



Manual Técnico

Sistemas Operativos 2

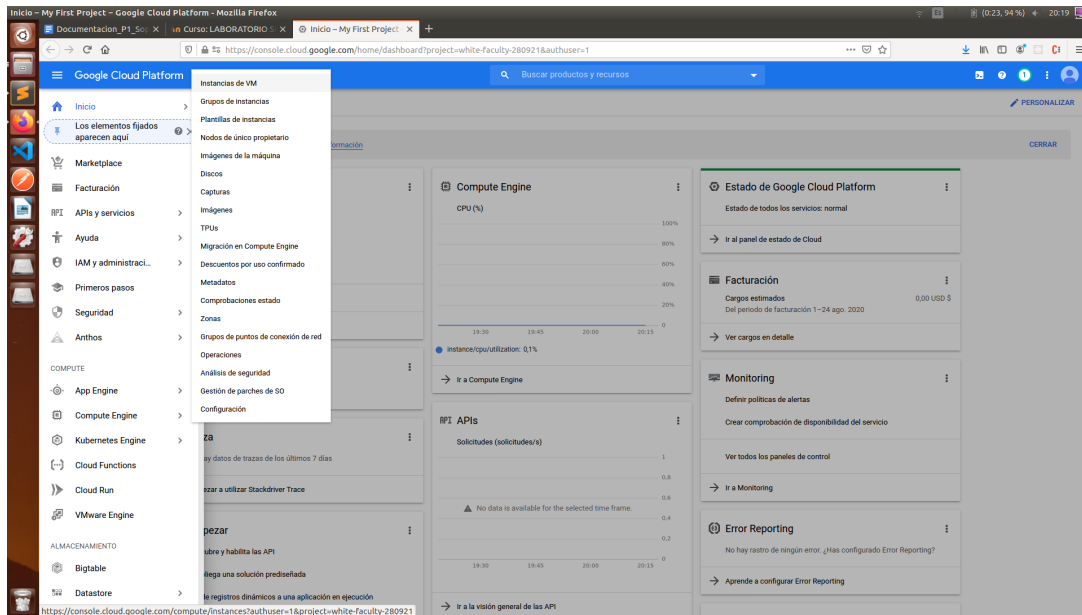
Integrantes

Ruben Emilio Osorio Sotorro	201403703
Brandon Bredly Alvarez López	201403862
Yoselin Annelice Lemus López	201403819

MÓDULOS

I. Creando máquina virtual de Linux en Google Cloud

Estando dentro de Google Cloud nos dirigimos a “Compute Engine” y seleccionamos Instancias de VM.



Luego, en la parte superior elegimos la opción de “Crear Instancia”, y se configuró de la siguiente manera, es necesario darle mínimo 100GB de almacenamiento a la máquina virtual para la compilación del Kernel:

Nombre ⓘ
El nombre es permanente.

Etiquetas ⓘ (Opcional)

Región ⓘ
La región es permanente.

Zona ⓘ
La zona es permanente.

Configuración de la máquina

Familia de máquinas

☒ **Uso general** ☐ Con memoria optimizada
☐ Optimizada para la computación

Tiempo de máquinas para cargas de trabajo habituales, optimizadas en cuanto al costo y a la flexibilidad

Serie

Con la tecnología de la plataforma de CPU Intel Skylake o de uno de sus predecesores

Tipo de máquina

	vCPU	Memoria	GPUs
	1	3,75 GB	-

[Plataforma de CPU y GPU](#)

Servicio de VM confidencial ⓘ
☐ Habilitar el servicio de computación confidencial en esta instancia de VM.

Contenedor ⓘ
☐ Desplegar una imagen de contenedor en esta instancia de VM. [Más información](#)

Disco de arranque

Select an image or snapshot to create a boot disk; or attach an existing disk. Can't find what you're looking for? Explore hundreds of VM solutions in [Marketplace](#).

