Universidad Nacional Autónoma De México

Facultad De Ciencias

Sistemas Operativos | Semestre 2026-1

Profesor: M. en C. Javier León Cotonieto

Proyecto Final:

Desarrollo de un monito básico del sistema operativo

Equipo 5

Integrantes:

Jiménez Sánchez Emma Alicia

Salazar Gonzalez Pedro Yamil

Sánchez Cruz Norma Selene

Suárez Ortiz Joshua Daniel

Fecha de entrega:

1 de Diciembre del 2025

Introducción

El monitoreo del funcionamiento interno de un sistema operativo constituye una herramienta fundamental para comprender su comportamiento, evaluar su desempeño y detectar problemas y posibles anomalías en tiempo real. En el contexto de la administración de sistemas y la ingeniería de software, los monitores del sistema permiten observar de manera estructurada numerosos recursos críticos, tales como el uso del procesador (CPU), la ocupación de memoria principal, el estado de los procesos activos, el acceso a dispositivos de almacenamiento y la actividad de la red. Estas métricas proporcionan una visión integral del estado del sistema y facilitan la toma de decisiones informadas respecto a la optimización, depuración y gestión eficiente de recursos.

El creciente nivel de complejidad de los sistemas modernos, sumado a la necesidad de ejecutar aplicaciones cada vez más exigentes, convierte al monitoreo en un componente muy importante tanto para usuarios avanzados como para desarrolladores y administradores. Un monitor básico del sistema no solo permite visualizar el comportamiento dinámico de los recursos, sino que también sirve como punto de partida para comprender cómo el sistema operativo distribuye el trabajo, gestiona el hardware y mantiene la estabilidad global del sistema operativo.

En este proyecto se implementa una herramienta sencilla capaz de mostrar en tiempo real información relevante del sistema operativo. Dicha herramienta incorpora métricas dinámicas y actualizables, y se apoya en librerías específicas que permiten la interacción directa con los recursos del sistema. A través de su desarrollo, se busca reforzar los conocimientos adquiridos en clase, así como mostrar los mecanismos mediante los cuáles el software accede, supervisa y reporta el estado interno de un sistema en funcionamiento.

Objetivos

Desarrollar e implementar un monitor básico del sistema que sea capaz de mostrar en tiempo real información relevante sobre el uso de recursos del sistema operativo con el fin de comprender, analizar y evaluar el comportamiento interno del sistema desde una perspectiva práctica y aplicada

Objetivos específicos

1. Diseñar e implementar una herramienta funcional que muestre al menos dos métricas del sistema operativo, garantizando su actualización en tiempo real.
2. Integrar librerías, APIs o interfaces del sistema operativo que permitan acceder a la información sobre los recursos del sistema.
3. Desarrollar una interfaz de usuario que permita presentar la información de manera clara, ordenada y accesible para el usuario.
4. Analizar el funcionamiento del monitor desarrollado, evaluando su capacidad para capturar métricas de forma correcta, consistente y eficiente bajo diferentes condiciones del sistema.

5. Documentar el proceso de diseño e implementación, describiendo las técnicas, herramientas, librerías y consideraciones tomadas para garantizar la correcta interacción con el sistema operativo.
6. Fortalecer los conocimientos teóricos adquiridos en la asignatura, relacionando los conceptos de administración de recursos, planificación de procesos y manejo de memoria con una herramienta práctica que muestre su funcionamiento.

Marco teórico

Sistemas Operativos: Conceptos Fundamentales

Un sistema operativo (SO) es el conjunto de programas que gestionan los recursos hardware y software de un equipo de cómputo, actuando como intermediario entre el usuario y el hardware. Su finalidad principal consiste en proporcionar un entorno estable y eficiente en el cual puedan ejecutarse aplicaciones, administrar la comunicación con los dispositivos físicos y garantizar la correcta distribución de los recursos críticos del sistema. Entre sus funciones esenciales se encuentran la administración del procesador, la gestión de la memoria, el control de dispositivos de entrada y salida, la gestión del almacenamiento, la protección del sistema y la provisión de interfaces para los usuarios y desarrolladores.

Desde una perspectiva general, los sistemas operativos modernos integran múltiples componentes interrelacionados que permiten el funcionamiento simultáneo de numerosos programas, garantizando mecanismos de aislamiento, seguridad y eficiencia. Estas capacidades permiten mantener un sistema multitarea, capaz de ejecutar varios procesos en paralelo mediante técnicas de planificación y multiplexación del hardware.

Procesos y Gestión del Procesador

Un proceso es una instancia en ejecución de un programa. Cada proceso cuenta con un estado propio, un conjunto de recursos asignados y un contexto que incluye registros, memoria, archivos abiertos y otros elementos esenciales para la ejecución. El sistema operativo se encarga de crear, planificar, suspender y finalizar procesos de acuerdo con las necesidades del usuario y de las aplicaciones del sistema.

