Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Sistemas Operativos 1

Manual Usuario Proyecto #1

Helen Janet Rodas Castro - 202200066

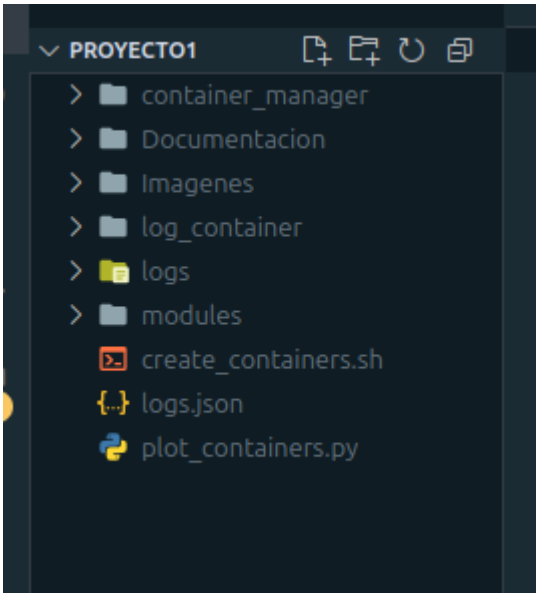
Guatemala 14 de marzo 2025

Introducción

Con la ayuda de este gestor de contenedores se podrá observar de manera más detallada los recursos y la representación de los contenedores a nivel de procesos de Linux y como de manera flexible pueden ser creados, destruidos y conectados por otros servicios.

➔ Estructura del proyecto

El programa contiene una estructura donde esta dividida por carpetas, unas especificas para rust, otras para python, tambien el create_containers.sh que es el encargado de crear los contenedores y modules que esta todo lo relacionado al kernel.



➔ Programas utilizados

- **Docker:**
  Instalación:
  sudo apt update
  sudo apt install docker.io -y
  sudo systemctl start docker
  sudo systemctl enable docker
  Uso: Crear y gestionar contenedores. Se crean contenedores con stress en RAM, CPU, I/O y disco.

  Comandos usados:
  docker ps -a: Lista todos los contenedores
  docker run: Crea y ejecuta contenedores
  docker rm <nombre del contenedor>: Elimina contenedores (en Rust).

- **Rust y Cargo**:

Instalación:
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh source $HOME/.cargo/env

Uso: Compilar y ejecutar el programa en Rust (container_manager).

Comandos usados:

cargo run: Compila y ejecuta el proyecto Rust

- **Cron**

  Instalación:

  Viene preinstalado para linux

  Uso: Programar tareas para ejecutar create_containers.sh cada 30 segundos.

  Comandos usados:

  crontab -e: Edita el archivo de cron para el usuario actual

  crontab -l: Lista las tareas programadas

  sudo crontab -e: Edita el cron de root

➔ Modo de uso

Para ejecutar la aplicación se necesita seguir una serie de pasos y comandos a realizar

1. Configuración del Contenedor de Logs (log_container)

1. **Crear el contenedor `log_manager`:**
   sudo docker run -d --name log_manager_202200066 -p 8080:8080 -v ~/logs:/app/logs log_container_image
2. **Conectar el contenedor con el servidor:**
   curl -X POST http://localhost:8080/logs -H "Content-Type: application/json" -d '[{"container_id": "test", "category": "ram", "created_at": "2025-03-10T00:00:00Z", "deleted_at": null}]'
3. **Verificar que el contenedor esté en ejecución:**
   sudo docker ps -a

4. **Eliminar un contenedor específico:**
   sudo docker rm <id_contenedor>
5. **Eliminar todos los contenedores:**
   sudo docker rm -f $(sudo docker ps -aq)

2. Configuración del Módulo del Kernel (`modules`)

1. **Compilar el módulo del kernel:**
   make
2. **Insertar el módulo en el kernel:**
   sudo insmod sysinfo.ko
3. **Verificar la salida del módulo:**
   cat /proc/sysinfo_202200066
4. **Limpiar y eliminar el módulo:**
   - Limpiar los archivos generados por `make`:
     make clean
   - Eliminar el módulo del kernel:
     sudo rmmod sysinfo

