Universidad Nacional Autónoma de México
Facultad de ciencias
Ciencias de la Computación
Equipo: 6
Equipo:

- García Gómez Eduardo Biali - 320113987

- Mendiola Montes Victor Manuel - 320197350

- Flores Muñoz Mauricio - 320236589

- Hurtado Aponte Joshua Abel - 320176900

Proyecto Final: Desarrollo de un monitor básico del sistema Implementa una herramienta sencilla que muestre información relevante del SO: uso de CPU, memoria, procesos, disco, red, etc.

Fecha de entrega: 27/Noviembre/2025

Introducción

El proyecto consiste en desarrollar un monitor básico del sistema operativo que muestre información relevante en tiempo real, en particular, presentaremos métricas de uso de CPU, memoria y procesos, y también métricas de I/O como actividad de disco y red, el monitor podrá ejecutarse en la línea de comandos y actualizará sus datos de forma periódica para observar el estado del sistema.

Objetivos

Objetivo general

Desarrollar un monitor de sistema en tiempo real que muestre métricas de CPU, memoria, procesos, disco y red desde una interfaz en la terminal.

Objetivos específicos

- Actualizar al menos dos métricas dinámicas cada 1 segundo (por ejemplo, %CPU total/por núcleo y uso de memoria).
- Mostrar una tabla de procesos con PID, nombre, %CPU y %MEM.
- Reportar I/O de disco (lecturas/escrituras) y red (bytes enviados/recibidos).
- Documentar la integración y las librerías utilizadas para acceder a los recursos del sistema.

Marco teórico

- **Proceso:** Un proceso es un programa en ejecución identificado por un PID. El sistema operativo administra sus recursos (CPU, memoria, archivos) y mantiene información de su estado. En Modern Operating Systems se describe cómo los procesos son la unidad básica de planificación y ejecución.
- **Estados de proceso:** Un proceso puede estar running (ejecutándose), sleeping/blocked (esperando I/O o un evento), stopped (detenido) o zombie (ya terminó pero aún existe su entrada para que el padre lea su estado). Estos estados permiten al SO coordinar CPU e I/O de forma eficiente.
- **Planificación y carga del sistema (Load Average):** El load average resume cuánta demanda hay sobre la CPU (procesos listos o en ejecución). Es un indicador práctico de saturación y competencia por CPU.
- **Cambios de contexto (context switches):** Ocurren cuando el SO cambia la CPU de un proceso a otro. Una tasa alta puede indicar muchas tareas compitiendo por CPU, bloqueos frecuentes o alta actividad del sistema.
- **Interrupciones:** Son señales al CPU para atender eventos (temporizador, disco, red, etc.). Su tasa puede aumentar con cargas de I/O o con actividad intensa del sistema.
- **Uso de CPU:** Porcentaje de tiempo en que el procesador ejecuta instrucciones de procesos del usuario o del sistema.

- **Memoria (RAM):** recurso principal para que los procesos mantengan datos y estados de ejecución, se reportarán valores básicos (usada, disponible).
- **I/O (E/S) de disco y red:** operaciones de lectura/escritura en almacenamiento y transferencia de datos a través de la red; sirven para identificar carga de entrada/salida.
- **Métricas dinámicas:** valores que cambian con el tiempo; se medirán mediante muestreos periódicos para observar variaciones.
- **Fuentes de datos del SO:** las métricas se obtienen mediante llamadas del sistema o interfaces expuestas por el SO (por ejemplo, /proc en Linux), accesibles desde Python a través de la librería psutil.

Desarrollo

El proyecto se desarrolló separando claramente la recolección de información del sistema y la presentación en la terminal, con el objetivo de mantener el código modular y fácil de extender. Para ello, se implementó un muestreador que construye periódicamente un snapshot (metrics.py) y, por otro lado, una interfaz que consume ese snapshot y lo muestra de forma ordenada y legible en consola (cli.py).

