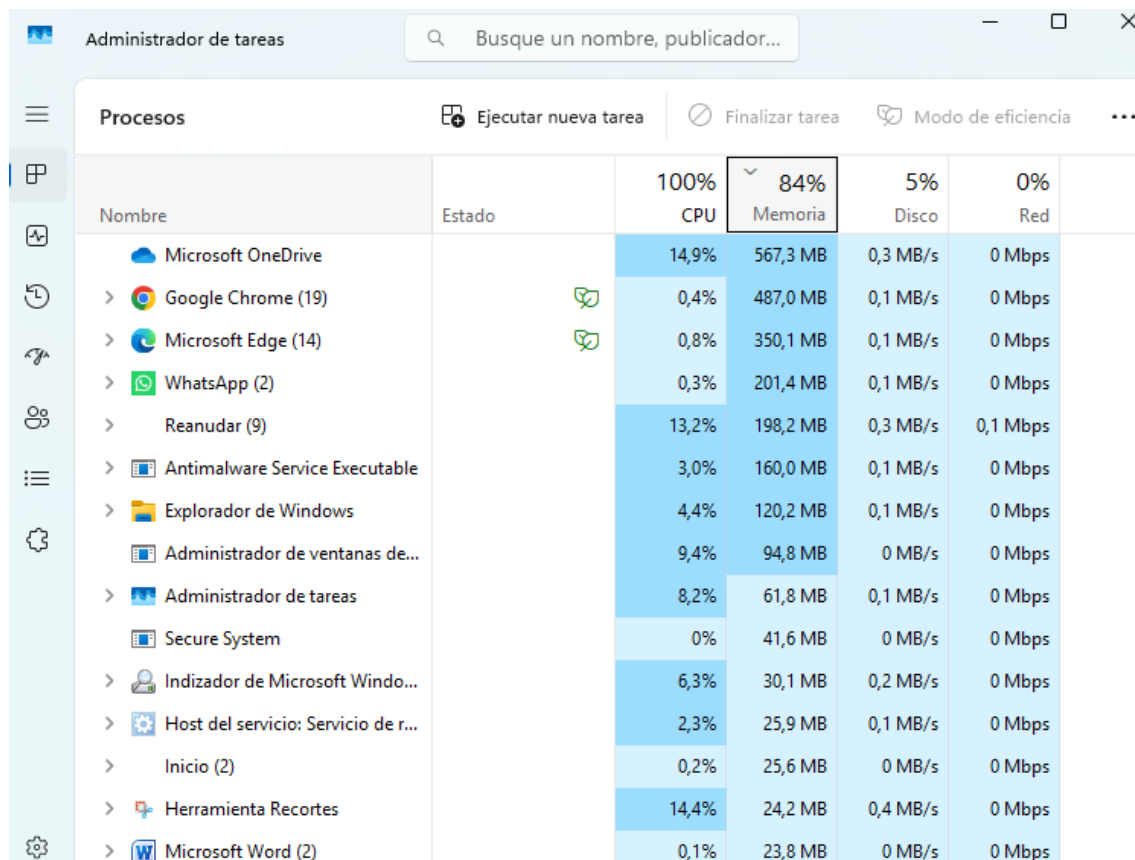


PROCESO FINAL S.O

LABORATORIO 1

Estados de Procesos

- Utilizar el Administrador de Tareas (Windows) o top/htop (Linux)



The screenshot shows the Windows Task Manager interface. The 'Procesos' tab is selected. The top bar indicates system resource usage: CPU at 100%, Memory at 84%, Disk at 5%, and Network at 0%. The main table lists running processes with columns for Name, State, CPU, Memory, Disk, and Network. The following table represents the data shown in the screenshot:

Nombre	Estado	100% CPU	84% Memoria	5% Disco	0% Red
Microsoft OneDrive		14,9%	567,3 MB	0,3 MB/s	0 Mbps
> Google Chrome (19)		0,4%	487,0 MB	0,1 MB/s	0 Mbps
> Microsoft Edge (14)		0,8%	350,1 MB	0,1 MB/s	0 Mbps
> WhatsApp (2)		0,3%	201,4 MB	0,1 MB/s	0 Mbps
> Reanudar (9)		13,2%	198,2 MB	0,3 MB/s	0,1 Mbps
> Antimalware Service Executable		3,0%	160,0 MB	0,1 MB/s	0 Mbps
> Explorador de Windows		4,4%	120,2 MB	0,1 MB/s	0 Mbps
> Administrador de ventanas de...		9,4%	94,8 MB	0 MB/s	0 Mbps
> Administrador de tareas		8,2%	61,8 MB	0,1 MB/s	0 Mbps
> Secure System		0%	41,6 MB	0 MB/s	0 Mbps
> Indizador de Microsoft Windo...		6,3%	30,1 MB	0,2 MB/s	0 Mbps
> Host del servicio: Servicio de r...		2,3%	25,9 MB	0,1 MB/s	0 Mbps
> Inicio (2)		0,2%	25,6 MB	0 MB/s	0 Mbps
> Herramienta Recortes		14,4%	24,2 MB	0,4 MB/s	0 Mbps
> Microsoft Word (2)		0,1%	23,8 MB	0 MB/s	0 Mbps

Esta captura representa una máquina con un desempeño muy limitado. Su procesador trabaja al 100%, y la memoria RAM está casi agotada. Entre los procesos que más recursos consumen destacan OneDrive, Google Chrome, Microsoft Edge y la herramienta Recortes. Bajo este escenario, es muy probable que el sistema se perciba lento y que las respuestas sean poco ágiles.

Además, para comprender mejor las etapas por las que pasa un proceso, elaboramos un programa sencillo en Python (versión 13.3.5) que transita por distintos estados del ciclo de vida de un proceso. A continuación se incluye el código utilizado:

```

import time

import threading

def proceso():

    print("Estado: Nuevo")

    time.sleep(1) # Simula el tiempo de creación del proceso

    print("Estado: Listo")

    time.sleep(1) # Simula que el proceso espera su turno para ejecutarse

    print("Estado: Ejecutando")

    # Simula carga intensa en la CPU

    for i in range(10**7):

        pass

    print("Estado: Bloqueado")

    time.sleep(3) # Simula que el proceso espera algún recurso externo

    print("Estado: Terminado")

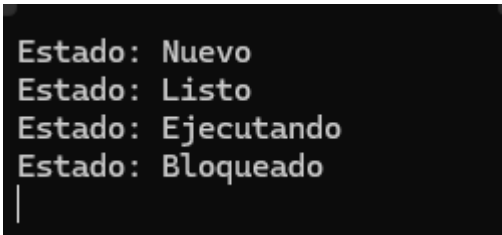
if __name__ == "__main__":

    proceso()

```

En este script, el programa imprime cada estado conforme avanza por las etapas que normalmente atravesaría un proceso en un sistema operativo. Los tiempos de espera (`time.sleep()`) simulan periodos de inactividad o espera por recursos externos, mientras que el bucle `for` representa trabajo pesado que consume CPU, como si el proceso estuviera realmente en ejecución.

- Documentar con capturas cada estado: Nuevo



```

Estado: Nuevo
Estado: Listo
Estado: Ejecutando
Estado: Bloqueado
|

```

- Listo
- Ejecutando
- Bloqueado
- Terminado

La simulación de los distintos estados del proceso se llevó a cabo en el Administrador de Tareas de Windows, tal como se describió previamente en el código desarrollado en Python (versión 13.3.5). Este programa reproduce las siguientes etapas del ciclo de vida del proceso:

- **Nuevo:** Representa el momento en que el proceso es creado por el sistema operativo.
- **Listo:** Indica que el proceso está preparado y esperando a que el planificador le asigne tiempo de CPU.
- **Ejecutando:** Simula la carga intensiva del procesador mientras el proceso está en plena ejecución.
- **Bloqueado:** Refleja una espera debido a que el proceso depende de algún recurso externo antes de continuar.
- **Terminado:** Marca la finalización del proceso y su liberación de recursos.

```
Estado: Nuevo
Estado: Listo
Transición 'Nuevo → Listo': 1.0025 segundos
Estado: Ejecutando
Transición 'Listo → Ejecutando': 1.0279 segundos
Estado: Bloqueado
Transición 'Ejecutando → Bloqueado': 0.7026 segundos
```

Medición de los tiempos de transición entre estados

Medir los tiempos de transición entre estados consiste en determinar cuánto tiempo le toma a un proceso pasar de un estado a otro; por ejemplo, de *Nuevo* a *Listo*, o de *Ejecutando* a *Bloqueado*. Para ello, ejecuté el programa en Python desde el CMD de Windows y registré el momento exacto antes y después de cada cambio de estado. Al restar ambos tiempos, obtuve la duración de cada transición. Este método permitió comprender mejor el comportamiento temporal del proceso a lo largo de su ciclo de vida.

Scheduling del sistema operativo

En esta parte del laboratorio ejecuté simultáneamente 5 programas intensivos en CPU para analizar cómo el planificador del sistema operativo reparte los recursos entre ellos. Al monitorizar el uso del procesador en el Administrador de Tareas, observé que Windows alterna rápidamente entre los procesos activos, manteniendo un uso equilibrado del CPU y asegurando que cada tarea recibiera una porción de tiempo de procesamiento.

