

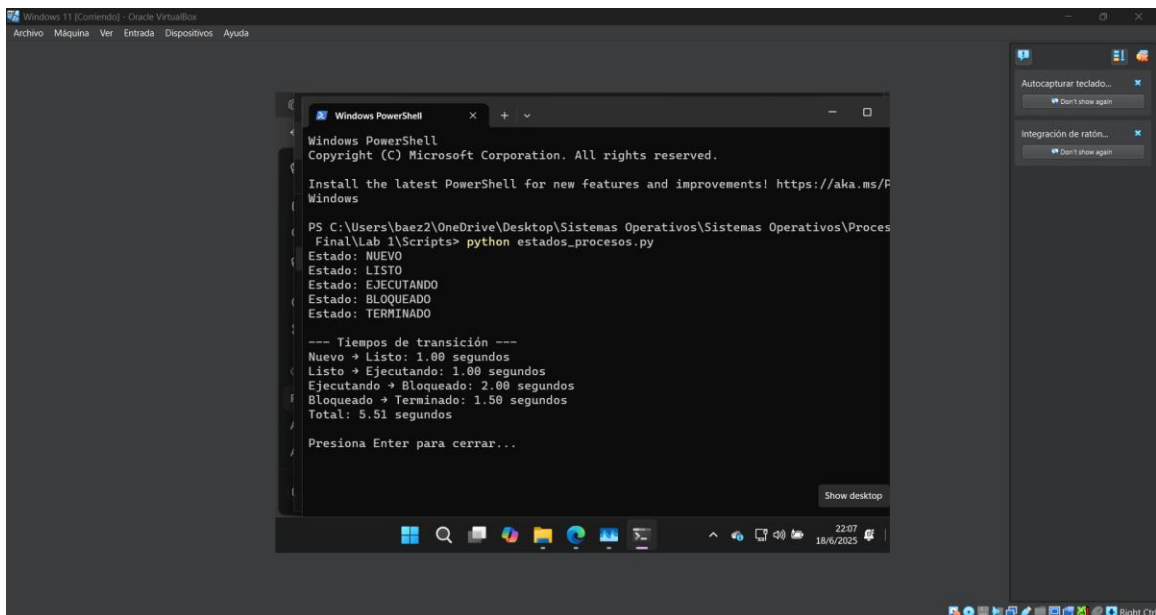
# INFORME LABORATORIO 1 - GESTIÓN DE PROCESOS

En este laboratorio trabajamos con distintos aspectos del manejo de procesos en un sistema operativo, utilizando una máquina virtual con Windows como entorno controlado. El objetivo fue entender cómo se comportan los procesos desde que se crean hasta que terminan, cómo se distribuye el uso del CPU entre ellos, y cómo se puede producir un bloqueo mutuo (deadlock) si no se gestionan bien los recursos.

## 1. Estados de los procesos:

Para esta parte del laboratorio creé un pequeño script en Python que simula el paso de un proceso por los estados clásicos: Nuevo, Listo, Ejecutando, Bloqueado y Terminado. En cada uno de estos estados se registró el tiempo exacto en que entraba, y luego se midió cuánto tiempo permanecía en cada uno.

Los datos obtenidos se colocaron en una tabla de Excel para facilitar el análisis. También se generó un gráfico de barras para visualizar cuánto duró cada estado.



```
Windows 11 [Comando] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PowerShellLatest

PS C:\Users\baez2\OneDrive\Desktop\Sistemas Operativos\Sistemas Operativos\Procesos> python estados_procesos.py
Estado: NUEVO
Estado: LISTO
Estado: EJECUTANDO
Estado: BLOQUEADO
Estado: TERMINADO

--- Tiempos de transición ---
Nuevo + Listo: 1.00 segundos
Listo + Ejecutando: 1.00 segundos
Ejecutando + Bloqueado: 2.00 segundos
Bloqueado + Terminado: 1.50 segundos
Total: 5.51 segundos

Presiona Enter para cerrar...
```

