MANUAL TÉCNICO

Integrantes

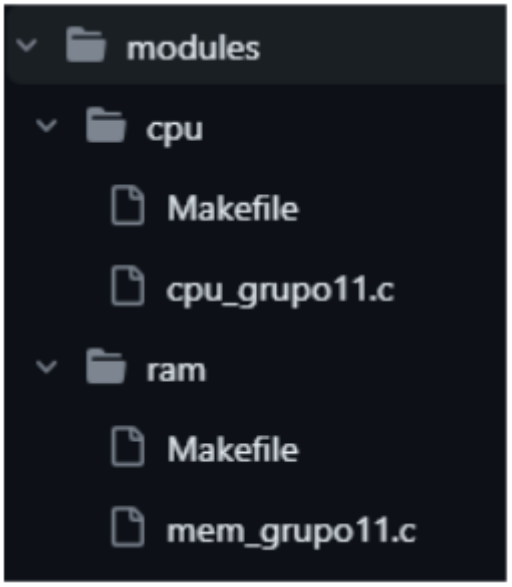
201905554 - Marvin Eduardo Catalán Véliz
201908053 - Sara Paulina Medrano Cojulún
201709110 - Wilson Eduardo Perez Echeverria

Implementacion de modulos

Monitor de memoria

Se creó un módulo de kernel el cual se llama mem_grupo11 el cual extrae información sobre el uso de memoria en el sistema.
Dentro del proyecto el módulo del kernel creado se encuentra en la carpeta ram

En la carpeta de modules



La estructura para el módulo del kernel es la siguiente:
Las librerías implementadas para realizar la operación requerida del total de memoria ram del sistema y total de memoria ram utilizada.
son las siguientes:

```c
#include <linux/module.h>      // Archivo de encabezado para la programación de módulos del kernel de Linux
#include <linux/init.h>        // Archivo de encabezado para macros de inicialización
#include <linux/seq_file.h>    // Archivo de encabezado para la escritura de archivos de secuencia
#include <linux/mm.h>          // Archivo de encabezado para operaciones de gestión de memoria
#include <linux/proc_fs.h>     // Archivo de encabezado para el sistema de archivos /proc
#include <linux/kernel.h>      // Archivo de encabezado para macros y funciones del kernel
#include <linux/sysinfo.h>     // Archivo de encabezado para obtener información del sistema
```

Estructura para almacenar información en el sistema

```c
struct sysinfo inf;
```

Función para escribir en el archivo de secuencia

```c
static int write_file(struct seq_file *file, void *v){
    long total_mem, free_mem;
    si_meminfo(&inf);   // Obtener información de memoria del sistema y almacenarla en la estructura inf
    total_mem = (inf.totalram * 4 / 1024);   // Calcular la cantidad total de memoria en mb
    free_mem = (inf.freeram * 4 / 1024);     // Calcular la cantidad de memoria libre en mb
    seq_printf(file, "{\n");                                    // Escribir una cadena en el archivo de secuencia
    seq_printf(file, " \"MemoriaTotal\":%8lu,\n", total_mem);       // Escribir la cantidad total de memoria
    seq_printf(file, " \"MemoriaLibre\":%8lu,\n", free_mem);        // Escribir la cantidad de memoria libre
    seq_printf(file, " \"MemoriaUsada\":%i\n", 100 - (free_mem * 100) / total_mem);   // Escribir el porcentaje de memoria utilizada
    seq_printf(file, "}\n");                                    // Escribir una cadena en el archivo de secuencia
    return 0;
}
```

Función para abrir el archivo

```c
static int to_open(struct inode *inode, struct file *file){
    return single_open(file, write_file, NULL);   // Abrir el archivo de secuencia y llamar a la función write_file
}
```

Si el kernel es 5.6 o superior, se usa la estructura proc_ops

```c
static struct proc_ops operations =
{
    .proc_open = to_open,    // Puntero a la función que se ejecuta al abrir el archivo /proc
    .proc_read = seq_read    // Puntero a la función de lectura de secuencia
};
```

Función de montaje del módulo

```c
static int mount_module(void){
    proc_create("mem_grupo11", 0, NULL, &operations);   // Crear una entrada en /proc para
    printk(KERN_INFO "Hola mundo, somos el grupo 11 y este es el monitor de memoria\n");
    return 0;
}
```

