**Reporte Técnico**

Desarrollo de un Monitor de Sistema en Tiempo Real con Notificaciones de Umbrales de Recursos

**Integrantes**

Claudia Lucía Serna Gómez

Paula Andrea Guarin Guarin

Duvan Ferney Ruiz Ocampo

**Docente**

Henry Alberto Arcila Ramirez

Universidad de Antioquia

Facultad de Ingeniería

Curso Sistemas Operativos

2025-1

**Contenido**

[1. Lenguaje y estándares utilizados 3](#_Toc202819646)

[2. Compilación y dependencias 3](#_Toc202819647)

[3. Arquitectura del sistema 3](#_Toc202819648)

[3.1. Componentes principales 4](#_Toc202819649)

[3.2. Diagrama de flujo 4](#_Toc202819650)

[4. Funcionalidades clave 4](#_Toc202819651)

[5. Pruebas y Validación 5](#_Toc202819652)

[6. Configuración de umbrales 5](#_Toc202819653)

**Tablas**

[Tabla 1. Componentes principales 4](#_Toc202819523)

[Tabla 2. Escenarios de Prueba 6](#_Toc202819524)

**Ilustraciones**

[Ilustración 1. Diagrama de flujo 5](#_Toc202819571)

[Ilustración 2. Configuración de Umbrales 7](#_Toc202819572)

El Monitor de Sistema en Tiempo Real es una herramienta desarrollada en C para sistemas Linux que permite supervisar el consumo de recursos (CPU y memoria) de los procesos activos. Su objetivo principal es detectar procesos que excedan umbrales predefinidos y notificar al administrador mediante alertas visuales, logs y señales. El sistema se inspira en herramientas como **top** y **htop**, pero con un enfoque en la personalización de umbrales y mecanismos de notificación.

# **Lenguaje y estándares utilizados**

* **Lenguaje de programación:** C (Copilado con gcc)
* **Entorno:** WSL/Ubuntu + VS Code
* **Lectura de métricas:** /proc, sysfs
* **Interfaz de usuario:** ncurses

# **Compilación y dependencias**

Para copilar el proyecto, como se menciono anteriormente se requiere de tener instalado el sistema operativo WSL2 Ubuntu + Visual Studio Code.

**Librerías**

sudo apt-get install libncurses-dev # Interfaz ncurses

sudo apt-get install stress-ng valgri# Para pruebas

**Configurar Umbrales en config.ini**

[thresholds]

cpu\_alert = 1  # 5% de CPU

mem\_alert = 1  # 5% de memoria total

**Comando de compilación y ejecución**

gcc -Iinclude src/\*.c -o build/monitor -lncurses

make

./build/monitor

# **Arquitectura del sistema**

## **Componentes principales**

|  |  |
| --- | --- |
| **Modulo** | **Función** |
| main.c | Punto de entrada. Gestiona el bucle principal y la interacción entre módulos. |
| process\_monitor.c | Obtiene métricas de procesos (CPU, memoria) desde /proc. |
| alerts.c | Maneja la configuración y el triggering de alertas (logs, señales). |
| config.ini | Archivo de configuración para umbrales y rutas de logs. |