La gestión del procesador o CPU scheduling constituye una de las tareas más relevantes del SO. El scheduler determina qué proceso se ejecuta en un momento dado y cómo se distribuye el tiempo de CPU entre las diferentes tareas. Entre los algoritmos de planificación más conocidos se encuentran Round Robin, Prioridades, Shortest Job First (SJF) y Multilevel Feedback Queue.

Para el monitoreo del sistema, el análisis de del uso del CPU resulta fundamental, ya que permite identificar cargas excesivas, procesos que consumen muchos recursos o problemas potenciales como battlenecks en el rendimiento global del sistema.

Gestión de la Memoria

La memoria principal es un recurso crítico y limitado del sistema. Su correcta administración garantiza que los procesos dispongan del espacio necesario para ejecutarse, evitando colisiones, pérdida de datos o degradaciones importantes en el rendimiento.

Los sistemas operativos implementan técnicas como segmentación, paginación y memoria virtual para gestionar el uso eficiente de este recurso. La memoria virtual, por ejemplo, permite que los procesos utilicen más memoria de la físicamente disponible mediante el uso de espacios en disco como extensión del espacio de direcciones.

El monitoreo de la memoria suele incluir métricas como:

- Memoria total
- Memoria usada
- Memoria libre
- Memoria intercambiada (swap)
- Caché y buffers del sistema

Estas métricas permiten evaluar la estabilidad del sistema y detectar situaciones de sobrecarga o fuga de memoria.

Sistemas de Archivos y Uso del Disco

Un sistema de archivos es la estructura lógica que organiza, almacena y recupera información en dispositivos de almacenamiento como discos duros, SSD o unidades externas. Los sistemas operativos utilizan sistemas de archivos como NTFS, ext4, APFS, entre otros.

El monitoreo del disco permite conocer:

- Uso del espacio total
- Capacidad disponible
- Lecturas y escrituras por segundo
- Actividad del disco en tiempo real

Estas métricas son esenciales para diagnosticar problemas de rendimiento relacionados con el acceso al almacenamiento o proyecciones de crecimiento del uso del espacio.

Red y Comunicaciones

La interfaz de red constituye otro de los recursos fundamentales del sistema. Los sistemas operativos adminstran la comunicación entre protocolo, adaptador de red y aplicaciones mediante pilas como TCP/IP.

El monitoreo de red generalmente incluye:

- Cantidad de datos enviados y recibidos
- Velocidad de transmisión
- Paquetes por segundo
- Conexiones activas
- Errores o colisiones

Este tipo de métricas permite analizar el rendimiento de las comunicaciones, diagnosticar congestiones y detectar anomalías en tráfico inusual.

Interfaces y Herramientas del Monitoreo del SO

Los sistemas operativos ponen a disposición de los usuarios diversas herramientas que permiten supervisar su funcionamiento. Entre las más conocidas se encuentran:

- Linux/UNIX
    - top
    - htop
    - vmstat
    - iostat
    - /proc
    - filesystem
- Windows
    - Task Manager
    - Resource Monitor
    - Performance Counters
- macOS
    - Activity Monitor

El acceso a estas métricas se realiza mediante llamadas al sistema, APIs específicas o interfaces como el directorio /proc en Linux, que expone información sobre procesos, memoria, CPU, interrupciones, entre otros.

Librerías para el Acceso a Información del Sistema

En el desarrollo de herramientas personalizadas de monitoreo, es común emplear librerías que encapsulan el acceso a APIs del sistema operativo. Una de las más utilizadas es psutil (Python System and Process Utilities), la cual proporciona funciones de alto nivel para consultar la utilización del CPU, Información de Procesos, Memoria virtual y swap, actividad de red y uso directo de disco.
Estas librerías facilitan la creación de aplicaciones de monitoreo al abstraer detalles complejos de bajo nivel y ofrecer una interfaz consistente en múltiples sistemas operativos.

Actualización en Timpo Real

Los monitores del sistema suelen mostrar datos dinámicos que reflejan el estado del sistema en tiempos muy cortos (típicamente cada 1-2 segundos). Para ello, se implementan bucles de actualización que consultan periódicamente las métricas y refrescan la interfaz del usuario. Este proceso puede incluir técnicas como:

- Temporizadores o interval timers
- Bucles de eventos
- Señales o callbacks
- Redibujado parcial de la interfaz de usuario

La precisión de la actualización y la eficiencia en la obtención de datos son factores que influyen directamente en el uso y en el rendimiento del monitor.

Importancia del Monitoreo del Sistema en la Ingeniería de Software

El monitoreo del sistema desempeña un papel importante en la administración de sistemas, la depuración de programas y el análisis del rendimiento de aplicaciones complejas. Permite identificar comportamientos extraños, planificar capacidad, mejorar la estabilidad en el sistema y garantizar el uso eficiente de los recursos.

