Reporte técnico: Proyecto final de Sistemas Operativos y Laboratorio

1. Información del Proyecto

- **Título del Proyecto:** Implementación de un monitor de sistema desde cero
- **Curso/Materia:** Sistemas Operativos
- **Integrantes:** Jonathan David Fernandez Vargas jonathand.fernandez@udea.edu.co, Valeria Alvarez Fernandez valeria.alvarezf@udea.edu.co, Santiago Arenas Gomez santiago.arenas1@udea.edu.co
- **Fecha de Entrega:** 10/07/2025

2. Introducción

2.1. Objetivo del Proyecto

El proyecto tiene como objetivo diseñar e implementar desde cero un monitor de sistema con interfaz gráfica, que muestre información clave del estado del sistema operativo en tiempo real, como uso de CPU, memoria RAM, almacenamiento, red y procesos activos. Se implementó en lenguaje C con GTK+3 para lograr una solución multiplataforma (Linux y Windows), reforzando el aprendizaje práctico de los conceptos fundamentales de sistemas operativos.

2.2. Motivación y Justificación

Elegimos este proyecto porque permite aplicar de forma práctica temas clave del curso, como la gestión de procesos, la administración de memoria y el uso de llamadas al sistema. En la actualidad, comprender el rendimiento de un sistema es esencial tanto para administradores como para desarrolladores. Crear un monitor desde cero nos permitió estudiar en profundidad cómo los sistemas operativos exponen información a nivel de kernel y cómo herramientas tradicionales como `top` o `htop` obtienen y presentan estos datos.

2.3. Alcance del Proyecto

Incluye:

- Visualización gráfica del uso de CPU, RAM, disco y red.
- Lista de procesos activos (PID y nombre) en una ventana GTK.
- Actualización automática de los datos cada 2 segundos.
- Código modular y funcional en Linux y Windows (MSYS2 MinGW 64-bit).
- Empaquetado y ejecución desde consola usando `make`.

Excluye:

- Funcionalidades de gestión de procesos (matar o suspender).
- Exportación de estadísticas.
- Soporte para otras plataformas fuera de Linux y Windows.

3. Marco Teórico / Conceptos Fundamentales

Gestión de Procesos: Comprende la planificación y ejecución de procesos por parte del sistema operativo. Información como PID, estado y consumo de recursos se obtiene de `/proc/[PID]/status`.

Planificación del CPU: Permite entender cómo se distribuye el tiempo de CPU entre procesos, lo cual se refleja en `/proc/stat`.

Administración de Memoria: Se estudia a través del archivo `/proc/meminfo`, que contiene estadísticas detalladas sobre RAM y swap.

/proc y acceso al kernel: `/proc` es un sistema de archivos virtual en Linux que expone estadísticas del kernel en tiempo real. Aquí se accede a archivos como `/proc/stat`, `/proc/meminfo` y `/proc/[PID]/status` para obtener información de recursos y procesos.

Llamadas al sistema y portabilidad: Se utilizan llamadas como `statvfs()` para almacenamiento y `sysconf()` para obtener configuración del sistema. En Windows, se implementaron funciones análogas usando la API del sistema.

GTK+3: Biblioteca gráfica multiplataforma para la construcción de interfaces gráficas de usuario en C. Permite el diseño de ventanas, etiquetas, listas y actualizaciones dinámicas.

Multiplataforma (Linux/Windows): Se emplearon condiciones de precompilación (`#ifdef _WIN32`) para adaptar el código a cada sistema operativo.

