SO1_Proyecto1

Proyecto #1: Sistemas Operativos 1

Monitor de Recursos y procesos en un servidor web utilizando goland.

Manual tecnico: Creacion de modulos

El proyecto se divide en 4 partes

1. Creacion de modulos
2. Api - Backend
3. FrontEnd
4. Test

1. Creacion de modulos

Esta parte se divide en 2

1. Modulo de CPU
2. Modulo de RAM

En ambos casos se crea un archivo .c en el cual se detalla el comportamiento del modulo y un archivo Makefile para generar el modulo en base al archivo .c.

Creacion de un modulo

Para crear un modulo se sigue el siguiente algoritmo:

- ELIMINAR EL MODULO * entrar en la carpeta proc

```
  $ cd proc
```

listar contenido

```
$ ls
```

remover el modulo si existe

```
$ sudo rmmod [nombre_modulo]
```

mostrar mensajes del kernel de modulos

```
$ dmesg
```

listar nuevamente el contenido de la carpeta

```
$ls
```

- GENERAR LOS ARCHIVOS QUE SE NECESITAN PARA LOS MODULOS *

Entrar en la carpeta donde esta el archivo make y el archivo .c

```
$ cd [nombre_carpeta]
  $ make all
```

genera un archivo .ko (ES EL IMPORTANTE)

MONTAR UN MODULO

En la carpeta donde se genero el archivo

se debe montar el archivo .ko (kernel object)

```
$ sudo insmod [nombre_archivo.ko]
```

mostrar los mensajes en el buffer

```
dmesg
```

ver contenido del archivo generado en /proc

```
$ cat [nombre_archivo]
```

Modulo de CPU

Para la creacion del modulo de CPU es necesaria la importacion de la libreria linux/sysinfo para un calculo en base a la memoria ram de los procesos y las librerias <linux/sched.h>, <linux/sched/signal.h> que seran con las cuales se accedera a los procesos que esta ejecutando el sistema.

Para los procesos tambien se necesita la libreria #include <linux/sched/task.h> para tener un struck que nos dira las tareas que se ejecutan.

El modulo hara un archivo dentro del directorio /proc en formato JSON con lo cual desde la API se podra leer y hacer una importacion de los datos hacia el frontend

Las demas librerias que se importaron para creacion de archivos y otras tareas son:

```
 #include <linux/sched.h>
 #include <linux/sysinfo.h>
 #include <linux/module.h>
 #include <linux/moduleparam.h>
 #include <linux/init.h>
 #include <linux/kernel.h>
 #include <linux/proc_fs.h>
 #include <linux/uaccess.h>
 #include <linux/fs.h>
 #include <linux/utsname.h>
 #include <linux/mm.h>
 #include <linux/swapfile.h>
 #include <linux/seq_file.h>
 #include <linux/sched/task.h>
```

Para obtener los procesos de ejecucion se utilizo una llama recursiva hacia la funcion 'escribir_archivo' con el cual se iba accediendo al struct de tipo task_struct. En cada llamada se va enviando la tarea principal y llamando recursivamente a cada uno de sus hijos.

```
static int escribir_procesos (struct seq_file *archivo, struct task_struct *task)
```

La llamada recursiva esta dada de la siguiente forma:

```
list_for_each(list, &task->children) {
   child = list_entry(list, struct task_struct, sibling);
   escribir_procesos(archivo, child);
   seq_printf(archivo, ",");
}
```

Para obtener el numero de procesos en cada estado se utilizo la misma funcion pero se toma en cuenta el atributo state de cada struct task_struct de la siguiente forma:

```
if (task->state == TASK_RUNNING){
  state = 'E';
  nprocexecuting = nprocexecuting + 1;
}
```

Otra accion que se realiza en base a lo solicitado es imprimir en el buffer de las acciones del kernel mensajes correspondientes a cada accion que se realice en el modulo, como lo es insertarlo o eliminarlo, la forma de hacerlo fue la siguiente:

```
static int inicializar(void)
{
    proc_create("cpu_201603095", 0, NULL, &operaciones);
    printk(KERN_INFO "Abner Fernando Cardona Ramirez\n");

    return 0;
}

static void finalizar(void)
{
    remove_proc_entry("cpu_201603095", NULL);
    printk(KERN_INFO "Diciembre 2020\n");
}
```

Modulo de RAM

Para el modulo de la memoria RAM se importo la libreria <linux/sysinfo> y con esta se obtuvieron los datos necesarios sobre la utilizacion de la memoria RAM asi tambien se obtuvieron los datos necesarios para que se puedieran hacer otros calculos requeridos.