Desarrollo

Para el desarrollo de este proyecto se necesitaron las siguientes librerías de Python:

psutil

Para el backend de datos, nos decantamos por **psutil** porque no tenía sentido reinventar la rueda leyendo archivos crudos del sistema (como `/proc`) o parseando comandos de terminal. Esta librería nos resuelve la abstracción de hardware de forma limpia; por ejemplo, la usamos para medir los ciclos de CPU y, en el caso de la red, implementamos una lógica manual con sus contadores para calcular la velocidad de subida y bajada en tiempo real, en lugar de solo mostrar totales históricos.

textual

En la interfaz, elegimos **textual** para darle un lavado de cara al script y pasar de una consola plana a una TUI (Text User Interface) moderna. La ventaja principal es su manejo de eventos asíncronos: montamos la aplicación sobre su clase `App`, lo que nos permite tener pestañas navegables y logs que se actualizan solos cada segundo sin que la interfaz se congele o parpadee, algo que con librerías más viejas costaría mucho programar.

io

Usamos el módulo **io** como un recurso de optimización. Necesitábamos un buffer en memoria (`StringIO`) para capturar texto temporalmente. La idea fue simular un archivo en la RAM para volcar ahí los datos del monitor en lugar de escribir en el disco duro o imprimir directo a la consola, actuando como un intermediario rápido entre el procesamiento de datos y la visualización.

contextlib

Esta fue la solución clave para la integración. Como queríamos reutilizar el código que ya teníamos (que usaba `print`) sin reescribirlo todo, usamos `redirect_stdout` de **contextlib**. Básicamente, interceptamos la salida estándar y la desviamos al buffer de memoria que creamos con `io`. Así, el código viejo "cree" que imprime en pantalla, pero nosotros capturamos eso para pintarlo en los widgets de la interfaz gráfica.

platform

Incluimos **platform** para darle contexto al usuario sobre el entorno. Nos sirve para sacar la ficha técnica del equipo al inicio: confirmar la distribución (Arch Linux en nuestro caso), la versión del kernel y la arquitectura. Es un paso de validación útil para saber si estamos corriendo en el entorno esperado.

datetime

Para el uptime, usamos **datetime** por comodidad. `psutil` nos devuelve el tiempo de encendido en segundos crudos, así que usamos la clase `timedelta` para convertir eso automáticamente a un formato legible de "días, horas y minutos", ahorrándonos tener que programar la conversión matemática manualmente.

time

La librería **time** es nuestro metrónomo. La usamos tanto para pausar el bucle de ejecución (con `sleep`) y evitar que el monitor se coma toda la CPU refrescando innecesariamente, como para tomar marcas de tiempo (`time.time`) que usamos de referencia para calcular las tasas de transferencia de red entre intervalos.

os

Mantenemos **os** por compatibilidad en la versión de terminal pura. Su función es sencilla: detectar el sistema operativo para ejecutar el comando de limpieza de pantalla correcto (`clear` en Linux, `cls` en Windows) y mantener la visualización ordenada antes de que cargue la interfaz gráfica completa.

Una vez con estas librerías instaladas se desarrollaron los siguientes métodos en el archivo **Monito.py**, donde aquí se recopilan los métodos útiles para poder recabar la información del sistema operativo de una manera ordenada y modulada en funciones a través de la librería de **psutil.** Además de usar las librerías anteriormente mencionadas como **os, time, datetime y platform**:

- clear()

Lo que hace esta función es limpiar la consola de acuerdo a los comandos ya conocidos como **cls** para Windows ó en su otra versión **clear** para sistema de basados en UNIX, como Linux o MacOS.

- bytes_to_gb(b)

Con esta función auxiliar se convierte el valor de los bytes a gigabytes para poder tener un mejor entendimiento y visualización de cuánta energía se está consumiendo, sin tener que estar convirtiendo nosotros mismo bytes a gigabytes.

- show_system_info()

En este método lo que se hace es poder mostrar la información general del sistema, donde se puede visualizar la siguiente información:

- Nombre del sistema operativo
- Versión del sistema operativo
- Arquitectura en la que se está trabajando
- Procesador del equipo
- El tiempo en el sistema operativo

Con esto se tiene en mente que sistema operativo está ocupando la máquina desde su versión hasta su arquitectura, lo cuál ayuda a un mejor entendimiento de cómo funciona y se maneja el sistema operativo de nuestro equipo.

● show_cpu()

Se muestra el uso y la información detallada del CPU, la cuál abarca la siguiente información:

- Uso total del CPU
- Número de núcleos físicos
- Número de núcleos lógicos
- Frecuencia actual
- Frecuencia máxima
- Uso por núcleo

Es importante saber cómo es que se está usando el CPU y su manejo dentro del sistema operativo, ya que nos permite identificar problemas de rendimiento como cuando se satura un procesador. Además de poder observar en qué factores o áreas se puede hacer una mejor optimización para la ejecución de las tareas y ayuda a poder aprender sobre el comportamiento del sistema operativo y del CPU.

