Universidad San Carlos de Guatemala
Facultad de ingeniería.
Ingeniería en ciencias y sistemas

Desarrollo de un módulo de kernel en C y un daemon en Go para el monitoreo de procesos y contenedores en Linux

Competencia que desarrollaremos

Al finalizar este proyecto, el estudiante será competente en:

- Diseñar, implementar, compilar y depurar módulos de kernel básicos en C para Linux.
- Comprender y utilizar estructuras de datos fundamentales del kernel relacionadas con la gestión de procesos.
- Crear y gestionar interfaces de comunicación entre el espacio de kernel y el espacio de usuario (a través de /proc) y consumir dicha interfaz desde una aplicación de usuario.
- Desarrollar aplicaciones en Go para interactuar con datos del sistema, incluyendo parseo de archivos.
- Integrar métricas con herramientas como Grafana para crear dashboards interactivos que faciliten el análisis del rendimiento del sistema.
- Aplicar buenas prácticas de programación tanto en el entorno del kernel como en el de aplicaciones de usuario.

Objetivos

Objetivo General

Desarrollar un sistema integrado que consta de un módulo de kernel en C para Linux que lista procesos activos de contenedores y expone su información vía /proc, y una aplicación en Go que interprete y presente estos datos de forma estructurada y amigable al usuario usando el sistema DAEMON en Linux.

Objetivos Específicos

Al finalizar el proyecto, los estudiantes deberán ser capaces de:

1. **Desarrollar e implementar un módulo de kernel funcional:** Crear un módulo en C que pueda ser cargado y descargado dinámicamente del kernel de Linux sin causar inestabilidad.

2. **Extraer información de procesos desde el kernel:** Acceder y recorrer las estructuras de datos internas del kernel para obtener PID, nombre del comando y memoria, cpu, entre otros datos.

3. **Crear una interfaz de kernel en /proc:** Implementar la lógica para crear un archivo virtual en /proc que, al ser leído, muestre la información recolectada de los procesos de contenedores.

4. **Desarrollar un Daemon con GO:** Implementar una herramienta que lea el archivo /proc generado por el módulo de kernel y a su vez parsee su contenido.

5. **Estabilizar el sistema por medio de cálculos realizados el Daemon de GO:** Desarrollar e implementar un sistema de monitoreo automatizado utilizando GO que, mediante el análisis de métricas de rendimiento (CPU, memoria, E/S), tome decisiones autónomas para estabilizar el entorno. El sistema evaluará en tiempo real el consumo de recursos de los contenedores y ejecutará acciones correctivas (detención y eliminación selectiva) cuando se superen umbrales predefinidos, garantizando la estabilidad del sistema sin intervención manual.

Enunciado del Proyecto

Este proyecto tiene como objetivo principal diseñar e implementar un sistema integral para la monitorización proactiva, análisis automatizado y gestión inteligente de contenedores en entornos Linux.

Descripción del problema a resolver

En la administración de sistemas y el desarrollo de aplicaciones contenerizadas, obtener información detallada sobre los procesos en ejecución y tomar acciones automatizadas es un desafío crítico. Aunque herramientas como ps o docker stats ofrecen datos básicos, carecen de acceso directo a las estructuras del kernel y no permiten una gestión proactiva de contenedores. Este proyecto propone una solución integral: desarrollar un módulo de kernel en C que exponga métricas avanzadas de procesos y contenedores (CPU, memoria, E/S) a través de /proc, junto con un Daemon en GO que no solo presente estos datos de forma legible, sino que también automatice decisiones (como terminar contenedores que excedan umbrales de recursos) a su vez que guarda datos importantes en SQLITE para su posterior uso en Grafana. Para validar el sistema, se implementará cronjobs que generen contenedores de prueba cada minuto, simulando condiciones de carga y permitiendo evaluar la eficacia de las acciones correctivas. Así, el proyecto combina aprendizaje en programación a bajo nivel (kernel) y alto nivel (GO), mientras resuelve un problema real en entornos contenerizados: la monitorización y estabilización autónoma del sistema. A su vez que se dichos datos se mostraran de manera visual por medio de un dashboard en Grafana.