Al igual que con el modulo del CPU en este caso tambien se escribio un archivo en formato JSON para su posterior lectura por la API. Para todas los requerimientos y mas comportamientos del modulo se incluyeron las siguientes librerias:

```
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <asm/uaccess.h>
#include <linux/hugetlb.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/sysinfo.h>
```

En este caso solo se requirio acceder al struct sysinfo para obtener la informacion necesaria para el modulo.

```
struct sysinfo informacion_sistema;
```

posteriormente se escriben los parametros en el archivo y los guarda, y asi este modulo sobreescribe los datos cada vez que es abierto.

El metodo para escribir el archivo tiene la siguiente definicion:

```
static int escribir_archivo(struct seq_file * archivo, void *v) {
    si_meminfo(&informacion_sistema);
    long totalRam   = (informacion_sistema.totalram * 4);
    long ramLibre   = (informacion_sistema.freeram * 4 );
    seq_printf(archivo, "{\n");
    seq_printf(archivo, "\t\"totalRam\":\"%8lu\",\n", totalRam);
    seq_printf(archivo, "\t\"ramLibre\":\"%8lu\"\n", ramLibre);
    seq_printf(archivo, "}\n");
    return 0;
}
```

Posteriormente se inserto tambien la informacion al momento de insertar o eliminar el modulo en el buffer de acciones del kernel.

```
static int inicializar(void)
{
    proc_create("memo_201603095", 0, NULL, &operaciones);
    printk(KERN_INFO "CARNET: 201603095\n");

    return 0;
}

static void finalizar(void)
{
    remove_proc_entry("memo_201603095", NULL);
    printk(KERN_INFO "Curso de Sistemas Operativos 1\n");
}
```

2. Api - Backend

Esta api esta hecha en lenguaje Go y sirve para pasar informacion alfrontend del archivo creado en los modulos. Se define que sea abierta en el puerto :5000 del localhost, y las rutas necesarias con las que se comunica el frontend son las siguientes:

```
func main() {
    router := mux.NewRouter()

    router.HandleFunc("/getall", getall).Methods("GET", "OPTIONS")
    router.HandleFunc("/getram", getRAM).Methods("GET", "OPTIONS")
    router.HandleFunc("/getcpuinfo", getCPUInfo).Methods("GET", "OPTIONS")
    router.HandleFunc("/getusername/{uid}", getnameuser).Methods("GET", "OPTIONS")

    router.HandleFunc("/kill/{id}", killing).Methods("GET", "OPTIONS")

    fmt.Println("El servidor esta escuchando en el puerto 5000")
    http.ListenAndServe(":5000", router)
}
```

Siendo estas rutas las necesarias para obtener la informacion de utilizacion de recursos, su funcion esta dada de la siguiente forma