● show_memory()

La herramienta permite observar en tiempo real el estado de la memoria RAM y la memoria SWAP, mostrando información clave para comprender cómo el sistema gestiona los recursos de memoria. Donde se puede observar los siguientes datos :

- Total de la memoria RAM
- Disponibilidad de la memoria RAM
- Porcentaje usado
- Total de la memoria SWAP
- Porcentaje usado de la memoria SWAP

Monitorear estas métricas permite **prevenir fallos y ralentizaciones** ya que si la memoria está saturada, el sistema puede congelarse o reiniciarse, por lo que es importante de vez en cuando revisar la memoria . Además de que esto va ligado con optimizar el rendimiento del sistema, identificando procesos que consumen más memoria de lo normal. Y lo más importante asegurar que siempre haya memoria disponible para las nueva tareas, garantizando la estabilidad del sistema operativo.

- show_disk()

Se muestra el uso de las particiones de disco montados, con la que se va a poder monitorear la siguiente información:

- Nombre de la partición
- Total de espacio para la partición
- Porcentaje usado de la partición
- Porcentaje libre de la partición

Es importante monitorear métricas ya que previene fallos en el sistema, aún más significativo en las particiones críticas llenas pueden impedir el arranque, la instalación de actualizaciones o la ejecución de aplicaciones. Debido a que esto puede afectar en la pérdida de datos, porque al tener una partición saturada puede corromper archivos o interrumpir procesos en curso.

- show_network()

Se muestra el total de tráfico de la red y la velocidad desde su última llamada, don lo que vamos a poder visualizar lo siguiente:

- Las subidas y bajadas totales de la red
- Velocidades de subida y bajada

La importancia de estas métricas radica en que ayuda a poder garantizar la estabilidad del sistema, ya que muchas aplicaciones dependen de una conexión estable. Poder prevenir situaciones que puedan causar lentitud o desconexiones y con este monitoreo se aumenta la seguridad, ya sea detectando comportamientos sospechosos o potencialmente maliciosos.

- show_temperatures()

Si se tiene disponibilidad, muestra la temperatura de los sensores del sistema operativo, con los siguientes datos :

- Sensor y su temperatura en grados celcius

Mostrar la temperatura de los sensores del sistema operativo es fundamental para mantener la estabilidad, seguridad y eficiencia del equipo, permitiendo prevenir daños, garantizar un desempeño óptimo y facilitar el diagnóstico de problemas térmicos o de ventilación.

- show_processes()

Se debe de mostrar una lista de los procesos principales ordenados por el uso del CPU, siguiendo de que solo se muestran los 10 primeros del CPU, para que verificar qué procesos está llevando el CPU en tiempo real, donde se obtiene los siguientes datos de los procesos:

- PID
- Nombre
- Porcentaje del CPU
- Porcentaje de la memoria RAM

Monitorear los procesos principales del sistema, especialmente aquellos que consumen más CPU y memoria, es fundamental para mantener el rendimiento y la estabilidad del sistema operativo. Permitiendo identificar procesos que consumen recursos en exceso, evitando que saturen el CPU o la memoria. Detectar comportamientos, como procesos trabados, fugas de memoria o posibles amenazas. Y por último,optimizar el rendimiento, cerrando o ajustando aplicaciones que están afectando al sistema.

- main()

En el main se juntan todas las funciones antes mencionadas para poder mostrar un monitoreo completo y detallado del sistema operativo en tiempo real, en el monitoreo se puede visualizar cuando fue la última hora en la que se actualizaron los datos.

Este código crea una aplicación de terminal interactiva (TUI) usando Textual para monitorear el sistema en tiempo real.

Funcionalidad principal:

- Interfaz con pestañas (tabs) que muestran diferentes métricas: Sistema, CPU, Memoria, Disco, Red, Temperaturas y Procesos
- Se actualiza automáticamente cada segundo
- Reutiliza funciones existentes de un módulo Monitor que ya funcionaban en terminal

Métodos:

compose(): Define la estructura de la interfaz con 7 pestañas, cada una conteniendo un RichLog para mostrar información

cap(): Captura la salida de las funciones de monitoreo (que imprimían a consola) y la convierte en texto

