Universidad de San Carlos De Guatemala

Facultad de Ingeniería

Escuela de ciencias y sistemas

MANUAL TECNICO PROYECTO SISTEMAS OPERATIVOS 1

GARY JOAN ORTIZ LOPEZ

200915609

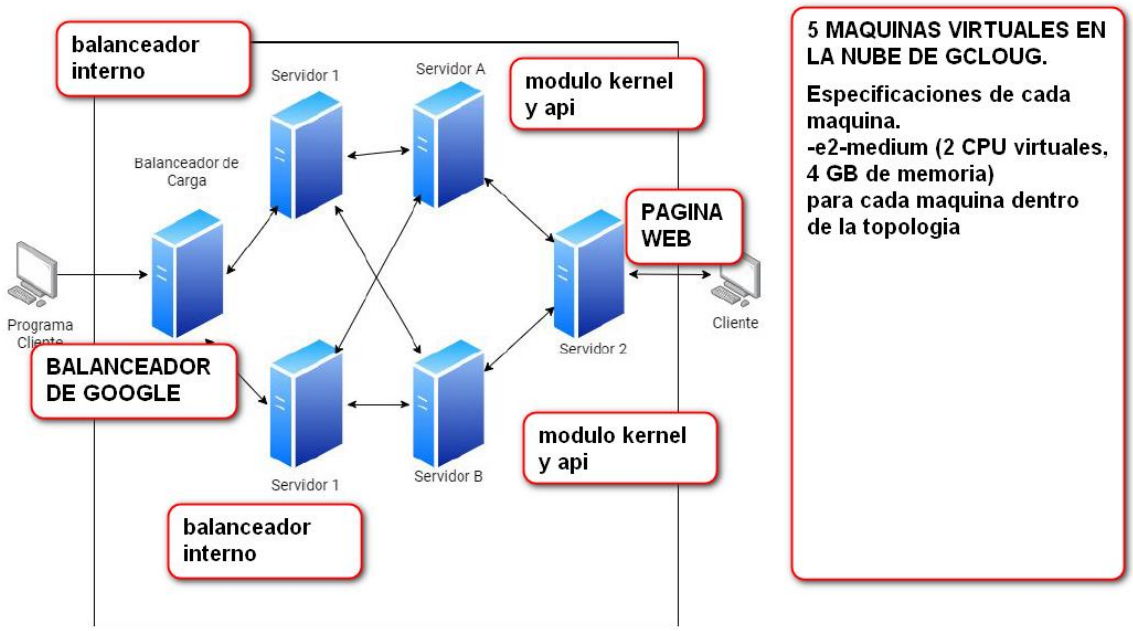
Contenido

INTRODUCCION

El siguiente proyecto se basa en la realización de módulos de kernel para poder obtener información acerca del uso de RAM y CPU por parte del sistema, además también se usar una arquitectura de maquinas virtuales colocas en la nube para poder crear servidores API los cuales nos van a proporcionar la información acerca de los módulos.

REQUERIMIENTOS

Para la realización del a topología de los servidores vamos a necesitar lo siguiente:



En cada instancia de la aplicación esta instalada una instancia de debian así como los header de los kernel para poder usar las librerías.

PROGRAMACION DE UN KERNEL EN LINUX

La programación de un módulo del kernel de Linux no es tan diferente de la programación de cualquier otro software, simplemente los errores se suelen pagar más caros. Vamos a ver el típico Hello World como módulo del kernel:

PASO 1:

Primero de todo deberemos incluir algunos **headers** que contienen definiciones que vamos a necesitar:

```
#include <linux/init.h>
#include <linux/module.h>
```

PASO 2:
A continuación deberemos definir la licencia del modulo:

```
MODULE_LICENSE("GPL");
```

PASO 3
Si nos la inventamos simplemente nos avisara al hacer el insmod:

```
helloworld: module license 'xGPL' taints kernel.
```

PASO 4:
A continuación deberemos definir las funciones de inicialización i destrucción del modulo en las cuales para este sencillo ejemplo haremos un simple printk (equivalente de printf en el kernel):

```
static int hello_world_init(void)
{
	printk(KERN_ALERT "Hello World!\n");
	return 0;
}
```

PASO 5:

En la función de destrucción haremos lo mismo:

```
static void hello_world_exit(void)
{
	printk(KERN_ALERT "Bye World!\n");
}
```

PASO 6:
Finalmente deberemos indicar como hemos llamado a las funciones mediante module_init y module exit:

```
module_init(hello_world_init);
module_exit(hello_world_exit);
```