En la parte de recolección de datos se implementó una clase encargada de generar el snapshot. Este snapshot integra métricas de CPU, como el porcentaje total de uso, el uso por núcleo y el load average (1, 5 y 15 minutos), además de indicadores del comportamiento interno del sistema como la tasa de cambios de contexto por segundo y la tasa de interrupciones por segundo. Estas métricas permiten observar no sólo cuánto se usa la CPU, sino también qué tan activa está la alternancia de procesos y la atención a eventos del hardware, lo cual se relaciona con lo estudiado en sistemas operativos.

También se integraron métricas de memoria, incluyendo el total, disponible, usada y el porcentaje de uso de RAM, así como el uso de swap (total, usada y porcentaje). Incluir swap ayuda a detectar presión de memoria: cuando la RAM no alcanza, el sistema recurre a memoria virtual en disco, lo que puede degradar el rendimiento. Todo esto se incorpora al snapshot de forma consistente para poder mostrarlo en la interfaz.

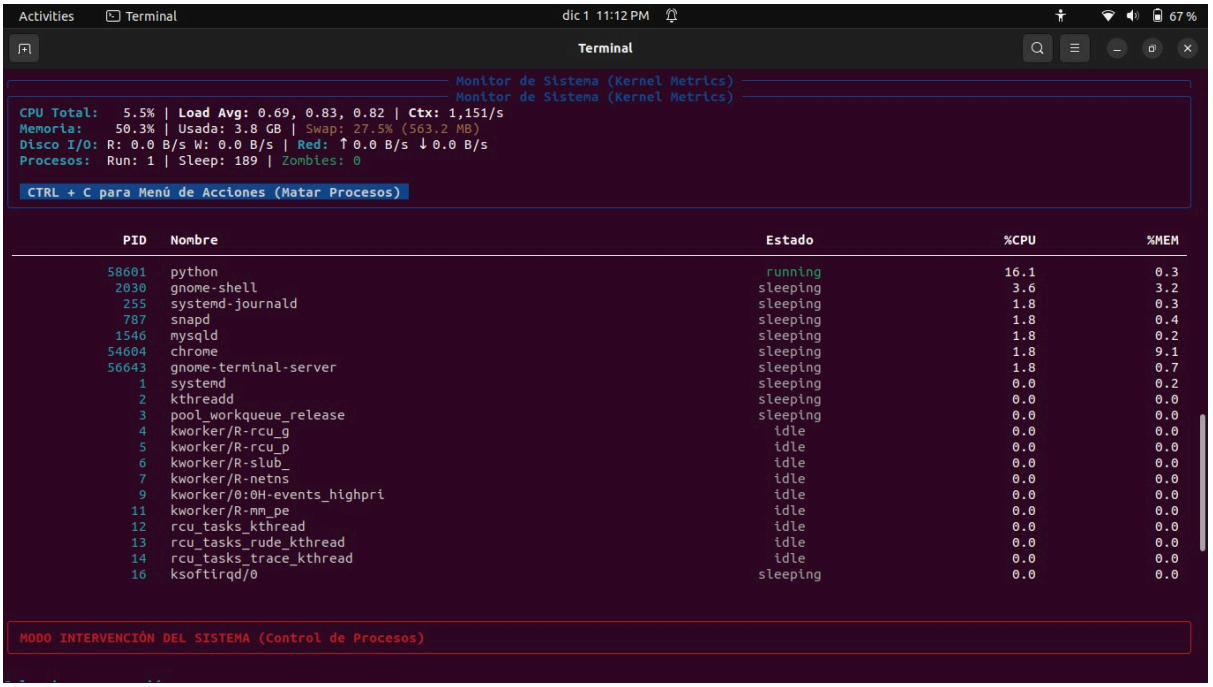
Para el apartado de procesos, el muestreador recorre los procesos activos y construye una lista con los top procesos ordenados por %CPU. Para cada proceso se captura su PID, nombre, estado, %CPU y %MEM. Además, se genera un resumen con el conteo de procesos por estado (por ejemplo running, sleeping, stopped, zombie), de manera que el monitor no solo muestre el “top” sino también una vista general del comportamiento del sistema. Durante esta recolección se manejan excepciones comunes (por ejemplo procesos que terminan mientras se leen o procesos sin permisos), evitando que el programa falle por lecturas momentáneamente inválidas.