Imágenes públicas

Imágenes personalizadas

Capturas

Discos disponibles

Sistema operativo

Ubuntu

Versión

Ubuntu 18.04 LTS


amd64 bionic image built on 2020-08-21, supports Shielded VM features

Tipo de disco de arranque

Tamaño (GB)


Disco persistente estándar

100



Nuevo disco persistente estándar de 100 GB

Imagen

 Ubuntu 18.04 LTS

Cambiar

Identidad y acceso de API

Cuenta de servicio

Compute Engine default service account

Alcance del acceso

☒ Permitir el acceso predeterminado

☐ Permitir el acceso completo a todas las API de Cloud

☐ Definir acceso para cada API

Cortafuegos

Añade reglas de cortafuegos y etiquetas para permitir tráfico de red concreto de Internet

☒ Permitir el tráfico HTTP

☒ Permitir el tráfico HTTPS

Administración

Seguridad

Discos

Redes

Único cliente

Etiquetas de red

todoind

todoout

Nombre de host

Especifica un nombre de host personalizado para la instancia o deja la opción predeterminada. Esta elección es permanente.

ubuntu.us-central1-a.c.white-faculty-280921.internal

Interfaces de red

La interfaz de red es permanente.

default default (10.128.0.0/20)

Las etiquetas de red “todoin” y “todoout” son firewalls que permiten todo el tráfico hacia esa máquina virtual.

Una vez creada aparecerá de la siguiente manera, luego seleccionamos la flecha que está a un lado de SSH y elegimos "Ver comando gcloud"

Instancias de VM

CREAR INSTANCIA

IMPORTAR VM

ACTUALIZAR

INICIAR/REANUDAR

Filtrar las instancias de VM

Columnas

<input type="checkbox"/>	Nombre ^	Zona	Recomendación	Usada por	IP interna	IP externa	Conectar
<input type="checkbox"/>	ubuntu	us-central1-a			10.128.0.36 (nic0)	35.232.161.193	SSH

Línea de comandos gcloud

La siguiente línea de comandos gcloud se puede utilizar para aplicar SSH en esta instancia.

```
gcloud beta compute ssh --zone "us-central1-a" "ubuntu" --project "white-faculty-280921"
```

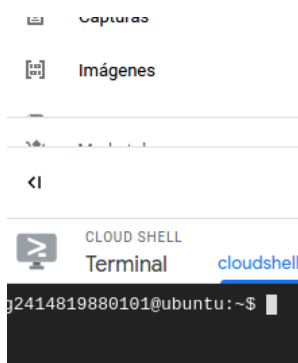
☒ Ajuste de línea

[Referencias de gcloud](#)

[CERRAR](#)

[EJECUTAR EN CLOUD SHELL](#)

Seleccionamos “Ejecutar en Cloud Shell” para entrar a la consola de Cloud de nuestra máquina virtual.



II. Compilación del Kernel de Linux

1) Para ver la versión actual del kernel ingresamos:

uname -a

2) Descargamos la versión actual del kernel (la 5.8.1 es la más reciente)

wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.8.1.tar.xz

3) Instalamos algunos archivos que son prerequisites para la compilación del kernel

sudo apt-get install git fakeroot build-essential ncurses-dev xz-utils libssl-dev bc flex libelf-dev bison

4) Descomprimos el archivo que descargamos del kernel

tar Jxvf linux-5.8.1.tar.xz

5) Ingresamos a la carpeta que se creó tras descomprimir

cd linux-5.8.1.tar.xz

6) Configuramos los módulos a incluir durante la compilación con los ya existentes

cp /boot/config-\$(uname -r) .config

7) Verificamos que estemos utilizando el archivo .config

make menuconfig

Se mostrará una ventana azul la cual, seleccionamos hasta abajo donde dice LOAD, ahí debe aparecer ese archivo, guardamos y salimos.

8) Inicializamos la compilación del kernel

make

Este proceso puede ser extenso, alrededor de 3 horas aproximadamente.

9) Luego de finalizado, instalamos los módulos

sudo make modules_install

10) Instalamos el nuevo kernel

sudo make install

11) Habilitamos el kernel para el arranque con

sudo update-initramfs -c -k 5.8.1

12) Actualizamos el grub

sudo update-grub

Luego reiniciamos la máquina

Una vez dentro verificamos nuevamente la versión del kernel, que coincide con la que descargamos.