/getall

Obtiene la informacion de los procesos y del cpu

/getram

Obtiene la informacion concerniente a la memoria RAM

/getcpuinfo

Obtiene la informacion del cpu

/getusername/{uid}

Obtiene el nombre de un usuario partiendo de su uid

/kill/{id}

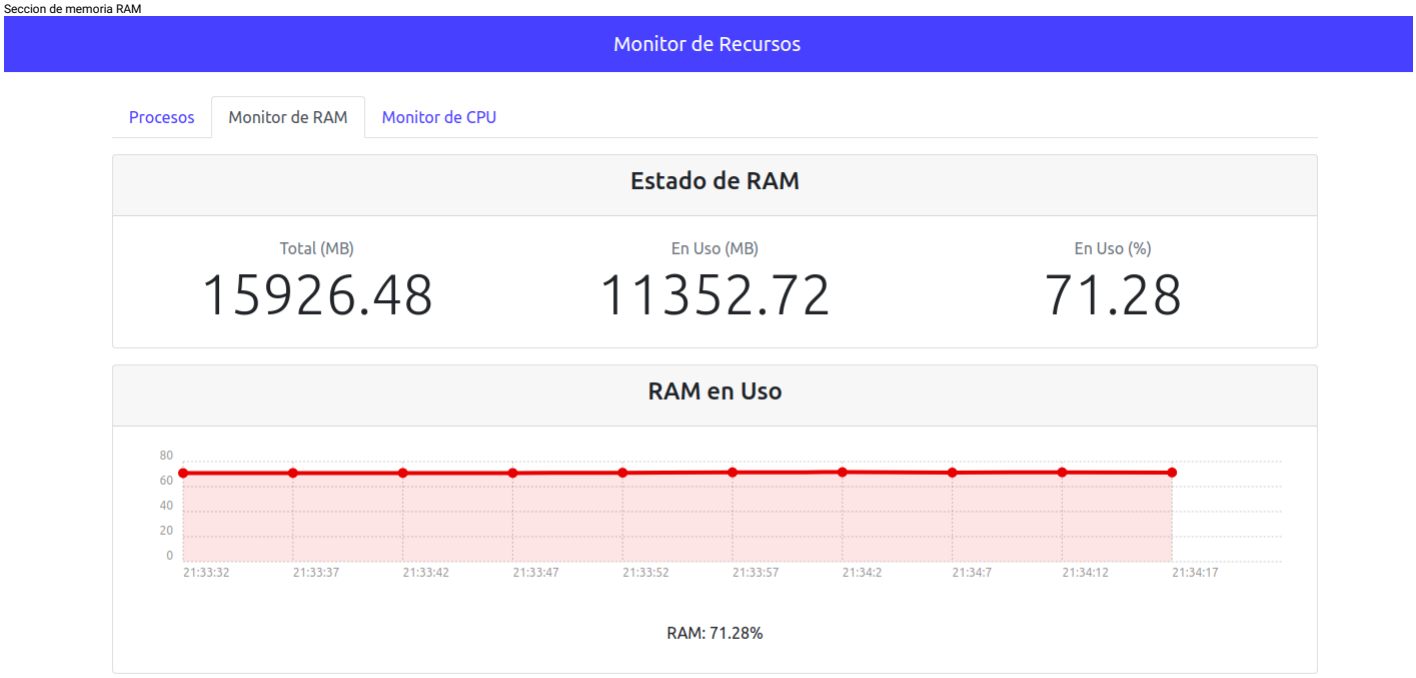
Elimina un proceso de la lista de procesos partiendo de su pid