### Script utilizado:

```
import time

def simular_proceso():
    inicio = time.time()
    print("Estado: NUEVO")
    time.sleep(1)

    tiempo_ready = time.time()
    print("Estado: LISTO")
    time.sleep(1)

    tiempo_running = time.time()
    print("Estado: EJECUTANDO")
    time.sleep(2)

    tiempo_bloqueado = time.time()
    print("Estado: BLOQUEADO")
    time.sleep(1.5)

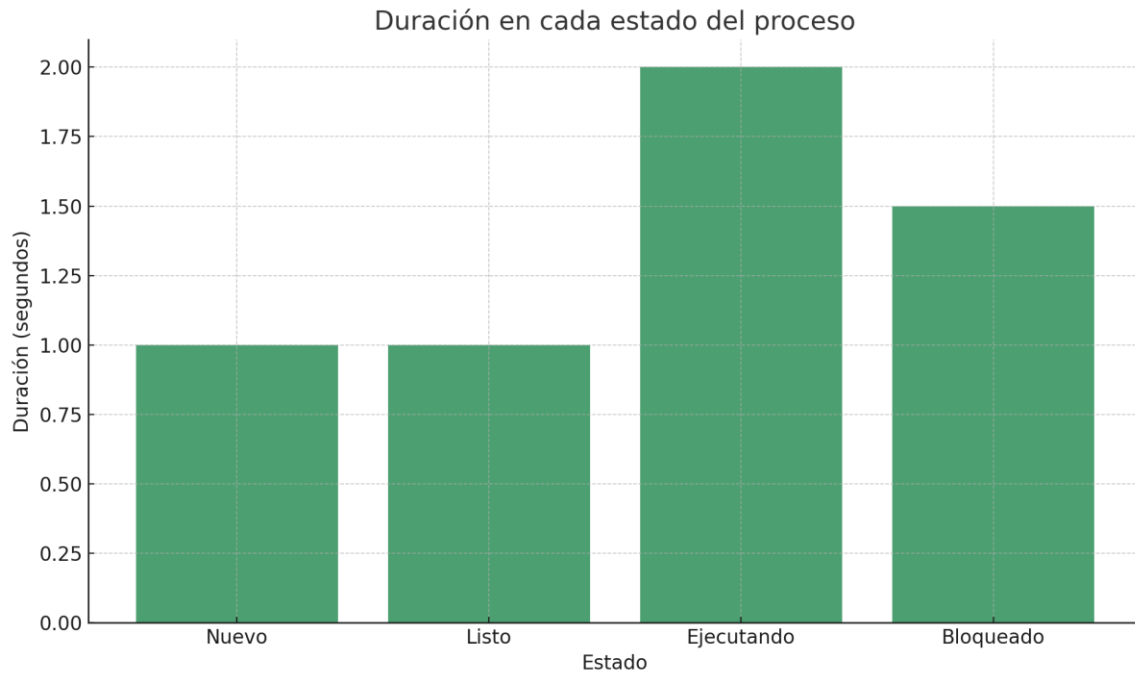
    tiempo_terminado = time.time()
    print("Estado: TERMINADO")

    # Medición de tiempos
    transiciones = {
        "Nuevo → Listo": tiempo_ready - inicio,
        "Listo → Ejecutando": tiempo_running - tiempo_ready,
        "Ejecutando → Bloqueado": tiempo_bloqueado - tiempo_running,
        "Bloqueado → Terminado": tiempo_terminado - tiempo_bloqueado,
        "Total": tiempo_terminado - inicio
    }

    print("\n--- Tiempos de transición ---")
    for transicion, duracion in transiciones.items():
        print(f"{transicion}: {duracion:.2f} segundos")

simular_proceso()
input("\nPresiona Enter para cerrar...")
```