Google Cloud Platform

My First Project

Compute Engine

Instancias de VM

CREAR INSTANCIA

IMP

Instancias de VM

Grupos de instancias

Plantillas de instancias

Nodos de único propietario

Imágenes de la máquina

Discos

Capturas

Imágenes

Filtrar las instancias de VM

<input type="checkbox"/>	Nombre ^	Zona	Recomendación	Usada por	IP
<input type="checkbox"/>	ubuntu	us-central1-a			10

Acciones relacionadas

Ver Informe de facturación

Ver y gestionar la facturación de Compute Engine

Monitor VMs

View outlier VMs a CPU and Network

CLOUD SHELL

Terminal

cloudshell

+

g2414819880101@ubuntu:~\$ uname -a

Linux ubuntu 5.8.1 #1 SMP Sat Aug 15 03:53:05 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux

g2414819880101@ubuntu:~\$

III. Configuración del módulo del CPU

Creamos un archivo con extensión .c con las siguientes importaciones, las estructuras necesarias para los mensajes de los procesos, así como las variables necesarias para la generación del archivo de salida.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/list.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/string.h>
#include <linux/fs.h>
#include <linux/seq_file.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/hugetlb.h>
#include <linux/sched/signal.h>
#include <linux/uidgid.h>

#define modulo_cpu "cpu_grupo14"

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

/*INFO DEL MODULO DE CPU*/
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Yoselin Lemus 201403819 - Brandon Alvarez 201403862 - Ruben Osorio 201403703");
MODULE_DESCRIPTION("Modulo con descripción del CPU");

int contadorPadre = 1;
int contadorHijos = 1;
```

Función que utiliza para escribir en el archivo "cpu_grupo14", que se genera cuando se carga el módulo del CPU. El archivo de salida se genera en formato JSON para su lectura dentro de la API, el cual contiene información del ID del proceso, UID del usuario que cargó el módulo, el nombre del proceso, su estado y sus procesos hijos (si poseen), así también su ID, su nombre y su estado. Se utilizaron variables int como contadores para facilitar la escritura en formato JSON.

```

static int escribiendoArchivo(struct seq_file *mfile, void *v)
{
    seq_printf(mfile, "{\n");
    seq_printf(mfile, "    \"cpu\" : {\n");
    contadorPadre = 1;
    for_each_process(task)
    {
        if(contadorPadre == 1){
            seq_printf(mfile, "        {\n");
        }
        else{
            seq_printf(mfile, ",\n");
            seq_printf(mfile, "        {\n");
        }
        seq_printf(mfile, "            \"pid\" : %d,\n", task->pid);
        seq_printf(mfile, "            \"uid\" : %d,\n", current_uid());
        seq_printf(mfile, "            \"nombre\" : \"%s\",\n", task->comm);
        seq_printf(mfile, "            \"estado\" : %ld,\n", task->state);
        seq_printf(mfile, "            \"hijos\" : {\n");

        contadorHijos = 1;

        list_for_each(list, &task->children)
        {
            task_child = list_entry(list, struct task_struct, sibling);

            if(contadorHijos == 1){
                seq_printf(mfile, "                {\n");
            }
            else{
                seq_printf(mfile, ",\n");
                seq_printf(mfile, "                {\n");
            }
            seq_printf(mfile, "                    \"pid\" : %d,\n", task_child->pid);
            seq_printf(mfile, "                    \"nombre\" : \"%s\",\n", task_child->comm);
            seq_printf(mfile, "                    \"estado\" : %ld,\n", task_child->state);
            seq_printf(mfile, "                }");
            contadorHijos++;
        }
        seq_printf(mfile, "\n");
        seq_printf(mfile, "            }\n"); //es para terminar los hijos
        seq_printf(mfile, "        }");
        contadorPadre++;
    }
    seq_printf(mfile, "\n");
    seq_printf(mfile, "    }\n");
    seq_printf(mfile, "}");
    return 0;
}

```

Función que actualiza el contenido del archivo cpu_grupo14 dentro de la carpeta /proc/ y la estructura necesaria para las operaciones de archivo al momento de cargarse el módulo

```

static int alAbrirArchivo(struct inode *inodo, struct file *mfile)
{
    return single_open(mfile, escribiendoArchivo, NULL);
}

```

```

static struct proc_ops operacionesDeArchivo={
    .proc_open = alAbrirArchivo,
    .proc_release = single_release,
    .proc_read = seq_read,
    .proc_lseek = seq_lseek,
};

```


La función `iniciandoCPU` es la que se ejecutará cuando se cargue el módulo del CPU, guardando en el buffer de `dmesg` la cadena de los nombres de integrantes del grupo. Y el método `terminandoCPU` se ejecuta cuando se descarga el módulo del CPU, guardando en el buffer el mensaje "Sistemas Operativos 2". Es necesario añadirle el salto de línea para que al momento de cargar el módulo y usar el comando "`dmesg`" se muestre el mensaje respectivo, de lo contrario no lo mostrará.

