURA DE SUPERBLOQUE
*/
file.Seek(particion.Part_start, 0)
var SB = Modelo.SUPERBOOT{};
fechaHora := time.Now();
copy(SB.SB_nombre_hd[:], nombre);
copy(SB.SB_date_creacion[:], fechaHora.Format("2006-01-02 15:04:05"))
copy(SB.SB_date_ultimo_montaje[:], fechaHora.Format("2006-01-02 15:04:05"))
SB.SB_arbol_virtual_count = cantidadEstructuras;
SB.SB_detalle_directorio_count = cantidadDD;
SB.SB_inodos_count = cantidadInodos;
SB.SB_bloques_count = cantidadBloques;
SB.SB_arbol_virtual_free = cantidadEstructuras;
SB.SB_detalle_directorio_free = cantidadDD;
SB.SB_inodos_free = cantidadInodos;
SB.SB_bloques_free = cantidadBloques;
SB.SB_montaje_count = 1;
SB.SB_ap_bitmap_arbol_directorio = Inicio_bitmapAVD;
SB.SB_ap_arbol_directorio = Inicio_AVD;
SB.SB_ap_bitmap_detalle_directorio = Inicio_bitmapDD;
SB.SB_ap_detalle_directorio = Inicio_DD;
SB.SB_ap_bitmap_tabla_inodo = Inicio_bitmapInodo;
SB.SB_ap_tabla_inodo = Inicio_Inodos;
SB.SB_ap_bitmap_bloques = Inicio_bitmapBloque;
SB.SB_ap_bloques = Inicio_Bloque;
SB.SB_ap_log = Inicio_Bitacora;
SB.SB_size_struct_arbol_directorio = sizeAVD;
SB.SB_size_struct_detalle_directorio = sizeDD;
SB.SB_size_struct_inodo = sizeInodos;
SB.SB_size_struct_bloque = sizeBloques;
SB.SB_free_first_bit_struct_arbol_directorio = SB.SB_ap_bitmap_arbol_directorio;
SB.SB_free_first_bit_struct_detalle_directorio = SB.SB_ap_bitmap_detalle_directorio;
SB.SB_free_first_bit_struct_inodo = SB.SB_ap_bitmap_tabla_inodo;
SB.SB_free_first_bit_struct_bloque = SB.SB_ap_bitmap_bloques;
SB.SB_magic_num = 201801237;
```

Luego de tener todos los tamaños nosotros insertamos dentro del sistema de archivos LWS

- Superbloque
- Bitmap AVD
- Árbol de Directorio
- Bitmap DD
- Detalle de Directorio
- Bitmap Inodos
- Inodos
- Bitmap Bloques
- Bloques
- Bitacoras
- Copia de Superbloque

Para simular la partición nos ubicamos al inicio del Bitmap y al final del bitmap así escribiremos los 0 de esa porción para indicar el formateo inicial, lo mismo se hace con cada estructura del archivo LWS.

```
var p = SB;
a := &p;
var binarioP bytes.Buffer;
binary.Write(&binarioP, binary.BigEndian, a);
file.Write(binarioP.Bytes());

/**
* ESCRITURA BITMAP AVD
*/
for i := Inicio_bitmapAVD; i < Inicio_AVD; i++ {
    var init int8 = '0'
    o := &init
    file.Seek(i, 0)
    var binarioTemp bytes.Buffer
    binary.Write(&binarioTemp, binary.BigEndian, o)
    escrituraBytes(file, binarioTemp.Bytes())
}
```

Graphviz

Graphviz es un programa de visualización gráfica de fuente abierta. La visualización de gráficos es una forma de representar información estructural como diagramas de gráficos y redes abstractos. Tiene importantes aplicaciones en redes, bioinformática, ingeniería de software, diseño de bases de datos y web, aprendizaje automático y en interfaces visuales para otros dominios técnicos.

Utilizado para generar los reportes dentro del proyecto para ello se utiliza la siguiente línea de código en Go.

En el comando exec.Command recibe 4 parametros para ejecutar graphviz las cuales son dot, strFile es la ubicación del archivo .dot, -o y srtPath es el path donde se encontrara la imagen.

```go
/**
 * EJECUTAR COMANDO
 */
func executeCommand(strFile string, strPath string)  {
    cmd := exec.Command("dot", "-Tpng", strFile, "-o", strPath)
    //color.Cyan(cmd)
    err := cmd.Run()

    if err != nil {
        log.Fatal(err)
        color.Red("╔═══════════════════════════════════════════════════════════════════════╗")
        color.Red("        Error al generar grafico comando D:")
        log.Fatal(err)
        color.Red("╚═══════════════════════════════════════════════════════════════════════╝")
    }
}
```

		Extendida									
MBR	Libre	EBR	Lógica	EBR	Lógica	EBR	Lógica	EBR	Lógica	Libre	Libre

SB_nombre_hd	00000000
SB_arbol_virtual_count	00000000
SB_detalle_directorio_count	00000000
SB_inodos_count	00000000
SB_bloques_count	00000000
SB_arbol_virtual_free	00000000
SB_detalle_directorio_free	00000000
SB_inodos_free	00000000
SB_bloques_free	00000000
SB_date_creacion	00000000
SB_date_ultimo_montaje	00000000
SB_montaje_count	00000000
SB_ap_bitmap_arbol_directorio	00000000
SB_ap_arbol_directorio	00000000
SB_ap_bitmap_detalle_directorio	00000000
SB_ap_detalle_directorio	00000000
SB_ap_bitmap_tabla_inodo	00000000
SB_ap_tabla_inodo	00000000
SB_ap_bitmap_bloques	00000000
SB_ap_bloques	00000000
SB_ap_log	00000000
SB_size_struct_arbol_directorio	00000000
SB_size_struct_detalle_directorio	00000000
SB_size_struct_inodo	00000000
SB_size_struct_bloque	00000000
SB_free_first_bit_struct_arbol_directorio	00000000
SB_free_first_bit_struct_detalle_directorio	00000000
SB_free_first_bit_struct_inodo	00000000
SB_free_first_bit_struct_bloque	00000000
SB_magic_num	00000000