En cuanto a disco y red, varias métricas llegan como contadores acumulados desde el arranque (bytes leídos/escritos o enviados/recibidos). Para poder mostrarlas como “por segundo”, se guardan lecturas previas y se calcula una tasa usando la diferencia entre el valor actual y el anterior, dividida entre el tiempo transcurrido entre muestras. Con esto, el monitor puede mostrar de forma clara picos de lectura/escritura y de subida/bajada, útiles para detectar cargas de I/O.

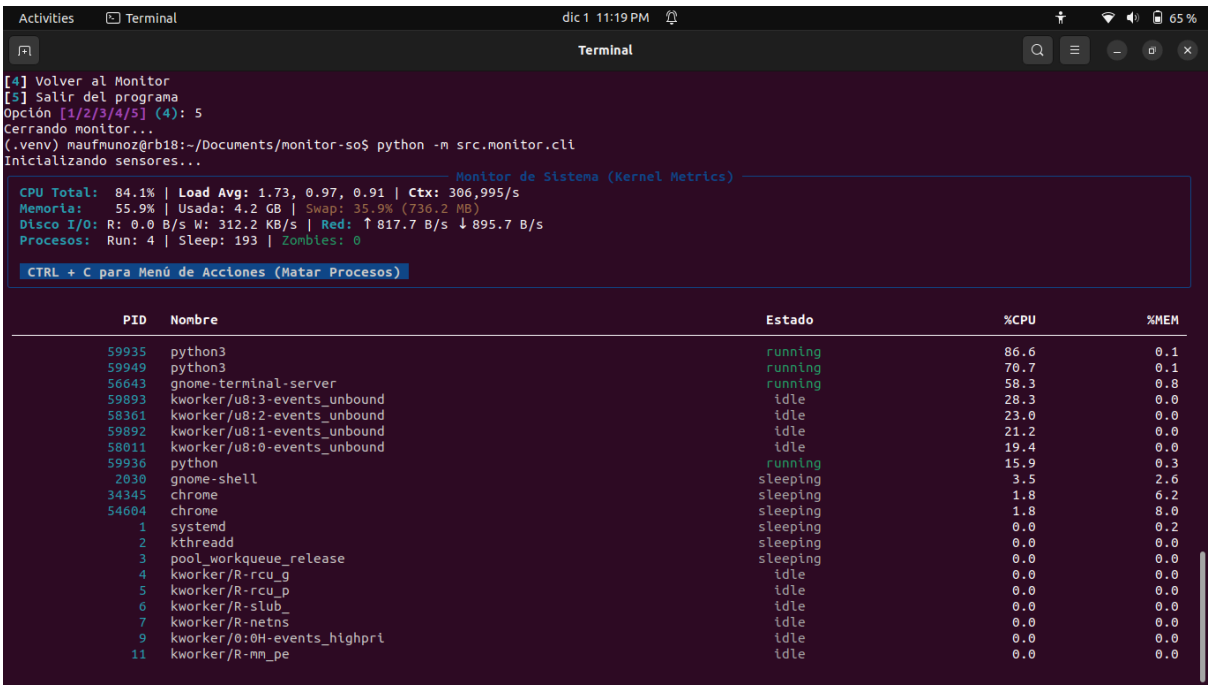
Finalmente, la interfaz en terminal se implementó con Rich, mostrando una cabecera de métricas y una tabla de procesos, refrescando en un ciclo continuo con una actualización aproximada de 0.5 segundos para mantener fluidez. Un punto importante del programa final es que Ctrl+C ya no solo cierra, sino que activa un modo de intervención: aparece un menú desde el cual el usuario puede realizar acciones sobre un PID, como terminar un proceso (SIGKILL), suspenderlo (SIGSTOP) o reanudarlo (SIGCONT). Esto conecta el monitor con una parte práctica de administración de procesos en sistemas operativos, ya que además de observar el sistema, permite controlarlo de forma segura y explícita desde la interfaz.