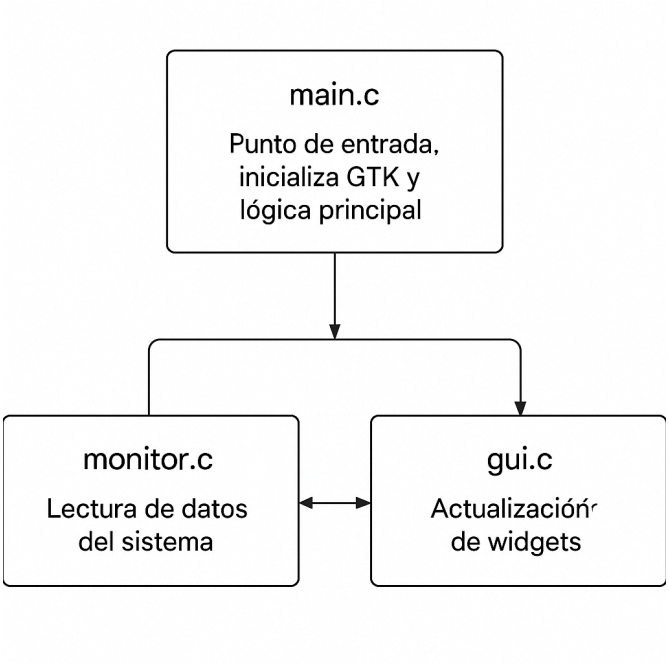
4. Diseño e Implementación

4.1. Diseño de la Solución

Se adoptó un enfoque modular para separar la lógica de recolección de datos del sistema y la presentación gráfica. La estructura general del proyecto es:

```
monitor-system-os/
├── Makefile
└── src/
    ├── main.c        // Punto de entrada, inicializa GTK y lógica principal
    ├── monitor.c     // Lógica de recolección de datos del sistema
    ├── monitor.h     // Declaraciones de funciones y estructuras
    └── gui.c         // Construcción y actualización de la interfaz GTK
```

Diagrama de Bloques:



4.2. Tecnologías y Herramientas

1. **Lenguaje de programación:** C
2. **Framework de GUI:** GTK+3
3. **Sistema Operativo:** Linux (Ubuntu 22.04), Windows (MSYS2 MinGW 64-bit)
4. **Editor de Código:** Visual Studio Code
5. **Compilación:** `make`, `pkg-config`
6. **Control de versiones:** Git

4.3. Detalles de Implementación

CPU y memoria: En Linux, la información se obtiene desde `/proc/stat` y `/proc/meminfo`. En Windows, se utiliza la API `GlobalMemoryStatusEx` y `GetSystemTimes`.

Procesos: Se recorren los directorios de `/proc` para leer el nombre y estado desde `/proc/[PID]/status`. En Windows, se usa `CreateToolhelp32Snapshot`.

Disco: Se consulta con `statvfs()` en Linux y `GetDiskFreeSpaceEx()` en Windows.

GUI: Se construyó con GTK usando `GtkGrid`, `GtkLabel`, `GtkListBox`, y temporizadores (`g_timeout_add`) para actualizar los datos cada 3 segundos.

5. Pruebas y Evaluación

5.1. Metodología de Pruebas

- Pruebas funcionales: Validar que los datos mostrados son correctos frente a herramientas estándar (`htop`, `free`, `df`, etc.).
- Pruebas de compatibilidad: Validado en Linux (Ubuntu) y Windows 10 (MSYS2).
- Pruebas visuales: Verificar la actualización fluida y correcta de los widgets en GTK.

5.2. Casos de Prueba y Resultados

Se presentan los casos de prueba más importantes que se ejecutaron y los resultados obtenidos:

ID Caso de prueba	Descripción del caso de prueba	Resultado esperado	Resultado obtenido	Éxito/Fallo
CP-001	Visualización de CPU	Se actualiza cada 3 segundos	Se actualiza correctamente	✅
CP-002	Lectura de RAM	Coincide con `free -h`	Mismo valor observado	✅
CP-003	Lista de procesos (PID y nombre)	Aparecen procesos actuales	Lista desplegable visible	✅
CP-004	Uso de disco	Coincide con `df -h`	Se actualiza correctamente	✅
CP-005	Ejecución en Windows	Se muestra interfaz sin errores	Funciona desde MSYS2	✅
CP-006	Rendimiento bajo carga	Interfaz se mantiene fluida	Confirmado sin retardos	✅

5.3. Evaluación del Rendimiento

El monitor mantiene un bajo uso de CPU (<2%) y RAM (<15MB). La eficiencia se debe al uso de estructuras ligeras y llamadas al sistema específicas sin sobrecarga de procesos secundarios.