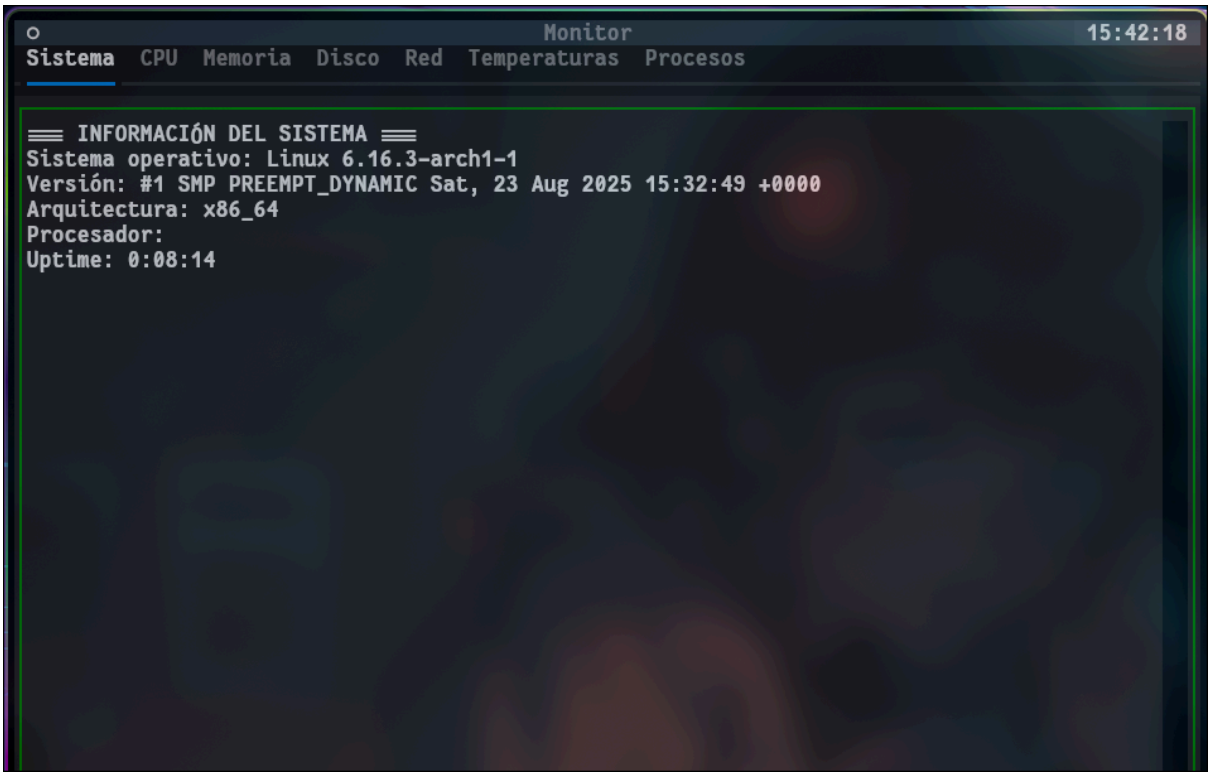
on_mount(): Inicializa la app y programa actualizaciones cada segundo

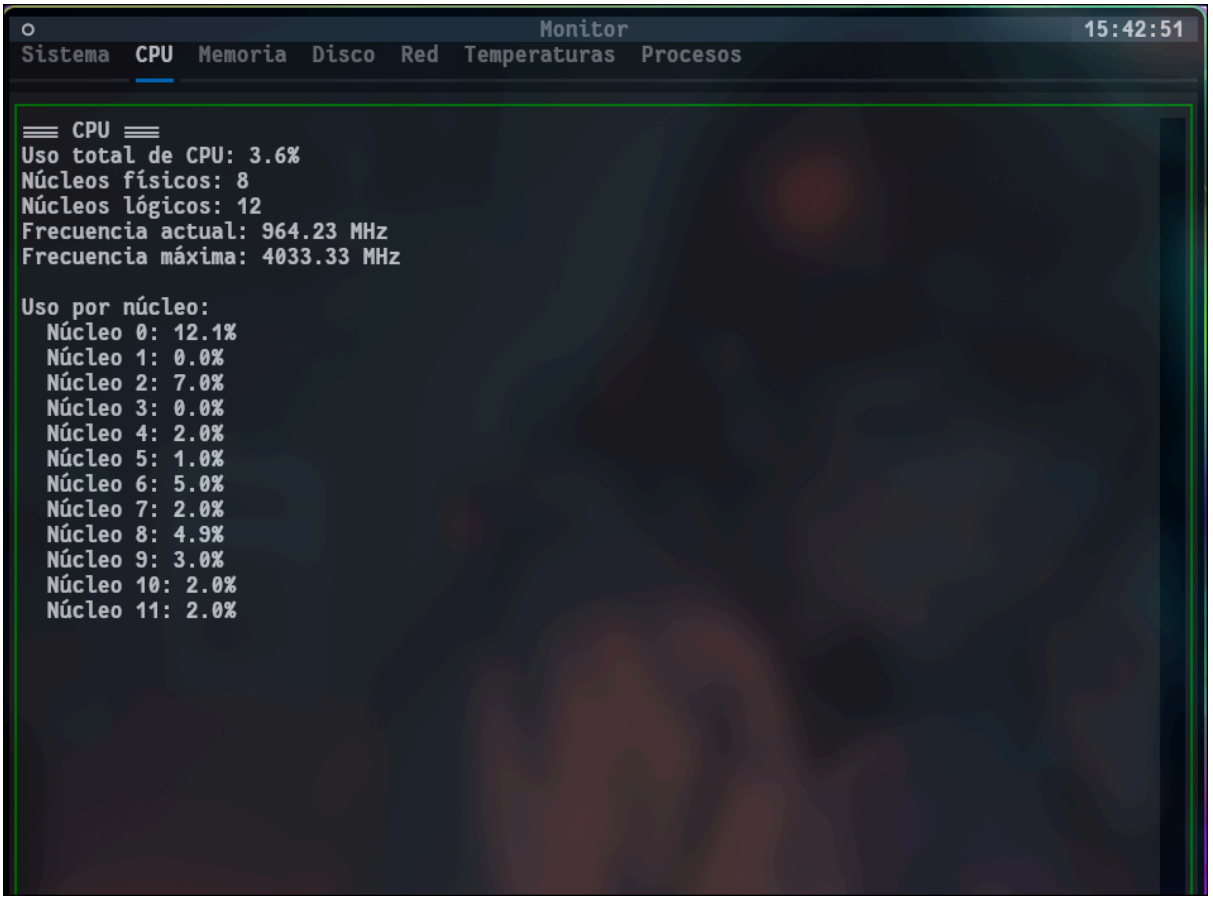
actualizar_todo(): Refresca todas las pestañas limpiando y escribiendo nueva información