```
static int iniciandoCPU(void)
{
    proc_create(modulo_cpu, 0, NULL, &operacionesDeArchivo);
    printk(KERN_INFO "Yoselin Lemus - Brandon Alvarez - Ruben Osorio\n");
    return 0;
}

static void terminandoCPU(void)
{
    remove_proc_entry(modulo_cpu, NULL);
    printk(KERN_INFO "Sistemas Operativos 2\n");
}
```

Se mandan a iniciar los módulos, así como su finalización.

```
module_init(iniciandoCPU);
module_exit(terminandoCPU);
```

IV. Configuración del módulo de memoria

Creamos un archivo con extensión .c con las siguientes importaciones, las estructuras necesarias para los mensajes de los procesos, así como las variables necesarias para la generación del archivo de salida.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/list.h>
#include <linux/types.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/string.h>
#include <linux/fs.h>
#include <linux/seq_file.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/hugetlb.h>

#define modulo_memoria "mem_grupo14"

struct sysinfo informacion;

/*INFO DEL MODULO DE MEMORIA*/
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Yoselin Lemus 201403819 - Brandon Alvarez 201403862 - Ruben Osorio 201403703");
MODULE_DESCRIPTION("Modulo con descripción de la memoria RAM");
```

Función encargada de escribir en el archivo “mem_grupo14” dentro de la carpeta /proc para mostrar la información de la memoria RAM en formato JSON

```
static int escribiendoArchivo(struct seq_file *mifile, void *v){
    #define S(x) ((x) << (PAGE_SHIFT -10))
    si_meminfo(&informacion);
    seq_printf(mifile, "{\n");
    seq_printf(mifile, "    \"memoria_total_mb\" : %lu,\n", S(informacion.totalram/1024));
    seq_printf(mifile, "    \"memoria_consumida_mb\" : %lu,\n", S(informacion.totalram/1024) - S(informacion.freeram/1024));
    seq_printf(mifile, "    \"memoria_utilizada_porcentaje\" : %lu}\n", 100 - S((informacion.freeram*100)/S(informacion.totalram)));
    return 0;
}
```

Función que actualiza el contenido del archivo mem_grupo14 dentro de la carpeta /proc/ y la estructura necesaria para las operaciones de archivo al momento de cargarse el módulo

```
static int alAbrirArchivo(struct inode *inodo, struct file *mifile){
    return single_open(mifile, escribiendoArchivo, NULL);
}

static struct proc_ops operacionesDeArchivo={
    .proc_open = alAbrirArchivo,
    .proc_release = single_release,
    .proc_read = seq_read,
    .proc_lseek = seq_lseek,
};
```

La función `iniciandoModulo` es la que se ejecutará cuando se cargue el módulo de memoria, guardando en el buffer de `dmesg` la cadena "Hola mundo...". Y el método `finalizandoModulo` se ejecuta cuando se descarga el módulo de memoria, guardando en el buffer el mensaje "Sayonara mundo...". Es necesario añadirle el salto de línea para que al momento de cargar o descargar el módulo y usar el comando "dmesg" se muestre el mensaje respectivo, de lo contrario no lo mostrará.

```
static int iniciandoModulo(void)
{
    proc_create(modulo_memoria, 0, NULL, &operacionesDeArchivo);
    printk(KERN_INFO "Hola mundo, somos el grupo 14 y este es el monitor de memoria\n");
    return 0;
}

static void finalizandoModulo(void)
{
    remove_proc_entry(modulo_memoria, NULL);
    printk(KERN_INFO "Sayonara mundo, somos el grupo 14 y este fue el monitor de memoria\n");
}
```

V. Configuración del archivo Makefile

Crearemos un archivo llamado “Makefile” el cual nos servirá para compilar y generar los módulos

Comando: **make**

Compila y crea los archivos necesarios en la misma ruta donde se encuentran los archivos .c

Comando: **make clean**

Elimina los archivos que se generaron a partir del comando anterior.

```
obj-m += mem_grupo14.o
obj-m += cpu_grupo14.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
```