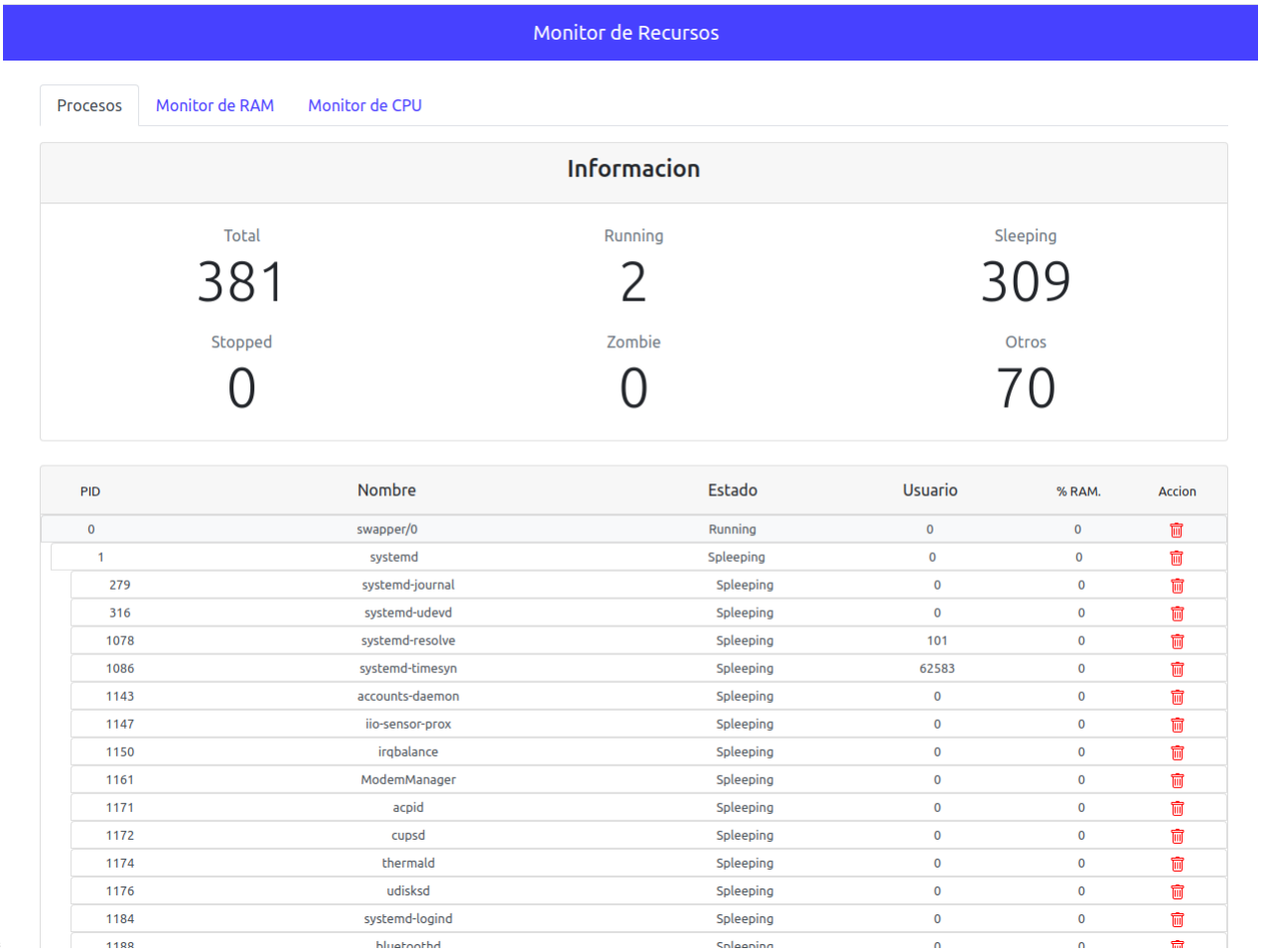
Todos las rutas definidas anteriormente siguen el mismo patron el cual consta de ir a leer el archivo generado por el modulo descrito anteriormente y posteriormente enviar la informacion al frontend (a excepcion de kill y getusername, los cuales ejecutan un comando del sistema para cumplir su objetivo).

3. FrontEnd

El frontend se encarga de precentar los datos de forma legible y clara de tal forma que sea entendible para el usuario comun. Esta hecho sobre las librerias de REACTJS y se ejecuta en el puerto 3000.