Resultados

Para validar el funcionamiento del monitor, se ejecutó el programa en la terminal y se observaron las métricas en tiempo real. En condiciones normales (sin cargas pesadas), la cabecera mostró un uso de CPU bajo y estable, con valores de memoria coherentes con los procesos del sistema, mientras que las tasas de disco y red se mantuvieron cercanas a cero. La tabla listó los procesos activos ordenados por %CPU, permitiendo identificar rápidamente cuáles estaban consumiendo más recursos, y el resumen de estados reflejó la distribución típica (mayoría sleeping).

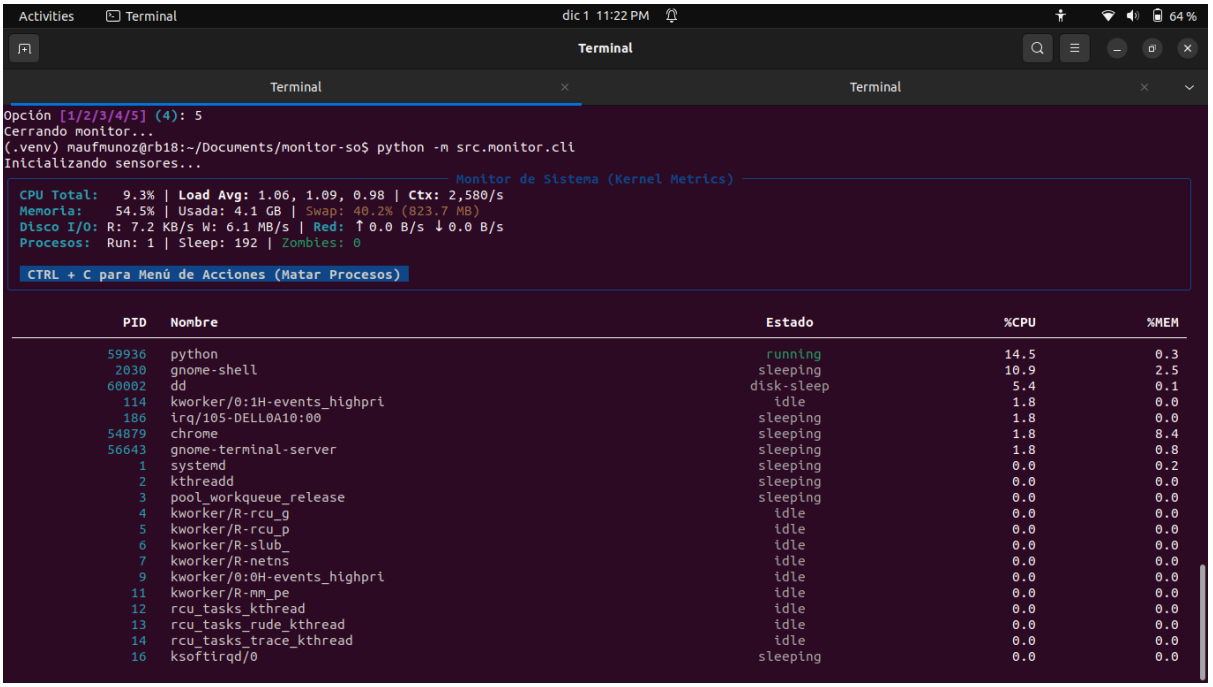


Posteriormente, se probó el comportamiento bajo carga de CPU ejecutando un proceso que consume CPU de forma continua (por ejemplo un bucle infinito). En este escenario se observó un incremento notable en el CPU total y el proceso de prueba apareció en los primeros lugares de la tabla. Además, el load average tendió a aumentar cuando la carga se sostuvo por más tiempo, y se apreció variación en la tasa de cambios de contexto por segundo, ya que el sistema alterna ejecución entre procesos y tareas del propio sistema. Esto confirmó que el monitor refleja correctamente el aumento de demanda sobre el procesador y su efecto en indicadores del sistema.

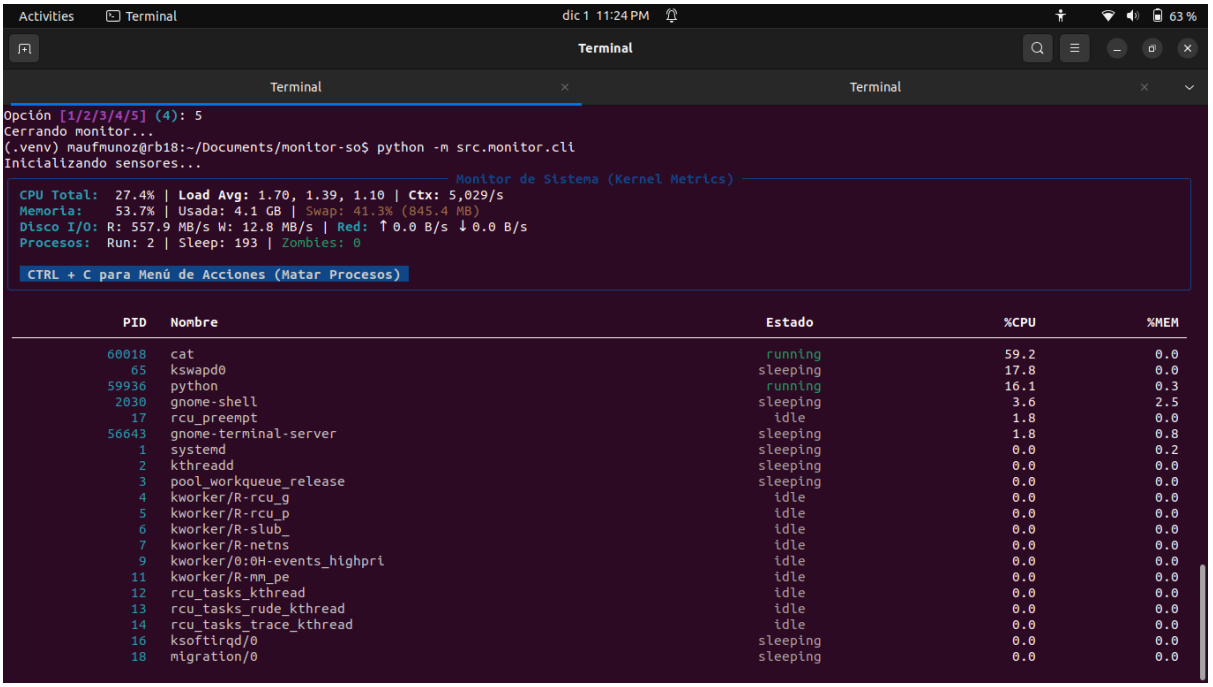


Para evaluar el monitoreo de disco, se generó actividad de escritura y lectura (por ejemplo creando un archivo grande y luego leyéndolo). Durante la escritura se observaron picos en write/s, y durante la lectura picos en read/s, regresando a valores cercanos a cero una vez concluida la operación. En la tabla también surgieron los procesos responsables de dichas operaciones, lo que permitió correlacionar de forma directa “quién” estaba provocando el uso de disco con “cuánto” uso estaba ocurriendo.

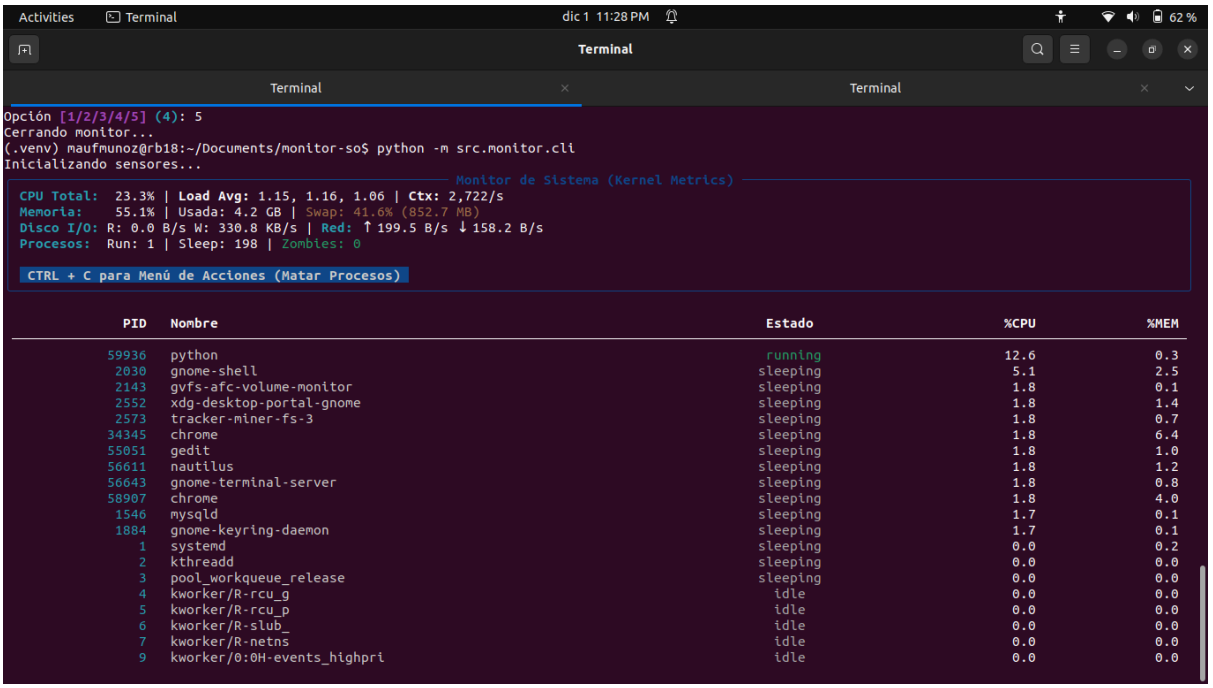
Generamos un archivo pesado: dd if=/dev/zero of=~/archivo_prueba.bin bs=4M count=300 status=progress