1. **Un módulo de kernel desarrollado en C** que actuará como sensor de bajo nivel, accediendo directamente a las estructuras internas del kernel para capturar métricas detalladas de los procesos asociados a contenedores, incluyendo consumo de recursos (CPU, memoria, E/S, etc)

2. **Un segundo módulo de kernel desarrollado en C** que actuará como sensor de bajo nivel, accediendo directamente a las estructuras internas del kernel para capturar métricas detalladas de los procesos generales del sistema, incluyendo consumo de recursos (CPU, memoria, E/S, etc)

3. **Una Daemon en GO** que funcionará como cerebro del sistema, procesando los datos del kernel en tiempo real para:

    - Tomar decisiones autónomas detener y eliminar contenedores basadas en umbrales dinámicos y patrones de comportamiento establecidos previamente.
    - Simular y validar la eficacia del sistema bajo condiciones de uso prolongado.
    - Ejecutar scripts de automatización durante la ejecución.
4. **Un Cronjob** encargado de ejecutar el script que generara los contenedores de Docker cada minuto.
5. **Un Dashboard en Grafana** utilizado para mostrar la información que recolecta el servicio de GO.

MÓDULO DE KERNEL

Con los conocimientos adquiridos en la creación de módulos del kernel, deberá crear un módulo que capture las métricas necesarias para el análisis de la memoria y los contenedores activos en el sistema asi como un segundo modulo para listar los procesos del sistema.

La información debe ser capturada y guardada en la carpeta `/proc`:

1. Capturar en MB o KB (a discreción del estudiante):

- Total de memoria RAM
- Memoria RAM libre
- Memoria RAM en uso

2. Todos los procesos relacionados a los contenedores generados por el script así como los procesos generales del sistema deberán contar con:

- PID
- Nombre
- Línea de comando que se ejecutó o ID del contenedor
- VSZ (Tamaño de la memoria virtual en KB)
- RSS (Tamaño de memoria física en KB)
- Porcentaje de Memoria utilizada
- Porcentaje de CPU utilizado
- Estado (uso unicamente solo para los procesos generales del sistema operativo)

Sugerencias:

- Utilizar la estructura task_struct (del kernel de Linux) para filtrar correctamente únicamente los procesos relacionados con los contenedores y extraer la información necesaria.

3. Los datos deberán ser guardados en sus respectivos archivos en /proc

- Módulo de Procesos de Contenedores: `/proc/continfo_so1_#CARNET`

- Módulo de Procesos del Sistema: `/proc/sysinfo_so1_#CARNET`

CRONJOB

Aplicando los conocimientos adquiridos sobre la creación de imágenes y contenedores de Docker mediante el uso de la CLI y la creación de scripts de Bash, se deberá implementar un cronjob que se ejecute cada **1 minuto** con la siguiente funcionalidad:

De manera aleatoria, deberá generar 10 contenedores de las imágenes explicadas en la siguiente sección.

El estudiante deberá generar **3 imágenes de Docker**, divididas en dos grupos:

1. **Imagenes de Contenedores de alto consumo:**
    - Consumo de RAM
    - Consumo de CPU
2. **Imagen de Contenedor de bajo consumo de RAM y CPU.**

Las tecnologías o métodos para la creación de estas imágenes quedan a discreción del estudiante. Estas mismas deben ser documentadas para su validación.

GRAFANA

Como estudiante deberá realizar un dashboard en Grafana que muestre la siguiente información.

1. Total de RAM
2. Memoria Libre
3. Número de Contenedores Eliminados a lo largo del tiempo (Gráfico de líneas Timestamp)
4. Gráfica de Uso de RAM a lo largo del tiempo.
5. Top 5 contenedores con más consumo de RAM (mostrando el PID del proceso y ID) (Gráfico de Pie) , NO IMPORTA SI YA FUERON ELIMINADOS
6. Top 5 contenedores con más consumo de CPU (mostrando el PID del proceso y ID) (Gráfico de Pie) ,NO IMPORTA SI YA FUERON ELIMINADOS
7. RAM Usada
8. *Grafico extra propuesto por el Estudiante*

Dashboard a Realizar CONTENEDORES:

Dashboard a Realizar PROCESOS DEL SISTEMA:

Daemon de GO

Descripción (corazón del proyecto):
Se requiere desarrollar un **gestor de contenedores en Go** encargado del análisis, ejecución y comunicación entre los diferentes componentes del servicio.