3. Ejecución del Administrador de Contenedores (`container_manager`)

1. **Ejecutar el administrador de contenedores:**
   cargo run

4. Configuración del Entorno Virtual de Python (`Proyecto1`)

1. **Activar el entorno virtual:**
   source ~/matplotlib_env/bin/activate
2. **Ejecutar el script de Python para graficar:**
   python plot_containers.py
3. **Desactivar el entorno virtual:**
   deactivate

➔ Ejemplo de uso

Se inicia creando el contenedor que se usa como parte del servidor:

```
helen@lenovo:~/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/log_container$ sudo docker run -d --name log_manager_
202200066 -p 8080:8080 -v ~/logs:/app/logs log_container_image
```

Luego en el siguiente comando se observa que ya fue un éxito de conexión:

```
helen@lenovo:~/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/log_container$ curl -X POST http://localhost:8080/log
s -H "Content-Type: application/json" -d '[{"container_id": "test", "category": "ram", "created_at": "2025-03-10T00:00
:00Z", "deleted_at": null}]'
{"status":"success"}helen@lenovo:~/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/log_container$
```

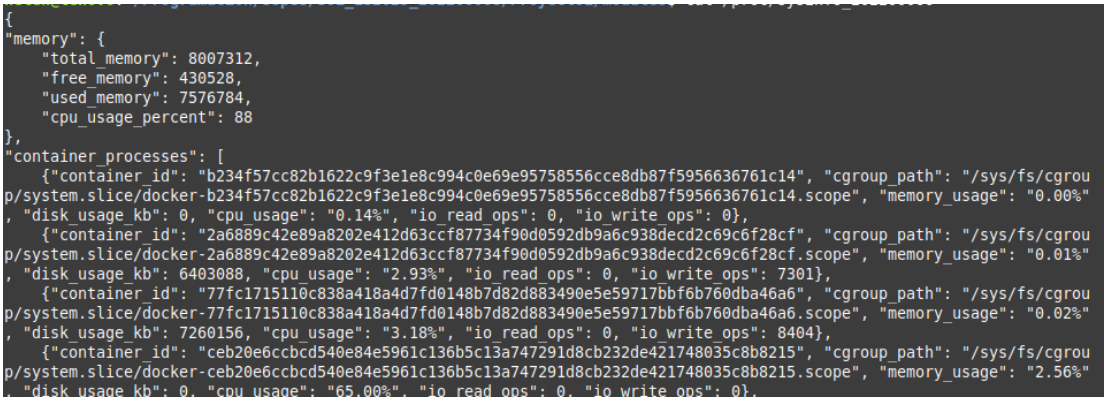
Luego se ejecuta el kernel:

```
helen@lenovo:~/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules$ make
make -C /lib/modules/6.8.0-45-generic/build M=/home/helen/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules mo
dules
make[1]: Entering directory '/usr/src/linux-headers-6.8.0-45-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-13 (Ubuntu 13.2.0-23ubuntu4) 13.2.0
  You are using:           gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
  CC [M]  /home/helen/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules/sysinfo.o
  MODPOST /home/helen/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules/Module.symvers
  CC [M]  /home/helen/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules/sysinfo.mod.o
  LD [M]  /home/helen/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules/sysinfo.ko
  BTF [M] /home/helen/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules/sysinfo.ko
Skipping BTF generation for /home/helen/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules/sysinfo.ko due to un
availability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.8.0-45-generic'
helen@lenovo:~/Programacion/sopes/SO1_1S2025_202200066/Proyecto1/modules$ sudo insmod sysinfo.ko
```