PASO 7
Finalmente deberemos crear el fichero de Makefile:

```
obj-m += helloworld.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Es obligatorio que antes de los comandos "make" haya un tabulador y no un conjunto de espacios, sino no va a reconocer el formato del Makefile.

COMANDOS PARA INSTALAR MODULOS DE KERNEL

1. Insmod nombre_archivo-ko **:** **comando para montar un modulo en el sistema**
2. Dmesg : **muestra el mensaje de los modulos cargados en el sistema**
3. Cat /proc/nombre_modulo **:** **ejecuta el modulo**
4. Rmmod nombre_modulo **:** **elimina el modulo elegido**

MODULO DE MEMORIA

Ya conocidos los métodos para generar los archivos para la creación de los modulos pasamos a ver la estructura del archivo ara obtener la información acerca de la memoria del cpu.

PASO 1: importamos las librerías necesarias para obtener la información

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <asm/uaccess.h>
/* to get works meminfo */
#include <linux/hugetlb.h>
#include <linux/mm.h>
#include <linux/mman.h>
#include <linux/mmzone.h>
#include <linux/quicklist.h>
#include <linux/swap.h>
#include <linux/vmstat.h>
#include <linux/atomic.h>
#include <asm/page.h>
#include <asm/pgtable.h>
```

PASO 2:

Agregamos las siguientes variables y estructuras a utilizar

```
struct sysinfo i;
unsigned long committed;
unsigned long allowed;
//struct vmalloc_info vmi;
long cached;
unsigned long pages[NR_LRU_LISTS];
int lru;
```

PASO 3:

Aquí tenemos la función que va a recolectar la información acerca de la RAM del servidor

```
static int memori_show(struct seq_file *m, void *v){

int porcentaje = 0;
seq_printf(m,"{");
seq_printf(m,"\"Nombre\": \"Gary Joan Ortiz Lopez\",\n");
seq_printf(m,"\"Carnet\": 200915609 ,\n");
#define K(x) ((x) << (PAGE_SHIFT - 10))
si_meminfo(&i);

porcentaje = (i.freeram*100)/i.totalram;
seq_printf(m,"\"MemTotal\": %8lu ,\n",K(i.totalram));
seq_printf(m,"\"MemFree\": %8lu ,\n",K(i.freeram));
seq_printf(m,"\"Buffers\": %8lu ,\n",K(i.bufferram));
seq_printf(m, "\"Porcentaje Libre\": %8u\n",porcentaje);

#ifdef CONFIG_HIGHMEM
```

Las medidas que tiran son unas medidas propias de la librería por lo tanto usamos la función “K()” para que nos devuelva unas métricas más concisas para poder utilizarla

PASO 4:

Creamos la siguiente estructura para poder recolectar la informacion y pasarla a archivo de modulo

```
static const struct file_operations memori_fops = {
.owner = THIS_MODULE,
.open = memori_open,
.read = seq_read,
.llseek = seq_lseek,
.release = single_release,
};
```

PASO 5:

Este método es para la inicialización del modulo

```
static int __init memori_init(void){
printk(KERN_INFO "Cargando modulo memoria_200915609.\r\n");
proc_create(PROCFS_NAME, 0, NULL, &memori_fops);
printk(KERN_INFO "Nombre : Gary Joan Ortiz Lopez \n Carnet : 200915609 \n Completado. Procceso: /proc/%s.\r\n", PROCF
return 0;
}
```

PASO 6:

Estos modulos son los que se utilizan para ingresar o eliminar los modulos para que no quede nada en el sistema.

```
module_init(memori_init);
module_exit(memori_exit);

MODULE_LICENSE("GPL");
```

MODULO DEL CPU

Se muestra la estructura para poder obtener la informacion acerca del uso del CPU del servidor

PASO 1:

SE carga las librerías necesarias

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <asm/uaccess.h>
#include <linux/sched/signal.h>
#include <linux/sched.h>
#include <linux/ktime.h>
s64  uptime;
static int hz=100;
#define PROCFS_NAME "cpu_200915609"
```

PASO 2:

Este paso es importante porque aquí importamos la estructura de datos de “TASK_STRUCT” que es la encargada de obtener la informacion acerca de los procesos del sistema y sis métricas para poder despues hacer los cálculos para la informacion del cpu.

```
struct task_struct *task;
struct task_struct *task_child;
struct list_head *list;
```

Esta estructura se encuentra dentro de la librería de “sched.h” para poder tomar la información de cada proceso

PASO 3:

Este método es para poder obtener la información usando la estructura de task para hacer los cálculos de cada proceso del sistema

```
static int cpu_show(struct seq_file *m, void *v){
    int total_time = 0;
    int start_time=0;
    int seconds = 0;
    seq_printf(m, "{ \"CPU\": ");
    for_each_process(task){
        uptime = ktime_divns(ktime_get_coarse_boottime(), NSEC_PER_SEC);
        total_time = total_time + task->utime + task->stime;
        start_time = start_time + task->start_time;
        seconds = seconds + (uptime - (start_time / hz));
        list_for_each(list, &task->children){

            task_child = list_entry( list, struct task_struct, sibling );

            total_time = total_time + task_child->utime +task_child->stime;
                }

    }
```

PASO 4:

Se crea la función para inicializar el modulo

```
static int cpu_open(struct inode *inode, struct file *file){
return single_open(file, cpu_show, NULL);
}
```

PASO 5:

Se agregarán las funciones de creación y eliminación del modulo.

