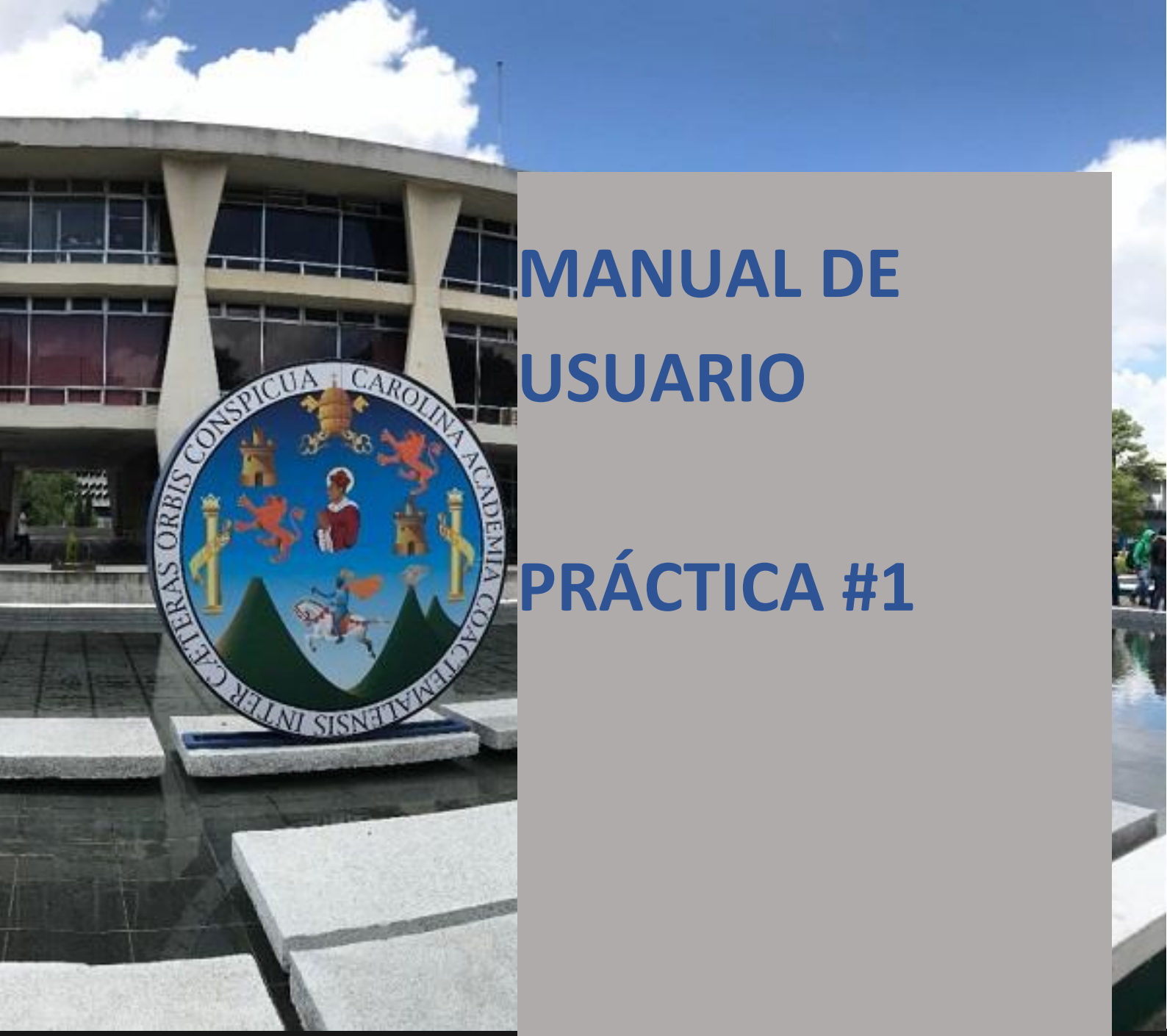


MANUAL DE USUARIO

PRÁCTICA #1

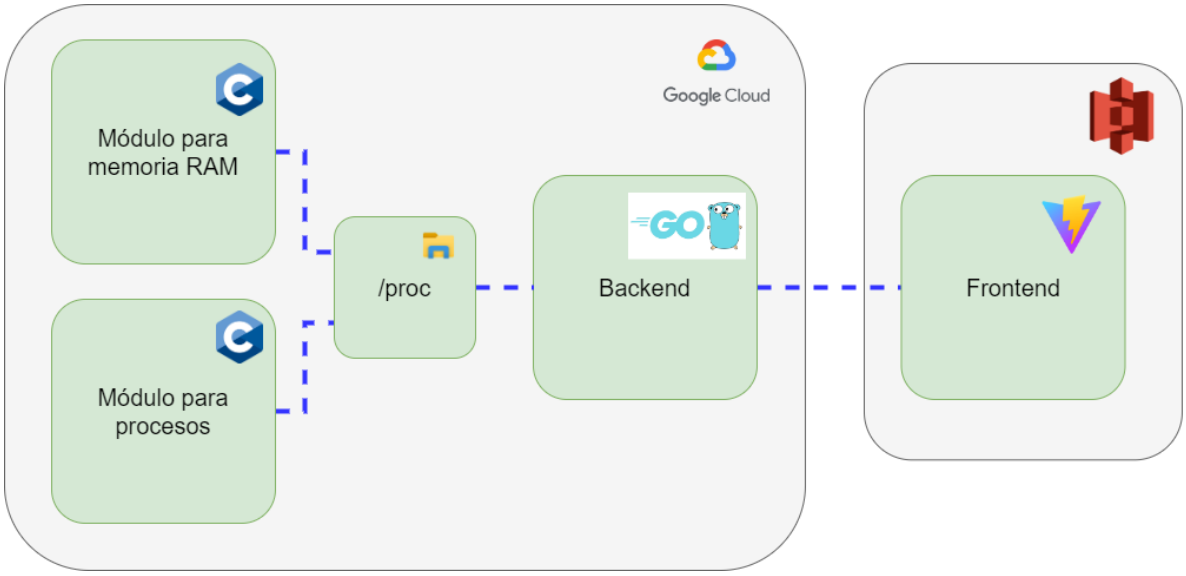
Laboratorio de Sistemas Operativos 2

Alvaro Emmanuel Socop Perez – 202000194
Carlos Daniel Acabal Pérez - 202004724
Javier Alejandro Gutierrez de León - 202004765

Guatemala, 8 de junio de 2023

Introducción

Se creó un sistema que se encarga de monitorear el uso en MB de memoria RAM y los procesos que se encuentran activos en una máquina virtual alojada en una EC2 de AWS, haciendo uso de módulos de kernel para una distribución Debian, escribiendo archivos con esta información en la carpeta /proc para posteriormente leerlos en Golang, donde serán consultados por el Frontend elaborado con Vite donde se muestra la información de forma gráfica.



Objetivos

- ✔ Poner en práctica los conocimientos sobre el Kernel de Linux.
- ✔ Familiarizarse con la terminal de Linux y comandos de sistema y usuario.
- ✔ Aprender a crear, monitorizar y montar procesos del Kernel de Linux.

Especificación Técnica

Requisitos de Hardware y software

- Soporte de kernel y CPU de 64 bits para virtualización.
- Al menos 4 GB de RAM.
- Soporte de virtualización KVM.
- QEMU versión 5.2 o posterior.
- GCC 10.2.1-6
- Distribución de Linux (preferiblemente Debian GNU/Linux 11)

Tecnologías utilizadas

- Debian 11
- Vite 4.3.9
- Golang 1.13

Lógica del programa (Métodos principales)