Tabla 1. Componentes principales

## **Diagrama de flujo**

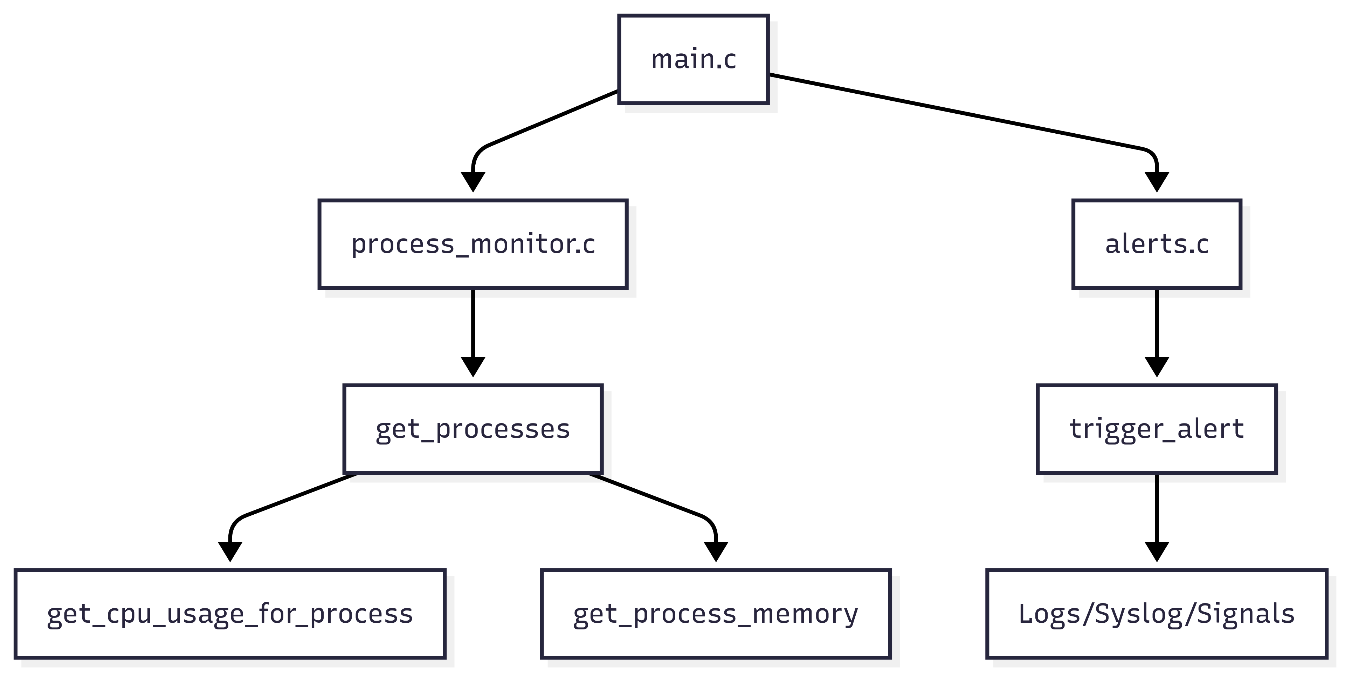


Ilustración 1. Diagrama de flujo

# **Funcionalidades clave**

**Monitoreo en Tiempo Real**

* **Lectura de /proc:**
  + Extrae información de procesos (/proc/[pid]/stat, /proc/[pid]/status).
  + Calcula el uso de CPU por proceso mediante diferencias de utime y stime.
  + Obtiene el consumo de memoria (VmRSS) en KB.
* **Métricas del Sistema:**
  + Uso total de CPU (lectura de /proc/stat).
  + Uso de memoria (lectura de /proc/meminfo).

**Sistema de Alertas**

* **Umbrales Configurables:**
  + Valores por defecto: 5% para CPU y memoria (para efectos de prueba)
  + Personalizables vía config.ini.

**Interfaz de Usuario (ncurses)**

* **Visualización:**
  + Lista de procesos ordenados por uso de CPU.
  + Códigos de color (rojo/amarillo) para procesos que exceden umbrales.
  + Barra superior con métricas globales del sistema.

# **Pruebas y Validación**

**Escenarios de Prueba**

|  |  |  |
| --- | --- | --- |
| **Herramientas** | **Prueba** | **Resultado Esperado** |
| Stress-ng | stress-ng --cpu 4 --timeout 60s | Alertas por CPU > 5% en los procesos de stress-ng (resaltados en rojo en la UI). |
| stress-ng | stress-ng --vm 1 --vm-bytes 1G | Alertas por memoria > 5% en UI. |
| valgrind | Valgrind ./build/monitor | Sin memory leaks o accesos inválidos a memoria. |

Tabla 2. Escenarios de Prueba

**Resultados Obtenidos**

* El monitor detectó correctamente procesos con > 5% de CPU y generó alertas.
* Los procesos que consumieron > 5% de la memoria total fueron resaltados en amarillo en la UI.
* No se detectaron memory leaks en las pruebas básicas.

# **Configuración de umbrales**



Ilustración 2. Configuración de Umbrales