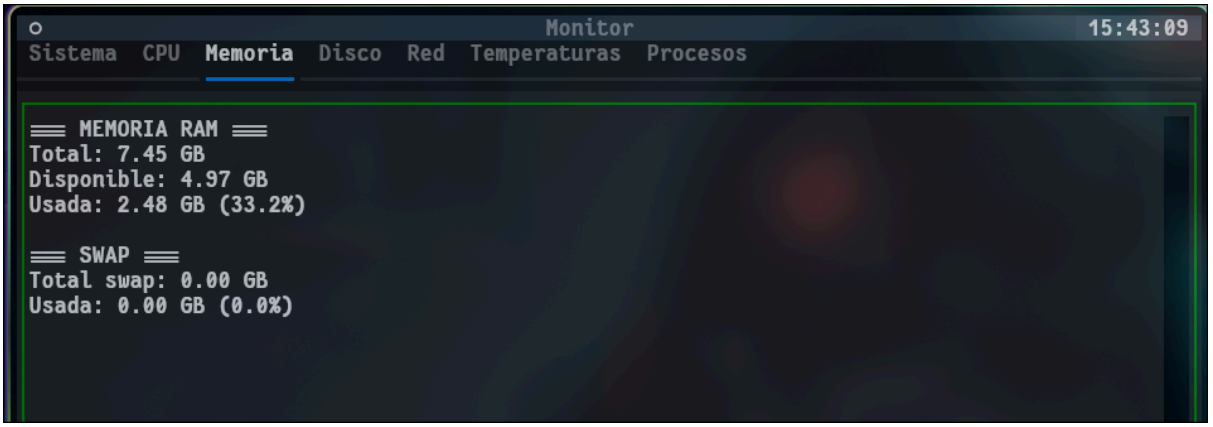
En resumen: Convierte un monitor de sistema basado en terminal a una aplicación con interfaz de pestañas que se actualiza automáticamente.

Nota: Para un mejor entendimiento de la lógica de los métodos leer el código ya documentado.