Debe garantizar **manejo seguro de memoria** y cumplir con las siguientes funcionalidades:

1. Inicio del servicio

- Crear un contenedor de **Grafana** al inicializar el código.
- Grafana será el encargado de leer los logs generados por el servicio de Go después del análisis de los datos.
- Se recomienda utilizar un Docker Compose para que el contenedor se pueda comunicar con la base de datos SQLITE

2. Cronjob

- El daemon de Go iniciará la implementación y ejecución del cronjob en el sistema operativo, lo que activará el proceso de creación de contenedores.

3. Ejecución del script para cargar el módulo de kernel

- El daemon de Go ejecutará un script encargado de cargar e inicializar el módulo de kernel.

4. Loop principal (ejecución cada 20 segundos)

- El daemon operará de manera infinita en segundo plano.
- En cada iteración (cada 20 segundos), realiza lo siguiente:
    - Lectura del archivo en `/proc/continfo_so1_#CARNET`
    - Lectura del archivo en `/proc/sysinfo_so1_#CARNET`
    - Deserialización del contenido
    - Análisis para la gestión de contenedores (alta/baja, ajuste de recursos)
        - detener y eliminar los contenedores según cálculos explicados en la siguiente sección.
    - Generación y almacenamiento de registros (logs) en una base de datos SQLITE, diseñado para su posterior visualización en Grafana

5. Finalización del servicio

- Antes de finalizar, el servicio deberá **eliminar el cronjob** asociado a la creación de los contenedores.