VI. Cargando módulos de memoria y CPU

Luego de hacer el comando **make** se generarán los siguientes archivos

```
g24148198801@ubuntu:/home/ubuntu/practical1/Sopes2_Practical1/modulos$ ls
Ej_Salida.Cpu.json  Makefile      README.md      cpu_grupo14.ko  cpu_grupo14.mod.c  cpu_grupo14.o  mem_grupo14.ko  mem_grupo14.mod.c  mem_grupo14.o
Ej_Salida.Mem.json  Module.symvers  cpu_grupo14.c  cpu_grupo14.mod  cpu_grupo14.mod.o  mem_grupo14.c  mem_grupo14.mod  mem_grupo14.mod.o  modules.order
g24148198801@ubuntu:/home/ubuntu/practical1/Sopes2_Practical1/modulos$
```

Para cargar el módulo de CPU utilizamos el comando

sudo insmod cpu_grupo14.ko

Luego utilizamos el comando **dmesg** para ver mensaje que se genera al cargar el módulo

Para cargar el módulo de memoria utilizamos el comando

sudo insmod mem_grupo14.ko

Usando **dmesg** podemos ver el mensaje que se genera al cargar el módulo. Luego dentro de la carpeta /proc deben haber 2 archivos llamados “cpu_grupo14” y “mem_grupo14”

```
g24148198801@ubuntu:/home/ubuntu/practical1/Sopes2_Practical1/modulos$ cd /proc/
g24148198801@ubuntu:/proc$ ls
1      113  17  21916 268  4  411 4920 5581 6010 698  738  76  79  86  91  consoles  driver  iomem  kmsg  mem_grupo14  pagetypeinfo  slabinfo  thread-self  zoneinfo
10     12  170  22  3  400  420 4922 5582 6017 700  74  760  80  854  93  cpu_grupo14  dynamic debug  ioports  kpagecgroup  meminfo  partitions  softirqs  timer_list
100    13  18  23  321  401 422 4923 5671 630  708  749  761  841  856  acpi  cpufreq  execdomains  irq  kallsyms  kpagecount  misc  pressure  stat  tty
104    14  19  24  322 402 424 5036 5672 633  72  748  763  842  88  buddyinfo  crypto  fb  kallsyms  kpageflags  modules  sched_debug  swaps  uptime
107    15  2  29  354 403 426 5038 5687 684  720  749  77  843  89  bus  devices  filesystems  kcore  loadavg  mounts  schedstat  sys  version
108    15580 20 2528 390 404 428 5470 5933 691  73  75  772  844  9  cgroups  diskstats  fs  key-users  locks  mtrr  net  self  sysrq-trigger  vmallocinfo
11     16  21  26  391 406 434 5579 6  694  735  759  78  85  90  cmdline  dma  interrupts  keys  mdstat  net  self  sysvipc  vmstat
g24148198801@ubuntu:/proc$
```

El cual haciendo un **cat mem_grupo14** o **cat cpu_grupo14** podremos visualizar el contenido del archivo.

```
g2414819880101@ubuntu:/proc$ cat mem_grupo14
{
  "memoria_total_mb" : 3680,
  "memoria_consumida_mb" : 676,
  "memoria_utilizada_porcentaje" : 19
}g2414819880101@ubuntu:/proc$
```

```
g2414819880101@ubuntu:/proc$ cat cpu_grupo14
{
  "cpu" : [
    {
      "pid" : 1,
      "uid" : 1002,
      "nombre" : "systemd",
      "estado" : 1,
      "hijos" : [
        {
          "pid" : 400,
          "nombre" : "systemd-journal",
          "estado" : 1
        },
        {
          "pid" : 411,
          "nombre" : "lvmetad",
          "estado" : 1
        },
        {
          "pid" : 420,
          "nombre" : "systemd-udev",
          "estado" : 1
        }
      ]
    }
  ]
}
```

Para eliminar dichos módulos utilizamos los comandos

sudo rmmod cpu_grupo14

sudo rmmod mem_grupo14

Luego podremos visualizar los mensajes de salida en el buffer con el comando **dmesg**

API

Golang 1.14.6

Descripción de librerías Importadas

"fmt": El paquete fmt implementa E / S formateado con funciones análogas a printf y scanf de C. El formato 'verbos' se deriva de C pero es más simple.