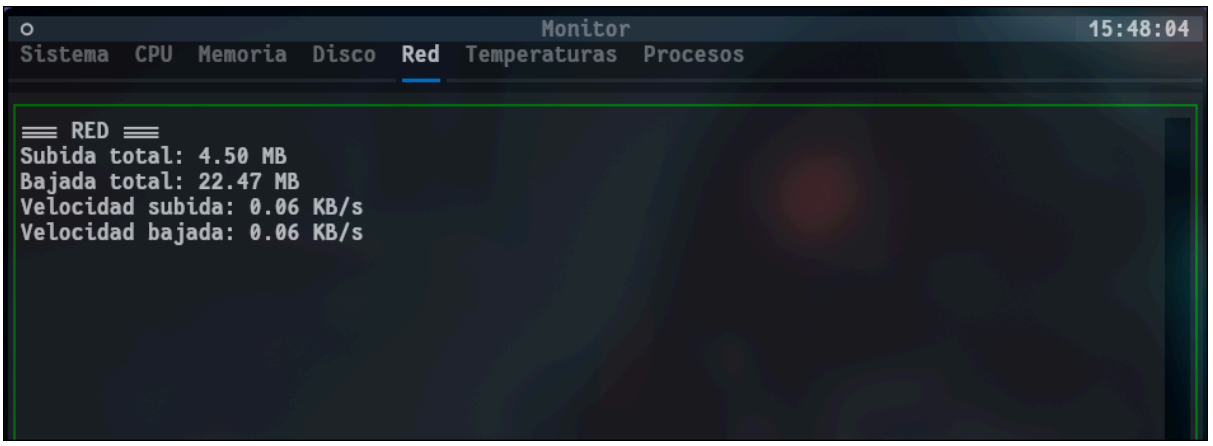
Resultados

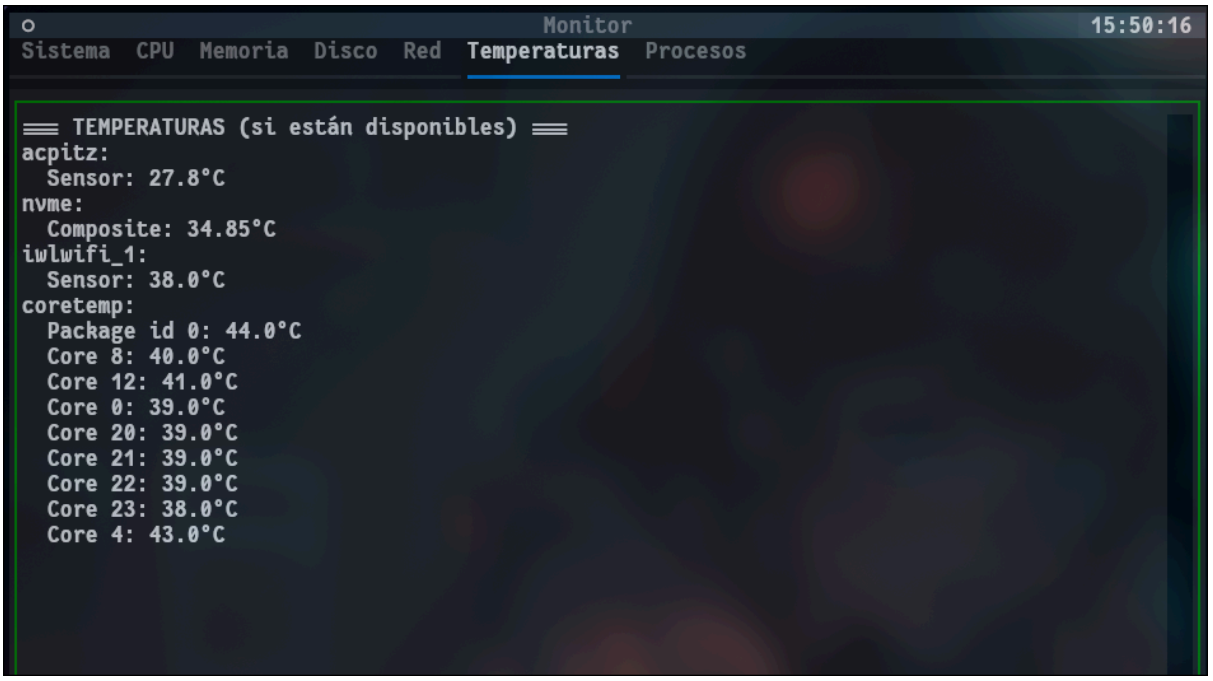

Monitor
15:42:18
Sistema CPU Memoria Disco Red Temperaturas Procesos
INFORMACIÓN DEL SISTEMA
Sistema operativo: Linux 6.16.3-arch1-1
Versión: #1 SMP PREEMPT_DYNAMIC Sat, 23 Aug 2025 15:32:49 +0000
Arquitectura: x86_64
Procesador:
Uptime: 0:08:14


Monitor
15:42:51
Sistema CPU Memoria Disco Red Temperaturas Procesos
CPU
Uso total de CPU: 3.6%
Núcleos físicos: 8
Núcleos lógicos: 12
Frecuencia actual: 964.23 MHz
Frecuencia máxima: 4033.33 MHz
Uso por núcleo:
Núcleo 0: 12.1%
Núcleo 1: 0.0%
Núcleo 2: 7.0%
Núcleo 3: 0.0%
Núcleo 4: 2.0%
Núcleo 5: 1.0%
Núcleo 6: 5.0%
Núcleo 7: 2.0%
Núcleo 8: 4.9%
Núcleo 9: 3.0%
Núcleo 10: 2.0%
Núcleo 11: 2.0%


Monitor
15:43:09
Sistema CPU Memoria Disco Red Temperaturas Procesos
MEMORIA RAM
Total: 7.45 GB
Disponible: 4.97 GB
Usada: 2.48 GB (33.2%)
SWAP
Total swap: 0.00 GB
Usada: 0.00 GB (0.0%)