## Gráfico con las mediciones:



## 2. Simulación de un deadlock:

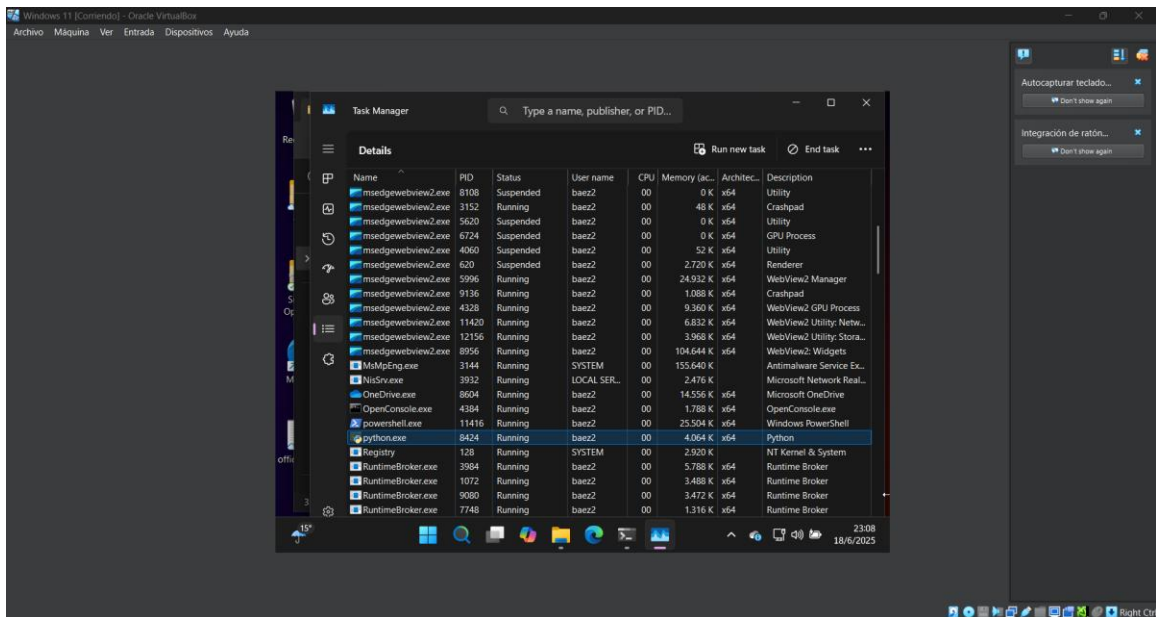
Para simular un deadlock, utilicé otro script en Python con dos hilos que intentan acceder a recursos en orden cruzado. Esto hace que ambos queden esperando uno al otro y ninguno pueda avanzar. El programa se queda congelado sin finalizar, lo cual es justamente el comportamiento esperado en un bloqueo mutuo.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\baez2\OneDrive\Desktop\Sistemas Operativos\Sistemas Operativos\Proceso Final\Lab 1\Scripts> python deadlock_simulado.py
Hilo 1: intentando acceder al recurso A
Hilo 1: accedió al recurso A
Hilo 2: intentando acceder al recurso B
Hilo 2: accedió al recurso B
Hilo 1: intentando acceder al recurso B
Hilo 2: intentando acceder al recurso A
```

The screenshot shows a Windows 11 desktop environment. In the center is a Windows PowerShell terminal window. The terminal displays the output of a Python script named 'deadlock\_simulado.py'. The script simulates two threads, Hilo 1 and Hilo 2, attempting to acquire two resources, A and B, in a cross-order manner. Hilo 1 acquires resource A, and Hilo 2 acquires resource B. Then, Hilo 1 attempts to acquire resource B, and Hilo 2 attempts to acquire resource A. Since each thread is holding a resource the other needs, they both enter a waiting state, resulting in a deadlock. The terminal output shows the sequence of events up to the point where both threads are waiting, but the script has not yet finished, illustrating the 'congelado' (frozen) state.



## Script Utilizado:

```
import threading
import time

recurso_A = threading.Lock()
recurso_B = threading.Lock()

def hilo_1():
    print("Hilo 1: intentando acceder al recurso A")
    recurso_A.acquire()
    print("Hilo 1: accedió al recurso A")
    time.sleep(1)
    print("Hilo 1: intentando acceder al recurso B")
    recurso_B.acquire()
    print("Hilo 1: accedió al recurso B")
    recurso_B.release()
    recurso_A.release()

def hilo_2():
    print("Hilo 2: intentando acceder al recurso B")
    recurso_B.acquire()
    print("Hilo 2: accedió al recurso B")
    time.sleep(1)
    print("Hilo 2: intentando acceder al recurso A")
    recurso_A.acquire()
    print("Hilo 2: accedió al recurso A")
    recurso_A.release()
    recurso_B.release()

t1 = threading.Thread(target=hilo_1)
t2 = threading.Thread(target=hilo_2)

t1.start()
t2.start()