Hacemos la prueba de lectura: cat ~/archivo_prueba.bin > /dev/null

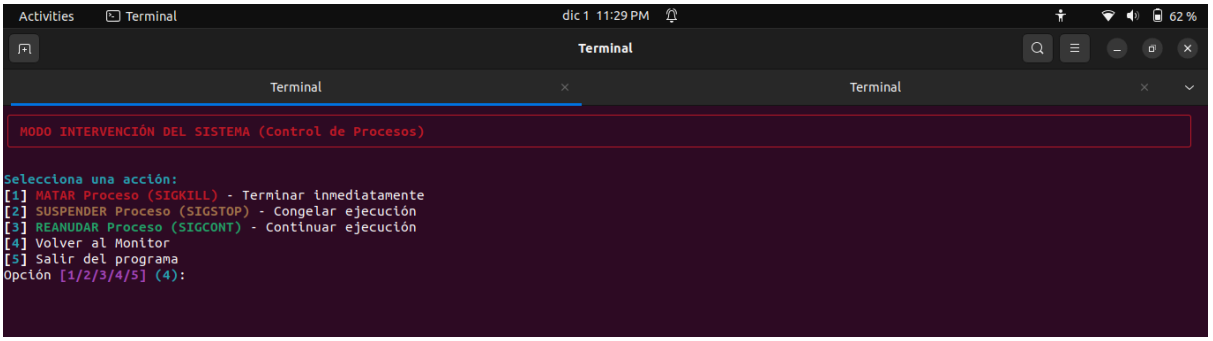


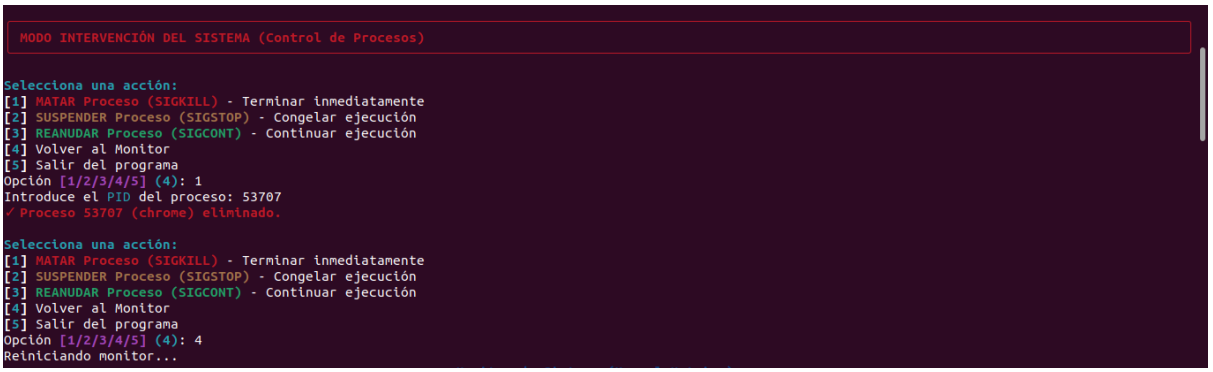
Para medir tráfico, descargamos un archivo desde internet, la cabecera mostró incrementos en bajada por segundo durante la descarga y en menor medida, subida por segundo (conexiones y confirmaciones), la tabla de procesos ubicó la herramienta de descarga visible entre los más activos mientras duró la transferencia. Al finalizar, los valores de red bajaron nuevamente.

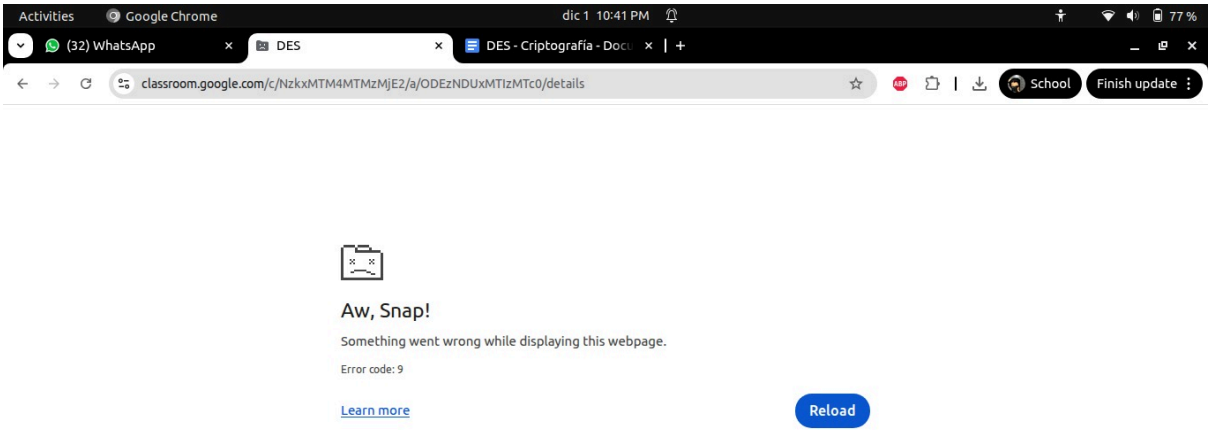


Finalmente se probó el modo de intervención activado con Ctrl+C, verificando que el programa no termina inmediatamente, sino que abre un menú de acciones para