Seccion de memoria RAM





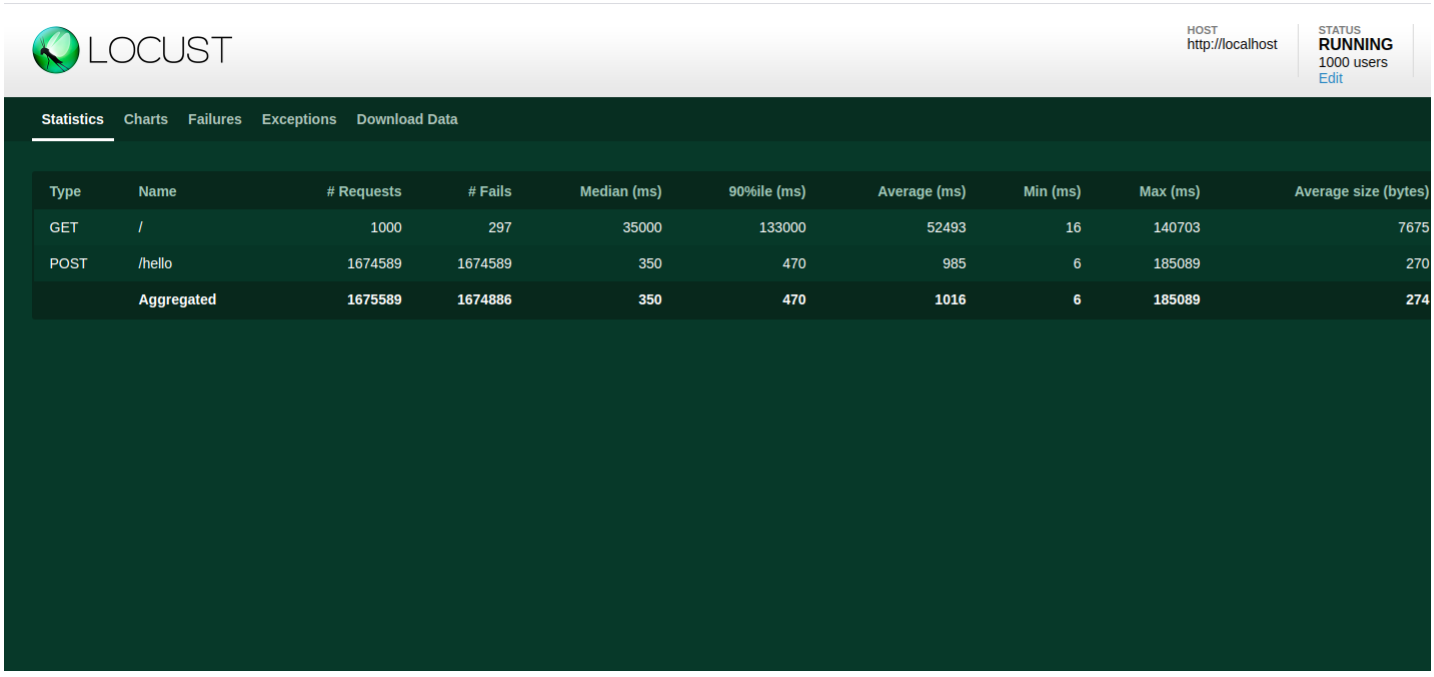
Seccion de Procesos

Seccion de CPU

4. Test

Los test se hicieron en base a la herramienta Locust la cual se encarga de estresar la pagina llenandola de solicitudes, emulando el trafico de diferentes usuarios.

Imagenes de Tests de Trafico con locust



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)
GET	/	1000	297	35000	133000	52493	16	140703	7675
POST	/hello	1674589	1674589	350	470	985	6	185089	270
	Aggregated	**1675589**	**1674886**	**350**	**470**	**1016**	**6**	**185089**	**274**

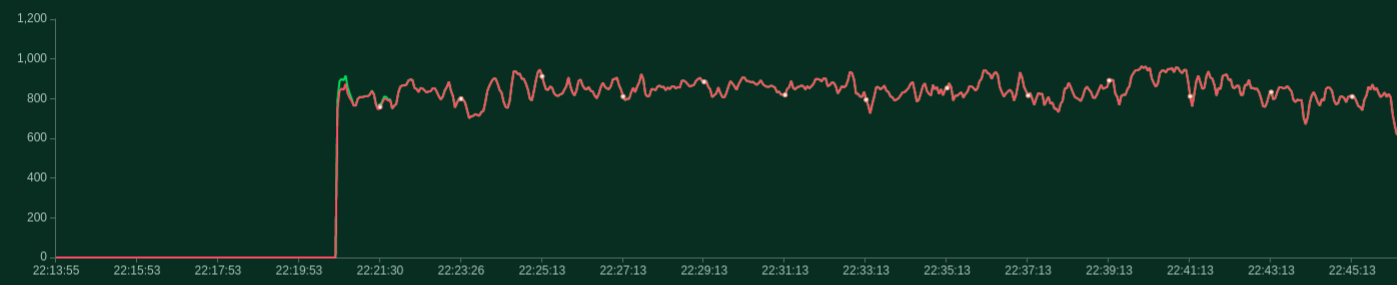
LOCUST

HOST
http://localhost

STATUS
RUNNING
1000 users
Edit

Statistics | **Charts** | Failures | Exceptions | Download Data

Total Requests per Second



Response Times (ms)