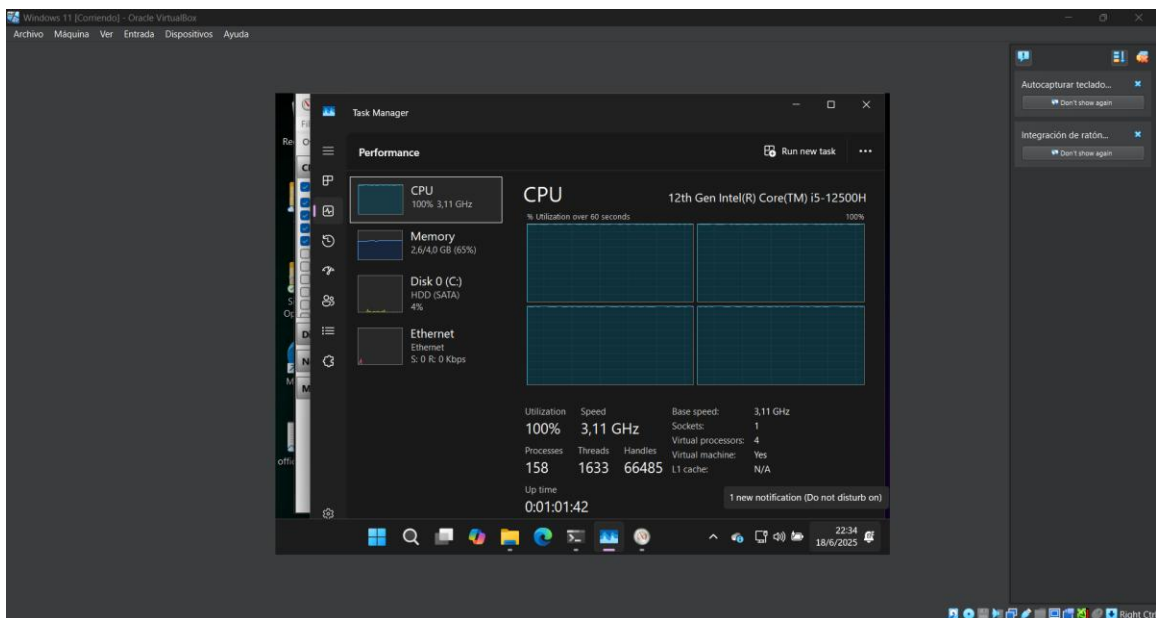
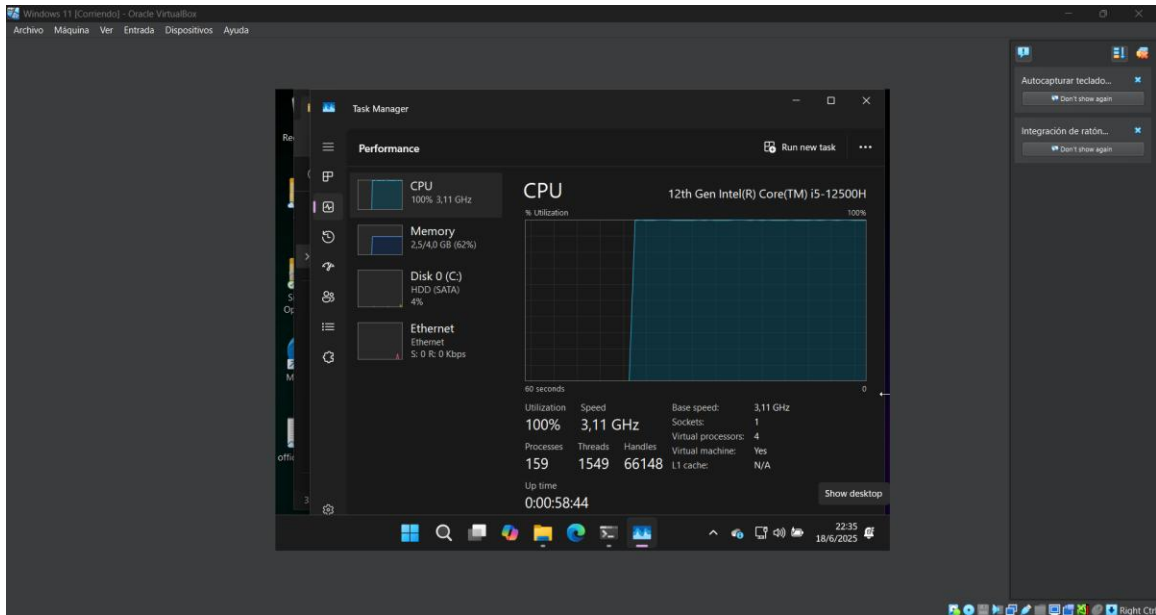
t1.join()
t2.join()