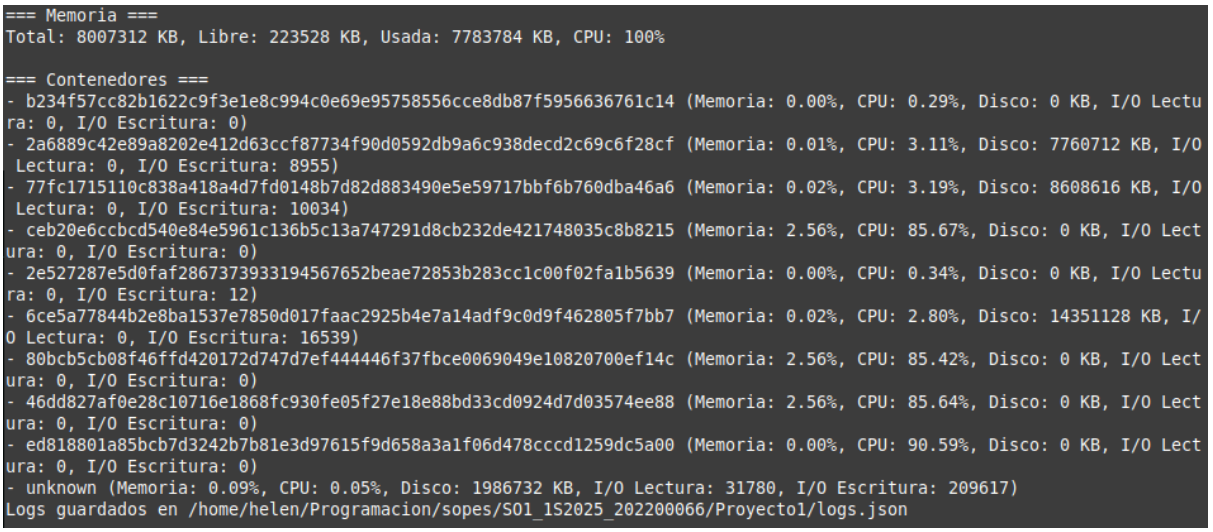
Se crean los otros 10 contenedores de forma random manualmente en este caso:

```
helen@lenovo:~/Programacion/sopes/SO1_1S2025_202200066/Proyecto1$ chmod +x create_containers.sh
helen@lenovo:~/Programacion/sopes/SO1_1S2025_202200066/Proyecto1$ sudo ./create_containers.sh
ed818801a85bcb7d3242b7b81e3d97615f9d658a3a1f06d478cccd1259dc5a00
Creado contenedor: cpu_1741928143_b26d6b6d (cpu)
46dd827af0e28c10716e1868fc930fe05f27e18e88bd33cd0924d7d03574ee88
Creado contenedor: ram_1741928143_4de29520 (ram)
80bcb5cb08f46ffd420172d747d7ef444446f37fbce0069049e10820700ef14c
Creado contenedor: ram_1741928144_93188150 (ram)
6ce5a77844b2e8ba1537e7850d017faac2925b4e7a14adf9c0d9f462805f7bb7
Creado contenedor: disk_1741928144_002c36c9 (disk)
2e527287e5d0faf2867373933194567652beae72853b283cc1c00f02fa1b5639
Creado contenedor: io_1741928145_92b839c1 (io)
ceb20e6ccbcd540e84e5961c136b5c13a747291d8cb232de421748035c8b8215
Creado contenedor: ram_1741928152_2c4679ee (ram)
77fc1715110c838a418a4d7fd0148b7d82d883490e5e59717bbf6b760dba46a6
Creado contenedor: disk_1741928168_11f87c3f (disk)
2a6889c42e89a8202e412d63ccf87734f90d0592db9a6c938decd2c69c6f28cf
Creado contenedor: disk_1741928200_2f6f1545 (disk)
b234f57cc82b1622c9f3e1e8c994c0e69e95758556cce8db87f5956636761c14
Creado contenedor: io_1741928221_31a895fe (io)
cf716277211eb6336284c5e5dd72c4b9c5003162030464ff57a318904cacf975
Creado contenedor: disk_1741928277_aba956a7 (disk)
```

Ya una vez cargados los contenedores con cat se puede ver la información de cada uno así como la información general del sistema:

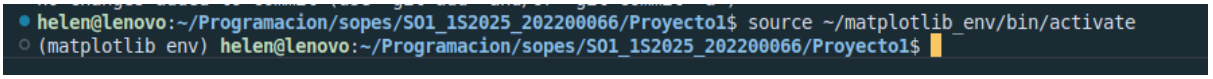


Ahora para hacerlo desde rust solo se ejecuta el comando cargo run y ya se puede ver los datos que cada 30 segundos se van recargando:



Para detenerlo se hace la combinacion CTRL + C y se detiene.

Para ver las graficas se debe estar dentro del entorno y activarlo:



Una vez activados se ejecuta el comando
python plot_containers.py

Y ya se pueden ver las gráficas que se muestran: