Universidad Nacional Autónoma de México

Facultad de Ciencias

Sistemas Operativos

Proyecto
Desarrollo de un Monitor Básico del Sistema

Autor:

Delgado Díaz Hermes Alberto
319258613

27 de noviembre de 2025

1. Objetivo

Desarrollar un monitor básico que muestre información relevante del Sistema Operativo (SO) en la terminal. Este monitor debe proporcionar métricas actualizadas en tiempo real sobre los siguientes puntos:

- Uso de CPU (por núcleo).
- Uso de memoria RAM.
- Particiones de Disco y uso de espacio.
- Tasas de I/O (lectura y escritura de disco por segundo).
- Tasas de Red (envío/recepción por segundo).
- Top 10 procesos con mayor uso de CPU.
- Estado de la Batería.

2. Herramientas

- **Sistema Operativo:** `Windows 10 u 11, Linux` (ej. Ubuntu, Debian, Kali), o `macOS`. El script es compatible con la mayoría de **SO** que soporten Python.
- **Lenguaje de Programación:** `Python 3.x` (se recomienda 3.6 o superior).
- **Librería Externa:** `psutil` utilizada para obtener métricas detalladas de **CPU**, **RAM**, **Disco**, **Red**, **Procesos**, etc.
- **Librerías Estándar:** `os`, `time`, `sys`, `platform` incluidas en Python, se usan para funciones básicas del sistema, manejo de tiempo y metadatos.

3. Introducción

El monitoreo de recursos es una tarea fundamental en la administración de cualquier sistema operativo. Permite a los usuarios y administradores comprender el estado operativo del hardware y el software, identificar cuellos de botella y diagnosticar problemas de rendimiento.

Este proyecto se centra en el desarrollo de un monitor del sistema de línea de comandos, una herramienta ligera y de bajo consumo que proporciona una vista consolidada y en tiempo real de los recursos más críticos. A diferencia de los monitores gráficos más pesados, una herramienta de terminal es útil para entornos de servidor, administración remota de sistemas, o cuando se requiere una revisión rápida del estado del sistema con recursos mínimos.

El script está diseñado para ser multiplataforma y auto-actualizable, mostrando información detallada de la CPU (con métricas individuales por núcleo), el uso de memoria RAM, las particiones de disco con sus respectivos sistemas de archivos, las tasas de transferencia de red (envío y recepción), las tasas de entrada/salida de disco, el estado de la batería, y los procesos que están consumiendo mayor cantidad de recursos del procesador. Esta combinación de métricas proporciona una visión completa del rendimiento del sistema, permitiendo identificar rápidamente qué componentes están bajo mayor carga y facilitando la toma de decisiones para optimizar el rendimiento o diagnosticar problemas.

4. Marco Teórico

El monitor utiliza la librería `psutil` de Python, un módulo multiplataforma que accede a detalles sobre el proceso en ejecución y la utilización del sistema (CPU, memoria, discos, red, sensores).

Conceptos Clave del Monitoreo de Recursos:

1. **Métricas de CPU (Unidad Central de Procesamiento):** El monitor obtiene el porcentaje de uso por cada núcleo (*cpu_percent(..., percpu=True)*). Es crucial monitorear el uso por núcleo para detectar desequilibrios de carga.

2. **Memoria RAM (Memoria de Acceso Aleatorio):** Se mide el uso total, usado y el porcentaje. La Memoria Virtual es la principal métrica que el script consulta (*psutil.virtual_memory()*).

3. **Particiones de Disco y Uso:** El script itera sobre las particiones montadas (*psutil.disk_partitions()*) y calcula el uso de espacio (*psutil.disk_usage()*) en cada una, incluyendo el tipo de sistema de archivos (*fstype*).

4. **Tasas de I/O y Red:** Las tasas de lectura/escritura de disco y envío/recepción de red se calculan midiendo la diferencia entre los contadores de bytes en dos puntos de tiempo (*dt*) y dividiendo por el intervalo de tiempo (*(bytes_now - bytes_prev) / dt*).

5. **Procesos:** El monitor lista los procesos activos, obteniendo su PID, Nombre, Estado y Porcentaje de Uso de CPU (*cpu_percent*), para luego ordenarlos y mostrar el Top 10: *top_procesos = sorted(procesos, key=lambda p: p['cpu_percent'] or 0, reverse=True)[:10].*

6. **Batería:** Se consulta el nivel de carga y el estado de conexión a la corriente (*power_plugged*) (*psutil.sensors_battery()*).

5. Desarrollo

El monitor fue desarrollado en un único script llamado `Monitor.py`.

Estructura y Clases

- **Clase** `Monitor`: Es la clase principal encargada de la lógica de obtención de datos.
  - Su método `__init__` inicializa las variables de estado, incluyendo la información del SO (`os_type, hostname, kernel`) y los contadores previos de red y disco para el cálculo de tasas.
  - El método `obtener_metricas` es el núcleo, donde se invocan las funciones de `psutil` para recopilar todas las métricas. Es fundamental la lógica de cálculo del **tiempo transcurrido** (`dt`) para asegurar la precisión de las tasas de I/O y Red en cada ciclo de actualización.

Funciones Auxiliares de Visualización

Para una visualización clara en la terminal, se crearon las siguientes funciones auxiliares:

- `limpiar_pantalla()`: Borra el contenido previo de la terminal para simular la actualización en tiempo real.
- `formatear_bytes()`: Convierte valores en bytes a formatos legibles (KB, MB, GB, TB).
- `crear_barra()`: Genera una representación visual de barras de progreso.
- `obtener_color_barra()`: Asigna códigos de color ANSI a las barras de progreso según el porcentaje (Verde $<50\,\%$, Amarillo $<80\,\%$, Rojo $>= 80\,\%$), mejorando la legibilidad.
- `mostrar_metricas()`: Formatea y aplica estilos de color a todas las métricas recopiladas para la visualización final en la terminal.

Guía de Ejecución

El script requiere la instalación de la dependencia externa antes de su ejecución.

1. **Instalar** `psutil`:

```
?- pip install psutil
```

2. **Ejecutar el Monitor**:

```
?- python Monitor.py
```

3. **Detener**: La ejecución puede detenerse el cualquier momento presionando `Ctrl + C`.

Repositorio de GitHub

https://github.com/HermesADD/SistemasOperativosProyecto.git

6. Resultados

Los resultados demuestran la funcionabilidad y la compatibilidad en diferentes plataformas del monitor del sistema. Se muestran métricas precisas tanto en un entorno `Windows 11 nativo` como en una máquina virtual `Kali Linux`.

Resultado en Windows

A. **Especificaciones del Sistema Host**

El monitor se ejecutó inicialmente en un sistema operativo host con las siguientes características:

- **Nombre del dispositivo:** Hermes-Laptop
- **Procesador:** AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx (2.10 GHz)
- **RAM Instalada:** 8.00 GB (6.94 GB utilizable)
- **Tipo de Sistema:** Sistema Operativo de 64 bits, procesador x64.

B. **Análisis de la Salida del Monitor**

- **Uso de CPU:** El sistema cuenta con **8 núcleos** (Core 0 a Core 7). El Core 3 muestra el pico de uso más alto con **45.8 %**, mientras que otros núcleos tienen un uso bajo (ej. Core 5 con 4.9 %)
- **Memoria RAM:** Se utiliza el **76.8 %** de la memoria disponible (5.33 GB de 6.94 GB).
- **Disco I/O:** Las tasas de transferencia de lectura y escritura son relativamente bajas, siendo la lectura de **181.38 KB/s** y la escritura de **914.03 KB/s**.
- **Particiones de Disco:** Se muestra la partición principal **C:** con un uso del **44.3 %** (210.70 GB de 476.11 GB) y el sistema de archivos **NTFS**.
- **Red:** Las tasas de red son mínimas (TX: 437.01 B/s, RX: 621.06 B/s), indicando una actividad de red muy ligera.
- **Batería:** El nivel es del 100 % y el estado es "Conectado", indicando que el equipo está cargando o ya está completamente cargado.
- **Top 10 Procesos:** Los procesos con mayor consumo de CPU son System Idle Process (687.5 %), Python.exe (52.3 %) y dwm.exe (14.7 %). El alto porcentaje de System Idle Process es normal en Windows, ya que representa el tiempo que la CPU pasa inactiva.

```
================================================================================
  MONITOR DEL SISTEMA  - 2025-11-26 17:01:20
================================================================================

[CPU USAGE - 8 cores]
  Core  0:                                15.5%
  Core  1:                                 7.6%
  Core  2:                                14.4%
  Core  3:                                45.8%
  Core  4:                                10.3%
  Core  5:                                 4.9%
  Core  6:                                 7.6%
  Core  7:                                 9.7%

[MEMORIA RAM]
  Usado: 5.33 GB / 6.94 GB (76.8%)

[DISCO I/O]
  Read:  181.38 KB/s
  Write: 914.03 KB/s

[PARTICIONES DE DISCO]
  C:\ (C:\)
    210.70 GB / 476.11 GB (44.3%)
    Libre: 265.40 GB | Tipo: NTFS

[RED]
  ↑ TX: 437.01 B/s
  ↓ RX: 621.06 B/s

[BATERÍA]
  Nivel: 100% (Conectado)

[TOP 10 PROCESOS]
  PID         NOMBRE                ESTADO      CPU %
  ------------------------------------------------------------
  0           System Idle Process   running     687.5%
  20416       python.exe            running      52.3%
  2968        MemCompression        running      15.4%
  7544        dwm.exe               running      14.7%
  35696       conhost.exe           running       3.6%
  4           System                running       2.8%
  10916       GooglePlayGamesServi  running       2.8%
  4620        MsMpEng.exe           running       2.1%
  4332        BrYNSvc.exe           running       1.4%
  18192       iPSMonitor.exe        running       1.4%

================================================================================
Actualización cada 1000ms | Presiona Ctrl+C para salir
```

Resultados en Máquina Virtual con Linux Kali

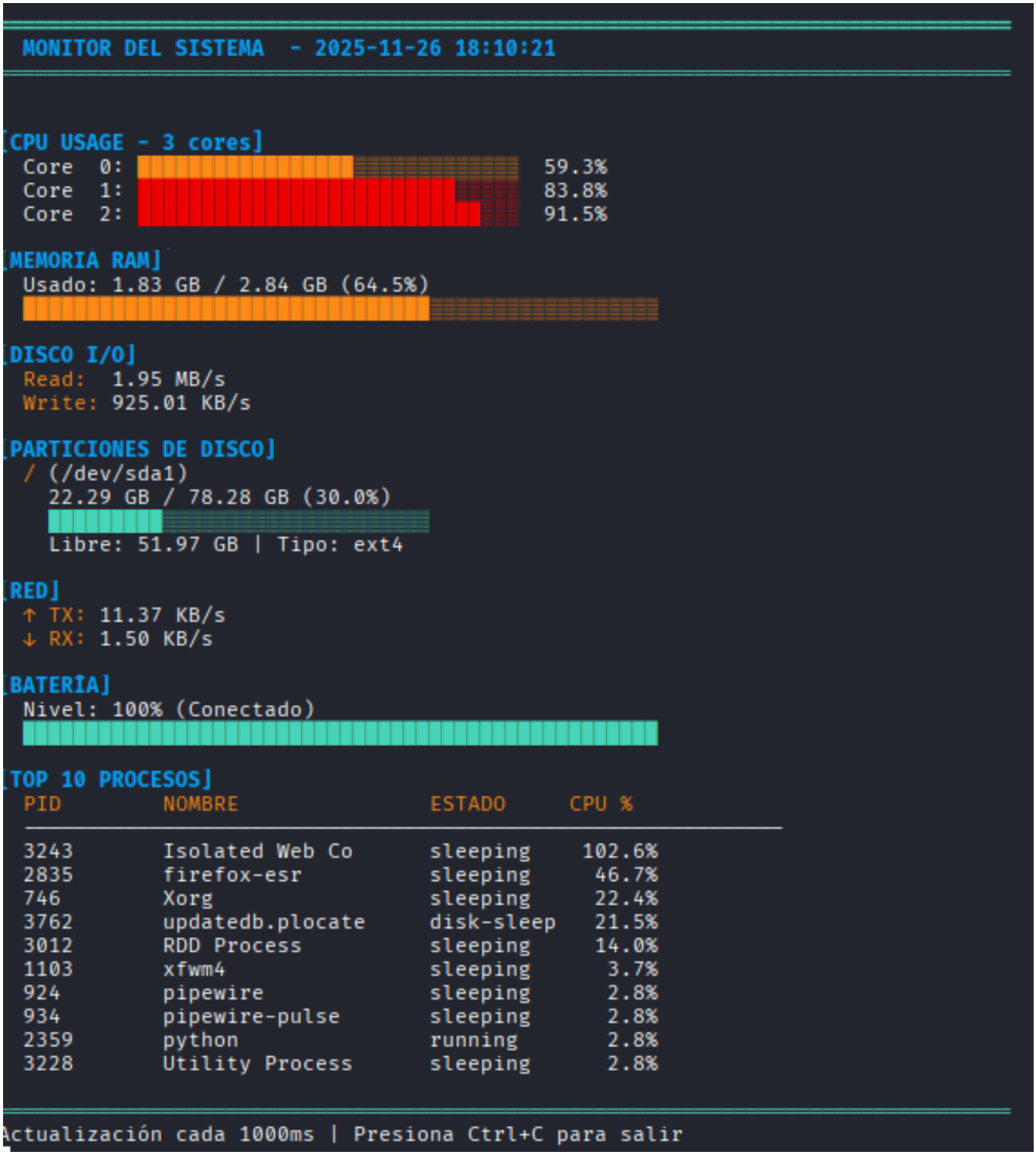
A. **Especificaciones del Entorno Virtual**

El monitor se ejecutó dentro de una máquina virtual corriendo Kali Linux (Debian) con las siguientes asignaciones de recursos en VirtualBox:

- **Memoria RAM**: 3000 MB
- **Procesadores**: 3
- **Almacenamiento**: 80.9 GB

B. **Análisis de la Salida del Monitor**

- **Uso de CPU:** La VM utiliza **3 núcleos**. Se observa una carga muy alta, con los núcleos 0, 1 y 2 con usos de **59.3 %**, **83.8 %** y **91.5 %** respectivamente, lo que podría deberse a la actividad en la VM al momento de la captura.
- **Memoria RAM:** El uso es del **64.5 %** (1.83 GB de 2.84 GB).
- **Disco I/O:** Se registra una alta actividad de lectura de disco con 1.95 MB/s, y una baja escritura de **925.01 KB/s**.
- **Particiones de Disco:** Se muestra la partición principal **/dev/Sda1** con un uso del **30.0 %** (22.29 GB de 78.28 GB) y el sistema de archivos **ext4**, estándar en Linux.
- **Red:** Las tasas de red son bajas (TX: 11.37 KB/s, RX: 1.50 KB/s).
- **Batería:** El nivel es del **100 %** y el estado es **"Conectado"**, ya que el sistema operativo virtualizado reporta el estado del anfitrión o una batería virtual constante.
- **Top 10 Procesos:** El consumo de CPU es dominado por procesos asociados a la navegación web y al sistema gráfico: Isolated Web Co (102.6 %), firefox-esr (46.7 %) y Xorg (22.4 %).


MONITOR DEL SISTEMA - 2025-11-26 18:10:21
[CPU USAGE - 3 cores]
Core 0: 59.3%
Core 1: 83.8%
Core 2: 91.5%
[MEMORIA RAM]
Usado: 1.83 GB / 2.84 GB (64.5%)
[DISCO I/O]
Read: 1.95 MB/s
Write: 925.01 KB/s
[PARTICIONES DE DISCO]
/ (/dev/sda1)
22.29 GB / 78.28 GB (30.0%)
Libre: 51.97 GB | Tipo: ext4
[RED]
↑ TX: 11.37 KB/s
↓ RX: 1.50 KB/s
[BATERIA]
Nivel: 100% (Conectado)
[TOP 10 PROCESOS]
PID NOMBRE ESTADO CPU %
3243 Isolated Web Co sleeping 102.6%
2835 firefox-esr sleeping 46.7%
746 Xorg sleeping 22.4%
3762 updatedb.plocate disk-sleep 21.5%
3012 RDD Process sleeping 14.0%
1103 xfwm4 sleeping 3.7%
924 pipewire sleeping 2.8%
934 pipewire-pulse sleeping 2.8%
2359 python running 2.8%
3228 Utility Process sleeping 2.8%
Actualización cada 1000ms | Presiona Ctrl+C para salir

7. Conclusiones

El desarrollo de este monitor básico del sistema cumplió exitosamente con el objetivo de proporcionar una herramienta ligera y eficiente para el monitoreo en tiempo real de recursos críticos del sistema operativo. A través de la implementación en Python utilizando la librería psutil, se logró crear una solución multiplataforma capaz de funcionar consistentemente en Windows, Linux y macOS.

La implementación demostró ser altamente efectiva en la captura y visualización de métricas del sistema. El uso de la arquitectura basada en clases permitió mantener un código organizado y modular, facilitando el cálculo preciso de tasas de I/O y red mediante la gestión de estados previos y deltas de tiempo. La interfaz de terminal, resultó intuitiva gracias al uso de códigos ANSI para colores y barras de progreso visuales que mejoran significativamente la interpretación de los datos.

El proyecto reforzó conceptos fundamentales de sistemas operativos, incluyendo la gestión de procesos, el monitoreo de recursos hardware y la interacción con el kernel a través de APIs especializadas. Se comprendió la importancia del cálculo diferencial para obtener tasas de transferencia precisas y la necesidad de manejar correctamente los intervalos de tiempo en sistemas de monitoreo en tiempo real.

Referencias

[1] Giampaolo Rodola, ”psutil documentation”, https://psutil.readthedocs.io/en/latest/

[2] GeeksforGeeks, ”Psutil module in Python ”, https://www.geeksforgeeks.org/python/psutil-module-in-python/

[3] Kurtuluş Cömert, ”The Art of System Monitoring in Python: Meet psutil”, https://medium.com/@licodingdev/the-art-of-system-monitoring-in-python-meet-psutil-eafe28f161cd

[4] Python Software Foundation, ”psutil 7.1.3”, https://pypi.org/project/psutil/

[5] Python Software Foundation, ”os — Interfaces misceláneas del sistema operativo”, https://docs.python.org/es/3.10/library/os.html

[6] Python Software Foundation, ”sys — System-specific parameters and functions”, https://docs.python.org/es/3/library/sys.html

[7] Python Software Foundation, ”platform — Access to underlying platform’s identifying data”, https://docs.python.org/es/3/library/platform.html

[8] Python Software Foundation, ”time — Time access and conversions”, https://docs.python.org/3/library/time.html