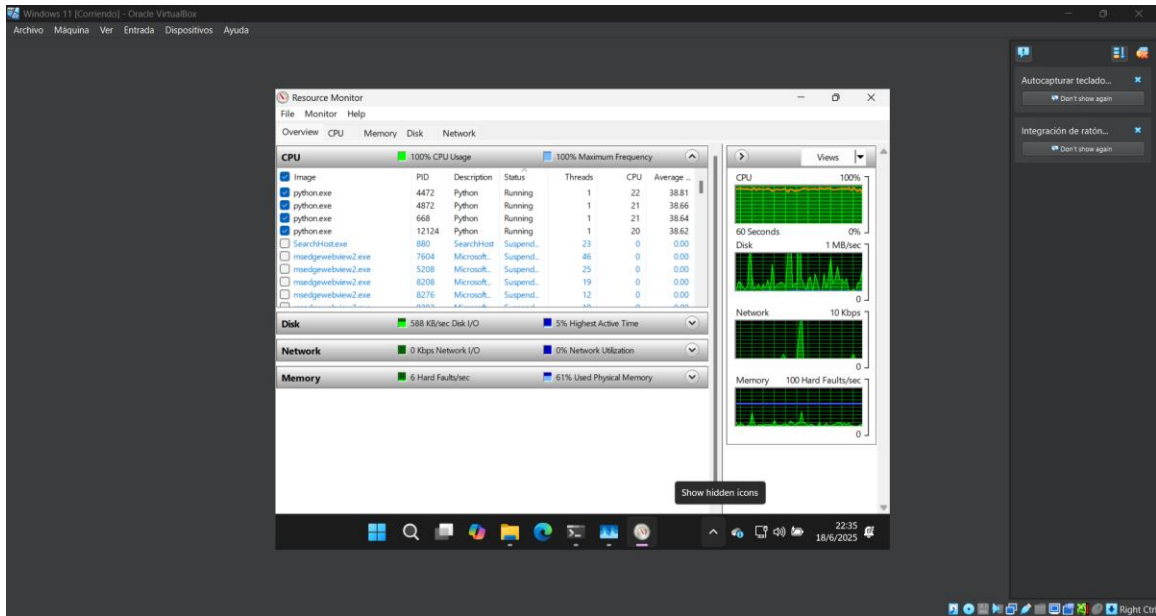
input("Presiona Enter para cerrar...")
```

### 3. Planificación (Scheduling) del sistema operativo:

En esta parte ejecuté varios scripts en paralelo (cuatro, en mi caso, porque la VM no me permitió más), y observé cómo el sistema operativo distribuía el uso del CPU. Todos los procesos estaban en ejecución al mismo tiempo, usando aproximadamente el mismo porcentaje de CPU.

Esto muestra que el sistema utiliza un esquema parecido al algoritmo Round Robin, donde se reparte el CPU entre todos los procesos activos.





### Script Utilizado:

```
import math

while True:
    math.factorial(1000)
```

Cuando corrí los programas pesados al mismo tiempo, noté que el sistema no dejaba que uno solo se lleve toda la CPU. Todos los procesos se mantenían activos y usaban casi la misma cantidad de CPU, lo que me hizo pensar que el sistema va “repartiendo” el tiempo entre ellos, como si les diera turnos cortos para trabajar.

Por eso, me pareció que el comportamiento se parece mucho al algoritmo Round Robin que vimos en clase, donde cada proceso tiene su momento para usar el procesador y después le toca a otro. No fue como FIFO, porque no esperaban uno a que termine el otro para empezar.

## Conclusión

Este laboratorio me ayudó a entender mejor cómo funcionan los procesos dentro del sistema operativo. Pude ver cómo cambian de estado, cómo pueden bloquearse entre sí, y cómo el sistema reparte el uso del CPU de forma justa entre varios procesos al mismo tiempo. Fue una buena manera de aplicar lo que vimos en clase de forma práctica y dentro de un entorno seguro como la máquina virtual.

