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

CONTENIDO DE TABLAS

CONTENIDO DE ILUSTRACIONES

Resumen

En el contexto de la administración de sistemas operativos, la detección oportuna de procesos que consumen recursos excesivos es fundamental para garantizar la estabilidad y el rendimiento del sistema. Este proyecto propone el diseño e implementación de un monitor de sistema desarrollado desde cero, utilizando el lenguaje de programación C (similar a top o htop) y haciendo uso del sistema de archivos /proc de Linux para recolectar información de los procesos activos. El sistema permitirá establecer umbrales definidos por el administrador (como uso de CPU o memoria) y notificará automáticamente cuando un proceso exceda dichos límites. La herramienta busca brindar una solución liviana, configurable y eficaz para el monitoreo en tiempo real, contribuyendo a la gestión proactiva de los recursos del sistema. Esta propuesta se enmarca en el desarrollo de competencias clave en el área de sistemas operativos, abordando conceptos como la gestión de procesos, planificación de recursos y comunicación entre usuario y administrador.

1. Introducción

En los sistemas operativos modernos, uno de los desafíos constantes es garantizar un uso equilibrado y controlado de los recursos del sistema, como la CPU y la memoria. Procesos mal diseñados, con errores o de comportamiento anómalo pueden llegar a consumir de forma excesiva estos recursos, afectando el rendimiento general del sistema e incluso provocando fallos. Esta situación es especialmente crítica en entornos multiusuario o servidores, donde la estabilidad es prioritaria.

El presente proyecto aborda la necesidad de contar con una herramienta que permita monitorear en tiempo real el comportamiento de los procesos del sistema y notificar al administrador cuando alguno de ellos exceda los límites de uso previamente definidos. Esta funcionalidad resulta clave para prevenir cuellos de botella, mejorar la administración del sistema y reducir el tiempo de respuesta ante situaciones de sobrecarga.

En el contexto tecnológico actual, donde los sistemas deben mantenerse disponibles y eficientes de forma continua, el desarrollo de soluciones personalizadas para la supervisión de recursos cobra gran relevancia. Implementar un monitor desde cero no solo permite adaptar la herramienta a las necesidades específicas del entorno, sino que además fortalece la comprensión de conceptos fundamentales de los sistemas operativos, como la gestión de procesos, la planificación de recursos y la interacción con el sistema de archivos /proc en Linux. Este proyecto combina conceptos fundamentales de Sistemas Operativos (gestión de procesos, scheduling, memoria) con programación en C, siendo una herramienta didáctica y práctica.

2. Antecedentes y/o marco teórico

Los sistemas operativos tienen la tarea clave de administrar los recursos del sistema (como CPU, memoria y almacenamiento) de manera eficiente para garantizar un buen rendimiento. Sin embargo, cuando estos recursos se usan en exceso, el sistema puede volverse lento o incluso fallar. Por eso, es esencial contar con herramientas que vigilen constantemente el estado del sistema, detecten situaciones de riesgo (como un consumo demasiado alto de recursos) y envíen alertas cuando se superen ciertos límites.

2.1. Principales aspectos teóricos para el desarrollo del proyecto:

- **Sistema de archivos /proc**
    - Archivos virtuales que exponen métricas del kernel (ej: /proc/stat para CPU, /proc/meminfo para memoria).
- **Gestión de procesos en Linux**
    - Estructuras de datos del kernel (task_struct), estados de procesos (running, sleeping, zombie).
- **Scheduling y prioridades**
    - Uso de nice y priority en procesos
- **Comunicación entre procesos (IPC)**
    - Señales (SIGALRM), pipes, o sockets para notificaciones.

2.2. Relación con el curso de Sistemas Operativos:

- **Teórica:**
    - Procesos, scheduling, gestión de memoria, sistemas de archivos, uso de comando UNIX/Linux, programación concurrente, procesos y subprocesos.
- **Laboratorio:**
    - Uso de llamadas al sistema (open, read), concurrencia, y manejo de señales, sistema de archivos virtuales.

3. Objetivos

3.1. Objetivo principal

Desarrollar un monitor de sistema que supervise en tiempo real el consumo de recursos (CPU y memoria) de los procesos activos, notificando al administrador cuando se superen umbrales predefinidos, para facilitar la gestión eficiente del sistema.

3.2. Objetivos específicos

- Implementar un mecanismo de monitoreo en tiempo real que evalúe el consumo de CPU y memoria de cada proceso.
- Desarrollar un sistema de notificación que alerte al administrador cuando un proceso exceda los umbrales establecidos.
- Validar el funcionamiento del monitor mediante pruebas controladas en diferentes escenarios de carga del sistema.