Función de desmontaje del módulo

```c
static void disassemble_module(void){
    remove_proc_entry("mem_grupo11", NULL);   // Eliminar la entrada en /proc para el archivo "
    printk(KERN_INFO "Sayonara mundo, somos el grupo 11 y este fue el monitor de memoria\n");
}
```

Especificar la función de inicialización del módulo

```c
module_init(mount_module);
```

Especificar la función de desmontaje del módulo

```c
module_exit(disassemble_module);
```

Backend

EL backend se realizó en lenguaje de go para realizar la una api que envia los datos que registran los módulos
La estructura utilizada para el backend en go es la siguiente.
Importación de librerías:

```go
package main

import (
	"fmt"
	"log"
	"net/http"
	"os/exec"
	"github.com/gorilla/mux"    // Importar paquete para enrutamiento HTTP
	"github.com/rs/cors"        // Importar paquete para configurar CORS (Cross-Origin Resource Sharing)
	"bytes"
	"time"
)
```

Función para leer el módulo de CPU

```go
func LeerCpu(w http.ResponseWriter, r *http.Request){
	// Ejecutar el comando "cat /proc/cpu_grupo11" en el sistema operativo
	cmd := exec.Command("sh", "-c", "cat /proc/cpu_grupo11")
	out, err := cmd.CombinedOutput()
	if err != nil {
		log.Fatal(err)
	}
	go fmt.Println("Módulo CPU obtenido correctamente")
	output := string(out[:])

	fmt.Fprintf(w, output)
}
```

Función para leer el módulo de RAM

```go
func LeerRam(w http.ResponseWriter, r *http.Request){
        // Ejecutar el comando "cat /proc/mem_grupo11" en el sistema operativo
        cmd := exec.Command("sh", "-c", "cat /proc/mem_grupo11")
        out, err := cmd.CombinedOutput()
        if err != nil {
                log.Fatal(err)
        }
        go fmt.Println("Módulo RAM obtenido correctamente")
        output := string(out[:])
        fmt.Fprintf(w, output)
}
```

Función para matar un proceso

```go
func killProcess(w http.ResponseWriter, r *http.Request){
        // Leer el cuerpo de la solicitud HTTP para obtener el ID del proceso a matar
        buf := new(bytes.Buffer)
        buf.ReadFrom(r.Body)
        newStr := buf.String()
        str := "kill "+newStr

        // Ejecutar el comando "kill <pid>" en el sistema operativo
        cmd := exec.Command("sh", "-c", str)
        out, err := cmd.CombinedOutput()
        if err != nil {
                go fmt.Println("error")
        }
        go fmt.Println(out)
        fmt.Fprintf(w, "OK")
}
```

Función main como principal

```go
func main() {
	// Crear un enrutador utilizando el paquete gorilla/mux
	router := mux.NewRouter().StrictSlash(true)

	go fmt.Println("Server Running on port: 8080")

	// Definir las rutas y las funciones de controlador correspondientes
	go router.HandleFunc("/leerram", LeerRam).Methods("GET")         // Ruta para leer el módulo de RAM
	go router.HandleFunc("/leercpu", LeerCpu).Methods("GET")         // Ruta para leer el módulo de CPU
	go router.HandleFunc("/killprocess", killProcess).Methods("POST")  // Ruta para matar un proceso

	time.Sleep(time.Second)

	// Configurar CORS (Cross-Origin Resource Sharing) para permitir acceso a recursos desde diferentes dominios
	handler := cors.Default().Handler(router)

	// Iniciar el servidor HTTP en el puerto 8080
	http.ListenAndServe(":8080", handler)
	log.Fatal(http.ListenAndServe(":8080", router))
}
```

Frontend

El frontend nos muestra las gráficas y los datos estadísticos consumidos desde la api de golang, el frontend está implementado con nodejs y react
Las librerías utilizadas para la aplicación son las siguientes
"axios": "^0.21.0",
"bootstrap": "^4.5.3",
"chart.js": "^2.9.4",
"react": "^17.0.1",
"react-bootstrap": "^1.4.0",
"react-chartjs-2": "^2.11.1",
"react-dom": "^17.0.1",
"react-router-dom": "^5.2.0",
"react-scripts": "4.0.1",
"web-vitals": "^0.2.4"
la aplicación se encuentra en la carpeta de Frontend
para ejecutar el programa se debe de tener instalado nodejs
y ejecutar el siguiente comando
npm i o bien npm i –force
luego de esto correr el comando
node start