Módulos de Kernel

Módulo de memoria RAM

Para lograr obtener el porcentaje de memoria RAM utilizado por la máquina virtual se requirió leer "sysinfo", de donde se obtuvo la memoria total y la libre, para que mas adelante sea calculado el porcentaje en el backend de Golang.

```
static int escribir_archivo(struct seq_file *archivo, void *v) {
    // unsigned long long total_memoria = (unsigned long long)totalram_pages
* (unsigned long long)PAGE_SIZE;
    // unsigned long long memoria_libre = (unsigned long
long)si_mem_available();

    struct sysinfo si;
    si_meminfo(&si);

    unsigned long long memoria_total = (unsigned long long)si.totalram *
(unsigned long long)si.mem_unit;
    unsigned long long memoria_usada = memoria_total - (unsigned long
long)si.freeram * (unsigned long long)si.mem_unit;

    seq_printf(archivo, "{\n");
    // printk(KERN_ERR "Memoria total: %llu mB\n", memoria_total
/(1000000));
    // printk(KERN_ERR "Memoria libre: %llu KB\n",
    // si.freeram * (unsigned long long)si.mem_unit /(1000000));
    // printk(KERN_ERR "Buffered: %llu KB\n",
    // (si.bufferram* (unsigned long long)si.mem_unit));
    // printk(KERN_ERR "Memoria en uso: %llu KB\n", memoria_usada
/(1000000));
    seq_printf(archivo, "\"Porcentaje\":%lld \n",
    (((memoria_total)-(si.freeram * (unsigned long long)si.mem_unit) -
(si.bufferram* (unsigned long long)si.mem_unit)- (si.sharedram *(unsigned
long long)si.mem_unit))*10000)/(memoria_total));

    seq_printf(archivo, "}\n");
    return 0;
}
```

Módulo de CPU para procesos del sistema

Para lograr obtener los procesos y sus procesos hijos que son utilizados por el sistema se utilizó "task_struct", obteniendo por cada proceso su PID, nombre, usuario, estado y porcentaje de ram utilizado.

```
static int escribir_archivo(struct seq_file *archivo, void *v)
{
    struct task_struct *task;
    struct task_struct *task_hijo;
    struct list_head *children;
    long memproc;
    long memproc2;
    int indext = 0
    struct file *file;
    struct file *file2;
    char *strstate =
    char buffer[256];
    int len;
    long cpu_usage = 0;
    struct sysinfo info;
    long mem_usage;
    bool first = true;
    long memoria_total = 0;
    long int ejecucion = 0;
    long int suspendido = 0;
    long int detenido = 0;
    long int zombie = 0;
    long int totales = 0;

    long total_time_prev = 0;
    long used_time_prev = 0;

    for_each_process(task)
    {
        total_time_prev += get_total_time(task);
        used_time_prev += task->utime + task->stime;
    }


    msleep(500);

    long total_time = 0;
```

```
    long used_time = 0;

    for_each_process(task)
    {
        total_time += get_total_time(task);
        used_time += task->utime + task->stime;
    }

    long total_time_diff; //= total_time - total_time_prev;
    long used_time_diff;
    if (total_time > total_time_prev)
    {
        if (total_time < total_time_prev)
        {
            total_time_diff = total_time_prev - total_time;
        }
        else
        {
            total_time_diff = total_time - total_time_prev;
        }
        if (used_time < used_time_prev)
        {
            used_time_diff = used_time_prev - used_time;
        }
        else
        {
            used_time_diff = used_time - used_time_prev;
        }

        cpu_usage = (used_time_diff * 100) / total_time_diff;
    }
    printk(KERN_INFO "cpu_usage: %ld%%\n, total_time:
%ld%%\n  total_time_prev: %ld%%\n used_time: %ld%%\n used_time_prev:
%ld%%\n", cpu_usage, total_time, total_time_prev, used_time,
used_time_prev);
    printk(KERN_INFO "total_time_diff: %ld%%\n", total_time_diff);
    printk(KERN_INFO "used_time_diff: %ld%%\n", used_time_diff);

    si_meminfo(&info);
    memoria_total = (info.totalram * info.mem_unit) >> 10;
    seq_printf(archivo, "{\n");
    seq_printf(archivo, "\"cpu_usage\":");
    seq_printf(archivo, "%ld , \n", cpu_usage);
```

```
    seq_printf(archivo, "\"data\": [");
    for_each_process(task)
    {
        if (!first)
        {
            seq_printf(archivo, ",");
        }
        if (task->mm)
        {
            memproc = (get_mm_rss(task->mm) << (PAGE_SHIFT - 10));
            mem_usage = ((memproc * 100) / (memoria_total >> 10));
        }
        if (task->state == 0 || task->state == 1026 || task->state == 2)
        {
            ejecucion++;
            strstate = "ejecucion";
        }
        else if (task->state == 4)
        {
            zombie++;
            strstate = "zombie";
        }
        else if (task->state == 8 || task->state == 8193)
        {
            detenido++;
            strstate = "detenido";
        }
        else if (task->state == 1 || task->state == 1026)
        {
            suspendido++;
            strstate = "suspendido";
        }
        totales++;

        seq_printf(archivo, "{\"id\": \"%d_%s\",\"pid\": %d, \"nombre\":
\"%s\", \"usuario\": \"%d\", \"estado\": \"%s\", \"ram\": %lu,
\n\"procesoshijos\": [",
                   indext,
                   task->comm,
                   task->pid,
                   task->comm,
                   task->cred->uid,
                   strstate, mem_usage);
```

```
        indext++;
        task_lock(task);
        children = &(task->children);
        list_for_each_entry(task_hijo, children, sibling)
        {
            if (task_hijo->mm)
            {
                memproc = (get_mm_rss(task_hijo->mm) << (PAGE_SHIFT - 10));
                mem_usage = ((memproc * 100) / (memoria_total >> 10));
            }

            seq_printf(archivo, "{\"pid\": %d, \"nombre\": \"%s\",
\"usuario\": \"%d\", \"estado\": \"%s\", \"ram\": %lu}",
                       task_hijo->pid,
                       task_hijo->comm,
                       task_hijo->real_cred->uid,
                       strstate,
                       mem_usage);

            if (task_hijo->sibling.next != &task->children)
            {
                seq_printf(archivo, ",");
            }
        }
        task_unlock(task);
        seq_printf(archivo, "]\n}");
        first = false;
    }

    seq_printf(archivo, "], \n");
    seq_printf(archivo, "\"ejecucion\":");
    seq_printf(archivo, "%li , \n", ejecucion);

    seq_printf(archivo, "\"zombie\":");
    seq_printf(archivo, "%li , \n", zombie);

    seq_printf(archivo, "\"detenido\":");
    seq_printf(archivo, "%li , \n", detenido);

    seq_printf(archivo, "\"suspendido\":");
    seq_printf(archivo, "%li , \n", suspendido);

    seq_printf(archivo, "\"totales\":");
```

```
    seq_printf(archivo, "%li  \n", totales);
    seq_printf(archivo, "}");

    return 0;
}
```