4. Metodología

El proyecto seguirá una metodología incremental y basada en prototipos, dividida en etapas claras que permitirán construir el monitor de sistema paso a paso, validando cada componente antes de avanzar al siguiente. Se emplearán prácticas de programación modular para facilitar el mantenimiento y pruebas unitarias.

4.1. Herramientas y Tecnologías

Componente	Herramientas/Librerías	Uso
Lenguaje	C (Copilado con gcc)	Lenguaje que será utilizado para el desarrollo del laboratorio
Entorno	WSL/Ubuntu + VS Code	Máquina virtual para la ejecución del programa
Lectura de métricas	/proc, sysfs	Acceso a datos del kernel (CPU, memoria, procesos).
Interfaz de usuario	ncurses	Mostrar datos en tiempo real con una interfaz interactiva tipo htop.
Notificaciones	syslog, signal.h, sockets	Alertas mediante logs locales, señales, o comunicación con el administrador.
Depuración	gdb, Valgrind	Detección de errores y memory leaks.
Gestión de código	Git + GitHub	Control de versiones y colaboración.

Tabla 1. Herramientas y tecnologías usadas

4.2. Actividades

4.2.1. Fase 1: Investigación y Diseño

- **Análisis de** /proc:
    - Estudio de archivos

      clave: /proc/stat (CPU), /proc/meminfo (RAM), /proc/[pid]/stat (procesos).
    - Definición de estructuras de datos en C para almacenar métricas.
- **Diseño de la arquitectura:**
    - Diagrama de módulos (lectura de datos, interfaz, notificaciones).

4.2.2. Fase 2: Implementación Básica

- **Lectura de métricas:**
    - Código para extraer %CPU, memoria libre, lista de procesos.
- **Pruebas unitarias:**
    - Verificar que los datos se lean correctamente con printf.

4.2.3. Fase 3: Interfaz de Usuario

- **Integración de ncurses:**
    - Creación de ventanas para mostrar métricas en tiempo real.
    - Uso de colores para resaltar procesos críticos.

- **Actualización periódica:**
    - Loop principal con sleep() o nanosleep() para refrescar la pantalla cada 1-2 segundos.

4.2.4. Fase 4: Mecanismos de Notificación

- **Umbrales configurables:**
    - Archivo de configuración (ej: config.ini) para definir límites de CPU/RAM.
- **Alertas:**
    - **Opción 1:** Logs en /var/log/monitor.log con syslog.
    - **Opción 2:** Señal SIGALRM para notificaciones en tiempo real.

4.2.5. Fase 5: Pruebas Integrales

- **Escenarios de prueba**:
    - Proceso con alto consumo de CPU: stress-ng --cpu 4 --timeout 60s.
    - Proceso con fuga de memoria: valgrind --leak-check=yes ./monitor.
- **Validación de alertas:**
    - Verificar que los logs o señales se activen al superar umbrales.

4.2.6. Fase 6: Optimización y Documentación

- **Optimización:**
    - Reducir el uso de CPU del monitor (evitar loops infinitos sin sleep).
- **Documentación:**
    - Manual de usuario: Cómo compilar, configurar umbrales y leer alertas.
    - Comentarios en código para futuras mejoras.

5. Cronograma

Desarrollo de un Monitor de Sistema en Tiempo Real con Notificaciones de Umbrales de Recursos																																					
			Marzo							Abril							Mayo							Junio							Julio						
Actividad	Fecha inicio	Fecha fin	1	5	10	15	20	25	30	1	5	10	15	20	25	30	1	5	10	15	20	25	30	1	5	10	15	20	25	30	1	5	10	15	20	25	30
Investigación de /proc y diseño inicial	1-mar	30-mar																																			
Implementacion de lectura de métricas (CPU, memoria)	1-abr	30-abr																																			
Desarrollo de la interfaz con ncurses	1-may	30-may																																			
Integración de notificaciones y umbrales	1-jun	30-jun																																			
Pruebas y optimizacion	15-jun	5-jul																																			
Documentacion y entrega	1-jul	10-jul																																			

Ilustración 1. Cronograma de actividades

6. Bibliografía

The kernel development community (s.f). The /proc Filesystem. Kernel. https://kernel.org/doc/html/latest/filesystems/proc.html

Love, R. D. (2008). Linux system programming. *Choice Reviews Online*, *45*(08), 45-4429. https://doi.org/10.5860/choice.45-4429

Bovet, D. P., Cassetti, M., & Oram, A. (2000). *Understanding the Linux Kernel*. http://ermak.cs.nstu.ru/Understanding.Linux.Kernel.pdf