RESTRICCIONES CLAVE

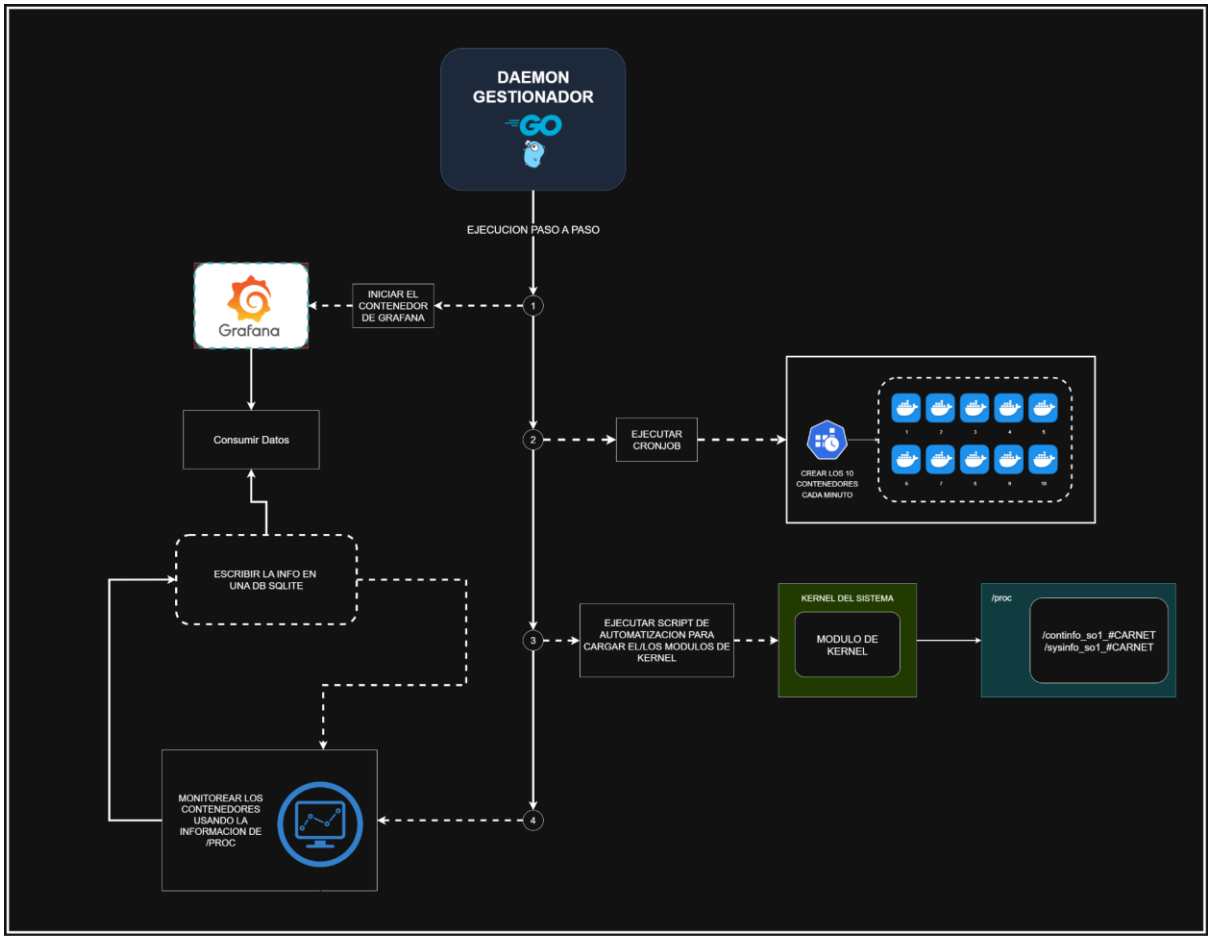
Contenedores en la máquina

- Siempre deben existir **3 contenedores de bajo consumo** en todo momento.
- Siempre deben existir **2 contenedores de alto consumo** en todo momento.
- **No debe eliminarse el contenedor de Grafana.**

Toma de decisiones

El servicio debe decidir **qué contenedores eliminar y cuáles mantener**, considerando las instrucciones dadas anteriormente.

ARQUITECTURA



Alcance del proyecto

- **Alcance obligatorio (Módulo de Kernel):**
  - Compilación, carga y descarga sin errores.
  - Creación de entrada en ***/proc/continfo_so1_#CARNET***
  - Creación de entrada en ***/proc/sysinfo_so1_#CARNET***
  - Manejar una estructura JSON para la información.
- **Alcance obligatorio (Daemon en GO):**
  - Debe compilar y ejecutarse sin errores.
  - Debe leer y parsear la información del archivo /proc creado por el módulo

Entregables

Tipo	Descripción
Link de repositorio con el código desarrollado del proyecto	Repositorio con todo el código y documentación utilizado para el desarrollo del proyecto (link de Github) en una carpeta llamada '*proyecto-1 separando por carpetas los códigos de la siguiente manera.'*.
/modulo-kernel: Módulos de Kernel	Módulos desarrollados en C, junto con el makefile y el archivo de extensión .ko
/go-daemon: Daemon de Go	Codigo desarrollado en Go para la implementación del Daemon y sus funcionalidades
/bash Archivos de Bash	Los archivo(s) utilizados para crear los contenedores por medio del cronjob, así como otros archivos bash que haya utilizado para el proyecto.
/dashboard Grafana	Docker compose y archivos utilizados para crear el dashboard en Grafana.
/documentacion: Manual Técnico	**Estructura del Módulo** ● Organización de archivos y directorios de su modulo. ● Funciones principales y su propósito. ● Dependencias externas (si las hay). **Compilación del Módulo** ● Instrucciones paso a paso para compilar. **Carga y Descarga del Módulo** ● Comandos para cargar (insmod) y descargar (rmmod). ● Verificación de carga correcta (dmesg, lsmod). **Pruebas y Verificación**

	● Cómo probar el módulo (ej: leer `/continfo_so1_#CARNET`). ● Comandos para validar su funcionamiento (cat). **Decisiones de Diseño y Problemas** ● Explicación de decisiones clave. ● Problemas encontrados y soluciones aplicadas. **Estructura del Daemon GO** ● **Documentar y explicar cada función dentro de su código** **Documentar el/los archivos de automatización usados**
/documentacion: Manual de Usuario	**Manual de Usuario** ● Instrucciones claras para el usuario final. ● Ejemplos de uso práctico. **Guía de Instalación** ● Requisitos del sistema. ● Pasos detallados de instalación y configuración. **Diagramas y Arquitectura** ● Diagramas de flujo o estructura (texto/imágenes). ● Descripción de la arquitectura del sistema para usuarios no técnicos. **Cómo acceder al Dashboard de Grafana** ● Explicación para el usuario sobre cómo acceder al dashboard y ver las gráficas.