Backend de Golang

Obtención de la memoria RAM

Con el siguiente código se busca el archivo creado en /proc por el módulo de kernel al obtener los datos de la memoria RAM

```
func RequestMemory() http.HandlerFunc {
    return func(rw http.ResponseWriter, r *http.Request) {
        salida, _, verificar := CMD("cat /proc/mem_grupo18")

        if verificar != nil {
            log.Printf("error: %v\n", verificar)
        } else {

            var dataJson Models.DATAJSONMEMORY
            json.Unmarshal(salida.Bytes(), &dataJson) //json a objeto
      rw.Header().Set("Content-Type", "application/json")
            json.NewEncoder(rw).Encode(dataJson)
        }
    }
}
```

Obtención de procesos del sistema

Con el siguiente código se busca el archivo creado en /proc por el módulo de kernel al obtener los datos de los procesos del sistema

```
func RequestPrincipal() http.HandlerFunc {
    return func(rw http.ResponseWriter, r *http.Request) {

    rw.Header().Set("Content-Type", "application/json")
        salida, _, verificar := CMD("cat /proc/cpu_grupo18")

        if verificar != nil {
            log.Printf("error: %v\n", verificar)
            } else {
                var dataJson Models.CPUDATAJSON
                json.Unmarshal(salida.Bytes(), &dataJson) //json a objeto
                json.NewEncoder(rw).Encode(dataJson)
        }
    }
}
```

Eliminar procesos

El siguiente código se ejecuta cuando desde la interfaz el usuario desea eliminar uno de los procesos que se están ejecutando, utilizando el comando “sudo kill -9 {PID_proceso}”

```
func RequestKill() http.HandlerFunc {
    return func(rw http.ResponseWriter, r *http.Request) {
        if r.URL.Path != "/Kill" {
            http.NotFound(rw, r)
            return
        }
        if  r.Method =="GET" {
            id := r.URL.Query().Get("pid")
            id = strings.TrimSuffix(id, "/")
      fmt.Println(id)
              _, _, verificar := CMD("sudo kill -9 " + id)
            if verificar != nil {
                log.Printf("error: %v\n", verificar)
            } else {
                fmt.Println("Eliminando Proceso: " + id)
            }
    }else{
```

```
            rw.WriteHeader(http.StatusNotImplemented)
            rw.Write([]byte(http.StatusText(http.StatusNotImplemented)))
    }
    }
}
```