"io/ioutil": El paquete ioutil implementa algunas funciones de utilidad de E / S.

"net/http": El paquete http proporciona implementaciones de servidor y cliente HTTP.

"sort": El paquete provee funciones de ordenamientos.

"strconv" : El paquete proporciona herramientas para la conversión de caracteres a números enteros.

"strings": Para el manejo de cadenas de caracteres

"encoding/json": El paquete proporciona funcionalidades para la lectura de formato json.

"github.com/tidwall/gjson": El paquete proporciona funcionalidades para la lectura de formato json.

"github.com/gorilla/mux": El paquete contiene funcionalidades necesarias para levantar un API con varios tipos de peticiones web.

Definición de rutas:

```
router.HandleFunc("/PROCESS", lista_procesos)
router.HandleFunc("/RAM", memoria_proceso)
router.HandleFunc("/kill/{id}", kill_proceso)
router.HandleFunc("/Arbol", arbol_procesos)
```

Descripción de métodos

@param: http.ResponseWriter

@param: *http.Request

@return: JSON

memoria_proceso(w http.ResponseWriter, r *http.Request):

Función que permite obtener la memoria RAM de la máquina virtual, leyendo el archivo mem_grupo14 escrito en el directorio /proc

formato de salida:

```
info_ram := RAM{
    Total_Ram_Servidor:  MemTotal_,
    Total_Ram_Consumida:  MemConsumida,
    Porcentaje_Consumo_Ram: PorcentajeConsumo,
}
```

@param: http.ResponseWriter

@param: *http.Request

@return: JSON

lista_procesos (w http.ResponseWriter, r *http.Request)

Función que permite obtener el listado de procesos escritos en un archivo json en el directorio /proc,

formato de salida:

```
info_general := Info_general{
    Procesos_en_ejecucion: librerias.NumeroRun,
    Procesos_suspendidos: librerias.NumeroSleep,
    Procesos_detenidos:  librerias.NumeroStop,
    Procesos_zombie:    librerias.NumeroZombie,
    Total_procesos:      len(arr_process),
    List_Procesos:       arr_process,
}
```

@param: http.ResponseWriter

@param: *http.Request

@return: Redirect

kill_proceso (w http.ResponseWriter, r *http.Request)

Uno de los metodos principales que hace uso de las librerias creadas matarProceso(), obtiene un valor de id desde frontend.

formato de salida:

```
http.Redirect(w, r, "/public/Principal.html", http.StatusFound)
```

@param: http.ResponseWriter

@param: *http.Request

@return: JSON

arbol_procesos(w http.ResponseWriter, r *http.Request)

Método que obtiene la lista de procesos en forma de arbol.

Formato de salida:

```
JSON_Data, _ := json.Marshal(info_tree)
```

```
w.Write(JSON_Data)
```

Estructuras Utilizadas:

```
type RAM struct
```

```
}
```

```
    Total_Ram_Servidor    int
```

```
    Total_Ram_Consumida   int
```

```
    Porcentaje_Consumo_Ram float32
```

```
}
```

```
type PROCESS struct {
```

```
    PID          string `json:"pid"`
```

```
    Nombre       string
```

```
    `json:"nombre"`
```

```
    Usuario      string
```

```
    Estado       string
```

```
    `json:"estado"`
```

```
    PorcentajeRAM string
```

```
    Proceso_padre string
```

```
    hijos        []PROCESS
```

```
    `json:"hijos"`
```

```
}
```

```
type Info_general struct {
```

```
    Procesos_en_ejecucion int
```

```
    Procesos_suspendidos  int
```

```
    Procesos_detenidos     int
```

```
    Procesos_zombie        int
```

```
    Total_procesos         int
```

```
    List_Procesos          []PROCESS
```

```
}
```



```
type Tree struct {  
    Arbol string  
}  
  
type CPU struct {  
    procesos []PROCESS  
}
```

STRESS LINUX

Es una herramienta para imponer cargas y sistemas de prueba de esfuerzo

Para instalarlo en el sistema nada más se utiliza:

```
sudo apt install stress
```

Se puede utilizar el comando a continuación para correr las pruebas de stress.

```
stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 10s
```

El anterior comando especifica la cantidad de carga por segundo que tendrá el cpu y la memoria RAM por un estimado de 10 segundos, ese valor puede cambiar dependiendo de la capacidad del sistema donde se aplique.