5.4. Problemas Encontrados y Soluciones

GTK en Windows: Inicialmente hubo errores con dependencias. Se solucionó documentando bien la instalación vía `pacman` en MSYS2.

Incompatibilidad de rutas entre sistemas: Se resolvió usando macros de preprocesador y funciones condicionales (`#ifdef`).

Flickering de interfaz: Solucionado utilizando actualizaciones parciales de los widgets en lugar de reconstrucción total.

6. Conclusiones

El desarrollo del monitor de sistema con interfaz gráfica y soporte multiplataforma representó una experiencia integral de aplicación de los conceptos fundamentales del curso de Sistemas Operativos. A través de esta implementación, fue posible profundizar en la comprensión de la arquitectura de los sistemas modernos, especialmente en lo relacionado con la gestión de procesos, administración de memoria, monitoreo del uso del CPU y del sistema de archivos.

Trabajar con el sistema de archivos virtual `/proc` en Linux y con las API específicas de Windows permitió consolidar el entendimiento del acceso a información del kernel y el uso de llamadas al sistema. Además, el uso del lenguaje C —en combinación con la biblioteca gráfica GTK+— fortaleció las habilidades de programación a bajo nivel, manejo de punteros, modularidad del código y adaptación a entornos distintos.

El proyecto no solo cumplió con los objetivos funcionales planteados, como la visualización de métricas clave en tiempo real y la implementación de una interfaz gráfica clara y eficiente, sino que también superó expectativas al lograr una solución portable entre sistemas operativos, con una estructura de código mantenible y preparada para futuras mejoras.

En términos de aprendizaje, el proceso permitió a los integrantes del equipo adquirir una visión práctica del funcionamiento interno de un sistema operativo, comprender cómo se exponen y manipulan los recursos del sistema, y desarrollar competencias en diseño de software interactivo. Asimismo, se enfrentaron y resolvieron retos técnicos reales, lo cual consolidó habilidades tanto técnicas como colaborativas.

Finalmente, el proyecto demuestra que es posible construir herramientas útiles y eficientes sin depender de librerías complejas ni entornos pesados, lo que resalta la importancia del conocimiento profundo del sistema operativo y del enfoque modular y portable en el desarrollo de software de sistemas.

7. Trabajo Futuro

- Incorporar funciones para filtrar y ordenar procesos.
- Añadir gráficas de barras o líneas en tiempo real.
- Permitir exportar estadísticas a archivos CSV.
- Añadir control de procesos (terminar, pausar).
- Mejorar internacionalización y diseño responsivo.

8. Referencias

Silberschatz, A., Galvin, P. B., & Gagne, G. (2020). *Operating System Concepts* (10th ed.). Wiley.

Love, R. (2010). *Linux System Programming*. O'Reilly Media.

Documentación oficial de GTK+3: https://docs.gtk.org

Linux Man Pages: `/proc`, `statvfs`, `meminfo`, `top`, `ps`.

Microsoft Developer Docs – Windows API para gestión de procesos y recursos: https://learn.microsoft.com

README del proyecto: Monitor del Sistema con Interfaz Gráfica (2025).

9. Anexos

Repositorio de GitHub: https://github.com/Jonathand77/monitor-system-os

- Código fuente completo en carpeta `src/`.
- Capturas de video de la ejecución: https://drive.google.com/file/d/17JJB5qaLIvSBVxiiNDQr3zAVfkt36Vbe/view?usp=sharing
- Diagramas de arquitectura y flujo.
- Tabla de casos prueba y resultados.
- Archivo `Makefile` y pasos de compilación.
- Instrucciones detalladas para instalar dependencias y ejecutar el monitor dentro del repositorio.