administrar procesos. Se probó la suspensión de un proceso (SIGSTOP), observando que su actividad disminuye o se detiene, y posteriormente su reanudar (SIGCONT), confirmando que vuelve a ejecutarse. También se verificó la opción de terminar procesos (SIGKILL) en casos seguros. Esta funcionalidad demuestra que el sistema no solo permite observar el estado del equipo, sino también aplicar control de procesos desde la misma herramienta.







Conclusiones

Eduardo:
Para mí, lo más valioso fue definir un snapshot con estructura fija como contrato entre la recolección de datos y la interfaz. Esa decisión permitió mantener el monitor estable y fácil de extender: pudimos agregar métricas nuevas (como load average, swap, conteo de estados y tasas de cambios de contexto/interrupciones) sin tener que reescribir la parte visual. Además, al separar claramente “muestreo” y “presentación”, confirmé una idea clave de sistemas operativos: observar el estado del sistema requiere consistencia y periodicidad; la herramienta logra eso con actualizaciones frecuentes y sin romperse aunque el sistema cambie entre muestras.

Joshua:
Trabajar con procesos e I/O me ayudó a entender dos cosas fundamentales en la práctica: primero, que el %CPU por proceso puede superar 100% en multicore (porque un proceso puede usar varios núcleos), y segundo, que disco y red suelen entregarse como contadores acumulados, así que para ver “velocidad” hay que convertirlos a tasas por segundo usando diferencias entre muestras. También fue importante integrar el estado de los procesos y manejar excepciones al iterarlos (procesos que mueren o niegan permisos), porque eso es parte del comportamiento real del SO. Finalmente, el menú de Ctrl+C conecta la observación con la administración: al aplicar señales (parar, continuar o terminar) se entiende mejor el control de procesos descrito en Tanenbaum.

Mauricio:
Al construir la cabecera en la terminal, mi objetivo fue que cualquiera pudiera leer el estado del sistema de manera sencilla. El formateo de bytes y el uso de Rich ayudaron a que CPU, memoria, disco y red se entendieran de forma inmediata, sin números difíciles de leer. Un desafío técnico puntual fue lograr que los estilos se aplicaran correctamente; descubrí que la librería imprimía las etiquetas literalmente ([bold]), por lo que tuve que implementar Text.from_markup para forzar al motor de renderizado a interpretar las negritas. Confirmé que el refresco cada ~1 segundo se percibe fluido y que al generar actividad (cálculo, copia de archivos, descarga) la cabecera refleja los cambios de manera clara. La interfaz no recalcula nada: solo muestra el snapshot, y eso le da estabilidad. Para mí, el aprendizaje principal fue traducir métricas del sistema a una presentación simple y consistente con lo que pide la práctica.

Victor:
En la tabla de procesos me enfoqué en la experiencia de lectura: ordenar por consumo de CPU, alinear columnas y añadir información útil para diagnóstico, como el estado del proceso, además de PID, nombre, %CPU y %MEM. También cuidé que el muestreo fuera estable (por ejemplo el “priming” del CPU por proceso) para evitar lecturas engañosas al inicio. Un cambio importante respecto a una salida “simple” es que Ctrl+C ahora abre un menú de acciones, lo cual hace la herramienta

más completa: no solo muestra qué proceso consume recursos, sino que permite intervenir (suspender/reanudar/terminar) de forma explícita. Esto refuerza el aprendizaje de administración de procesos y señales del sistema operativo.