Monitor
15:44:03
Sistema CPU Memoria Disco Red Temperaturas Procesos
Libre: 0.00 GB
Porcentaje: 100.0%
Partición: /dev/loop2 → /var/lib/snapd/snap/core24/1225
Total: 0.07 GB
Usado: 0.07 GB
Libre: 0.00 GB
Porcentaje: 100.0%
Partición: /dev/loop4 → /var/lib/snapd/snap/gtk-common-themes/1535
Total: 0.09 GB
Usado: 0.09 GB
Libre: 0.00 GB
Porcentaje: 100.0%
Partición: /dev/nvme0n1p1 → /boot
Total: 1.00 GB
Usado: 0.25 GB
Libre: 0.75 GB
Porcentaje: 24.7%
Partición: /dev/loop5 → /var/lib/snapd/snap/mesa-2404/912
Total: 0.28 GB
Usado: 0.28 GB
Libre: 0.00 GB
Porcentaje: 100.0%
Partición: /dev/loop6 → /var/lib/snapd/snap/snapd/25202
Total: 0.05 GB
Usado: 0.05 GB
Libre: 0.00 GB
Porcentaje: 100.0%
Partición: /dev/loop7 → /var/lib/snapd/snap/snapd/25577
Total: 0.05 GB
Usado: 0.05 GB
Libre: 0.00 GB
Porcentaje: 100.0%
Partición: /dev/loop8 → /var/lib/snapd/snap/zoom-client/258
Total: 0.44 GB
Usado: 0.44 GB
Libre: 0.00 GB
Porcentaje: 100.0%


Monitor
15:48:04
Sistema CPU Memoria Disco Red Temperaturas Procesos
RED
Subida total: 4.50 MB
Bajada total: 22.47 MB
Velocidad subida: 0.06 KB/s
Velocidad bajada: 0.06 KB/s


Monitor
15:50:16
Sistema CPU Memoria Disco Red Temperaturas Procesos
TEMPERATURAS (si están disponibles)
acpitz:
Sensor: 27.8°C
nvme:
Composite: 34.85°C
iwlwifi_1:
Sensor: 38.0°C
coretemp:
Package id 0: 44.0°C
Core 8: 40.0°C
Core 12: 41.0°C
Core 0: 39.0°C
Core 20: 39.0°C
Core 21: 39.0°C
Core 22: 39.0°C
Core 23: 38.0°C
Core 4: 43.0°C


Monitor
15:50:33
Sistema CPU Memoria Disco Red Temperaturas Procesos
PROCESOS PRINCIPALES (Top 10 CPU)
PID 1241 | Hyprland | CPU: 9.0% | RAM: 1.71%
PID 4391 | slurp | CPU: 6.0% | RAM: 0.19%
PID 1303 | waybar | CPU: 3.0% | RAM: 0.90%
PID 3246 | python | CPU: 3.0% | RAM: 0.59%
PID 483 | kworker/u48:5-flush-259:0 | CPU: 1.0% | RAM: 0.00%
PID 484 | irq/149-ELAN06FA:00 | CPU: 1.0% | RAM: 0.00%
PID 527 | kworker/u49:1+i915_flip | CPU: 1.0% | RAM: 0.00%
PID 1441 | pipewire-pulse | CPU: 1.0% | RAM: 0.22%
PID 1536 | cava | CPU: 1.0% | RAM: 0.15%
PID 1537 | sed | CPU: 1.0% | RAM: 0.05%

Conclusiones

Para el desarrollo de este monitor se ha identificado y permitido el valor de validar de manera práctica los conceptos abstractos estudiados durante el curso, especialmente sobre la administración de recursos de que se hace de un sistema operativo. Con esto podemos destacar que la visualización de las métricas del CPU, procesos y núcleos no son simples estadísticas, sino es un reflejo de cómo las decisiones que toma el núcleo afecta directamente el sistema operativo en tiempo real. Además de poder observar fluctaciones en el uso del procesador.
Si bien el sistema operativo gestiona el hardware directamente, el acceso a esta información en varias ocasiones para el usuario final requiere capas de abstracción eficientes.

El proyecto evidenció que la monitorización de un sistema operativo es un proceso inherentemente dinámico que presenta desafíos de concurrencia y eficiencia. Se concluye que para que una herramienta de este tipo sea funcional, debe operar bajo ciclos de actualización optimizados, que sean capaces de refrescar la interfaz de usuario sin generar una sobrecarga significativa en el propio sistema. Un monitor que consume excesivos recursos para medir los recursos es contraproducente; por lo tanto, se logró comprender la importancia de equilibrar la precisión de la toma de datos con la eficiencia del código, garantizando que la herramienta sirva para detectar cuellos de botella y no para crearlos.

Referencias

- Apple Inc. (s. f.). *System monitoring and activity APIs (macOS)*. Recuperado de https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/EventOverview/MonitoringEvents/MonitoringEvents.html
- Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books. https://pages.cs.wisc.edu/~remzi/OSTEP/
- *Red Hat. (s. f.). Monitoring and managing system resources. Recuperado de https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/monitoring_and_managing_system_status_and_performance/overview-of-performance-monitoring-options_monitoring-and-managing-system-status-and-performance*
- *Rodola, G. (s. f.). psutil documentation. Recuperado de https://psutil.readthedocs.io/en/latest/index.html#psutil-documentation*
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10.ª ed.). Wiley.
- Stallings, W. (2018). *Operating systems: Internals and design principles* (9.ª ed.). Pearson.
- Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4.ª ed.). Pearson.
- The Linux Project. (s. f.). *Linux man pages: Documentación de /proc y herramientas del sistema*. Recuperado de https://man7.org/linux/man-pages/man5/proc.5.html