Procesos		100% CPU	87% Memoria	4% Disco	0% Red
Nombre	Estado				
> Python 3.13 (5)		61,8%	21,8 MB	0 MB/s	0 Mbps
Administrador de ventanas de...		12,3%	99,6 MB	0,1 MB/s	0 Mbps
> Herramienta Recortes (2)		9,0%	66,0 MB	0,1 MB/s	0 Mbps
Administrador de tareas		4,4%	56,9 MB	0 MB/s	0 Mbps
> Explorador de Windows		4,0%	120,6 MB	0 MB/s	0 Mbps
System		1,7%	0,1 MB	0,1 MB/s	0 Mbps
> Microsoft Edge (18)		1,6%	243,6 MB	0,1 MB/s	0 Mbps
> Host del servicio: Servicio Ad...		1,1%	3,6 MB	0,1 MB/s	0 Mbps
> Host del servicio: Servicio de r...		0,7%	20,9 MB	0 MB/s	0 Mbps
Interrupciones del sistema		0,6%	0 MB	0 MB/s	0 Mbps
> Microsoft Office 2010		0,5%	6,7 MB	0 MB/s	0 Mbps
Microsoft OneDrive		0,5%	70,6 MB	0 MB/s	0 Mbps
> Terminal (13)		0,4%	128,1 MB	0 MB/s	0 Mbps
Antimalware Service Executable		0,4%	125,0 MB	0,1 MB/s	0 Mbps
> Host del servicio: Servicio de u...		0,3%	9,4 MB	0 MB/s	0 Mbps

Ejecución de programas intensivos y distribución del CPU

Al lanzar 5 programas que consumen muchos recursos al mismo tiempo, se pudo comprobar que el procesador llega al 100% de su capacidad. Sin embargo, el sistema operativo reparte el tiempo de CPU entre todos los procesos activos de forma eficiente. Así, en lugar de ejecutar las tareas una por una como lo haría un planificador tipo FIFO, Windows asigna pequeños intervalos de tiempo a cada programa, logrando que todos progresen de manera simultánea. Esta estrategia —muy similar al algoritmo **Round Robin**— permite que la computadora mantenga su capacidad de respuesta y funcione de forma fluida, incluso cuando está bajo una carga intensa.

Comparación con algoritmos teóricos

Al contrastar este comportamiento con los algoritmos clásicos, como FIFO (Primero en Entrar, Primero en Salir), queda claro que Windows no procesa una tarea hasta completarla antes de pasar a la siguiente. Por el contrario, utiliza una política de planificación por turnos cortos que imita el esquema Round Robin. Gracias a este método, el sistema puede gestionar múltiples procesos pesados a la vez, proporcionando a cada uno un tiempo equitativo del procesador y evitando que una sola tarea monopolice la CPU. Esto da la sensación de que las tareas se ejecutan en paralelo, mejorando el rendimiento global y la experiencia del usuario.

Creación de Deadlock

Para comprender qué es un deadlock, investigamos un escenario simple en el que dos o más procesos (o hilos) quedan bloqueados de manera permanente esperando recursos que ya están retenidos por otros procesos del mismo grupo. Esta situación, también llamada **interbloqueo**, impide que cualquiera de ellos pueda continuar su ejecución, ya que cada proceso depende de un recurso que nunca se libera.

Comportamiento del sistema operativo

Al provocar intencionalmente un deadlock en el entorno de Windows, observamos que ambos procesos permanecen en estado “No responde” y el consumo de CPU es prácticamente nulo, ya que están esperando indefinidamente. Como resultado, el sistema operativo mantiene estos procesos en memoria sin asignarles tiempo de procesador, hasta que se libere alguno de los recursos involucrados. Sin embargo, sin una intervención externa, el deadlock persiste y no se resuelve automáticamente.

Nombre	Estado	100% CPU	94% Memoria	7% Disco	0% Red
> Bloc de notas		0%	18,9 MB	0 MB/s	0 Mbps
Python		0%	4,9 MB	0 MB/s	0 Mbps

Se puede observar que tanto Bloc de notas como Python muestran un uso del 0% de CPU, lo que indica que en ese momento no están ejecutando procesos que demanden ciclos de procesamiento.

Intentar resolver el deadlock

```
Iniciando hilos para simular deadlock...
Proceso A: Intentando adquirir Lock1...
Proceso A: Lock1 adquirido. Esperando un momento...
Proceso B: Intentando adquirir Lock2...
Proceso B: Lock2 adquirido. Esperando un momento...
Proceso A: Intentando adquirir Lock2...
Proceso B: Intentando adquirir Lock1...

Verificando el estado de los hilos después de un tiempo.
Proceso A sigue activo (posiblemente bloqueado).
Proceso B sigue activo (posiblemente bloqueado).
Fin del programa principal.
```

Se puede observar un interbloqueo (deadlock) que involucra dos hilos, **Proceso A** y **Proceso B**, junto con dos recursos compartidos denominados **Lock1** y **Lock2**. Estos "locks" actúan como permisos o mecanismos exclusivos que un hilo debe adquirir para poder acceder a un recurso o a una sección específica del código. La regla principal es que solo un hilo puede tener cada lock en su poder a la vez.