```
static int ver_cpu_init(void){
    printk(KERN_INFO "Cargando modulo cpu.\r\n");
    proc_create(PROCFS_NAME, 0, NULL, &cpu_fops);
    printk(KERN_INFO "Nombre : Gary Joan Ortiz Lopez \n C
    return 0;
}

static void ver_cpu_exit(void){

        printk(KERN_INFO "Modulo CPU Deshabilitado.\r\n")
        remove_proc_entry(PROCFS_NAME, NULL);
}

module_init(ver_cpu_init);
module_exit(ver_cpu_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("HB");
MODULE_DESCRIPTION("ejemplo de como menejar cpu");
```

API PYTHON

Para el consumo de información de los módulos se utilizó lo siguiente:

```
@app.route('/')
def home():
   return "<h2>API SERVIDOR A</h2>"

@app.route('/memoria')
def memoria():
    arry =os.popen('cat /proc/memoria_200915609').read()
    memoria = loads(arry)
    return memoria
@app.route('/cpu')
def cpu():
    arry = os.popen('cat /proc/cpu_200915609').read()
    cpu = loads(arry)
    return cpu
```

Aquí se muestras las api para obtener la informacion de cada modulo que se ejecutaron y dar la información correspondiente.

PROGRAMA CLIENTE

El programa clientes es un programa sencillo escrito en Python el cual toma la ruta de un archivo y generar uno objetos de tipo json para enviarlo al balanceador de carga.

Codigo

```
while opcion != "n":
        #inicio del programa que pide la ruta del archivo y la dir
eccion del balanceador de google cloud
        print("PROGRAMA CLIENTE")
        autor = input("Nombre del autor del archivo: ")
        ruta_archivo = input("Ingrese ruta del archivo: ")
        ruta_balanceador = input("Ingrese IP del balanceador con (
https): ")
        archivo = open("hello.txt", 'r')
        contenido = archivo.read()
        #iteramos la lista de oraciones para enviarlos al balancea
dor
        lista_contenido= sent_tokenize(contenido)
        for item in lista_contenido:
            json_publicacion ={
                "autor": autor,
                "nota" : item
            }
            publicacion=json.dumps(json_publicacion)
            newHeaders = {'Content-
type': 'application/json', 'Accept': 'text/plain'}
            try:
                rq = requests.post(ruta_balanceador+'/balanceador'
,data=publicacion,headers=newHeaders)
                print(rq.status_code)
            except requests.exceptions.RequestException as e:
                raise SystemExit(e)
```

BALANCEADOR INTERNO

Para el balanceador tomamos la informacion de las API de cpu y memoria y de la base de datos para poder hacer las comparaciones necesaria para poder enviar los paquetes al servidor correspondiente.

RUTA BALANCEADOR:

```
@app.route('/balanceador',methods=['POST'])
def loadB():
    data=request.get_json()
    #print(data,flush=True)
#info acerca de ambos servidores
    #info servidor 1
    memoria_servidor1 =requests.get(Servidor1_url+'/memoria')
    cpu_servidor1 = requests.get(Servidor1_url+'/cpu')
    #info servidor 2
    memoria_servidor2 =requests.get(Servidor2_url+'/memoria')
    cpu_servidor2 = requests.get(Servidor2_url+'/cpu')
```

Obtenemos la informacion de los servidores tanto el cpu como la memoria.

COMPARACIONES

Se muestra una porción del código de comparación para enviar los paquetes según las compraciones necesarias

```
 if(countS1 > countS2):
        print("Insertar en B POR COUNT")

        try:
            rq = requests.post(Servidor2_url+'/new',data=dumps(dat
a),headers=newHeaders)
            print(rq.status_code)
        except requests.exceptions.RequestException as e:
                raise SystemExit(e)
    elif (countS1 < countS2):
        print("Insertar en A POR COUNT")
        try:
            rq = requests.post(Servidor1_url+'/new',data=dumps(dat
a),headers=newHeaders)
```

```
            print(rq.status_code, flush=True)
        except requests.exceptions.RequestException as e:
                raise SystemExit(e)
    elif (countS1== countS2):
#SI EL NUMERO ES IGUAL SE TOMA LA RAM
        if(porcentaje_utilizacion_servidor1>porcentaje_utilizacion
_servidor2):
```