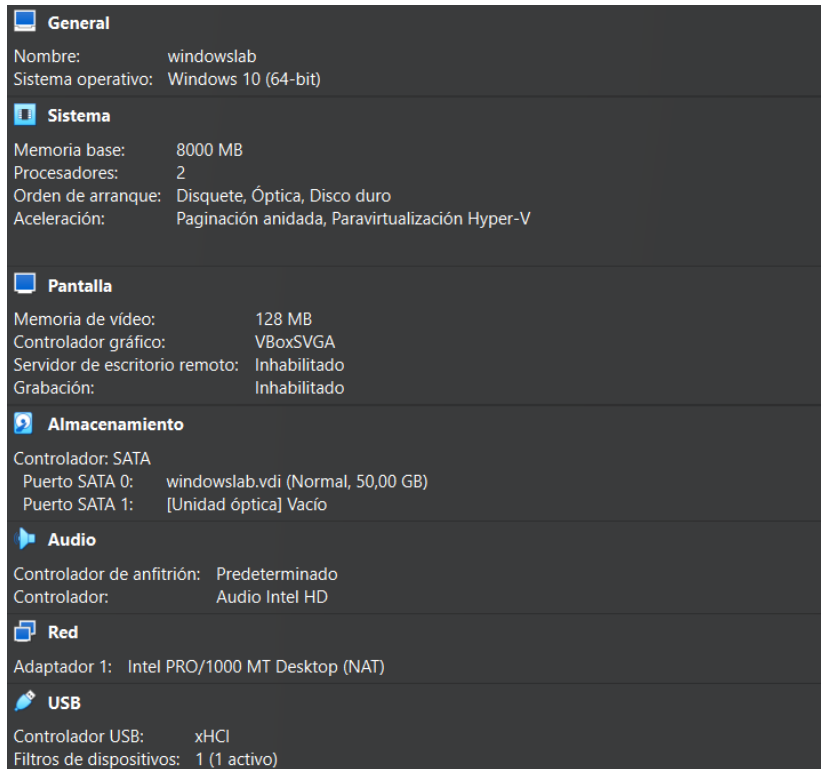


Laboratorio 1 : Gestion de procesos

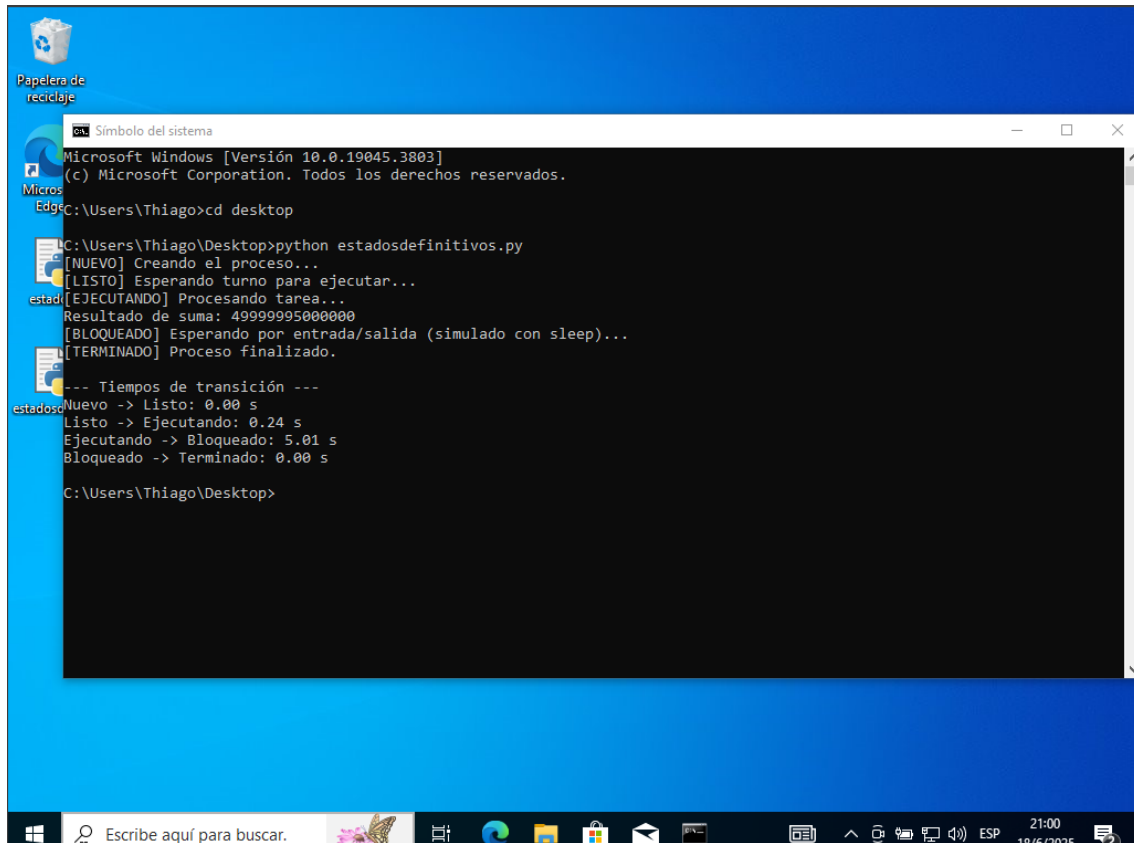
Para tener en cuenta las pruebas fueron realizadas con un computadora, con las siguientes características:



Estado

de

procesos



```
Microsoft Windows [Versión 10.0.19045.3803]
(c) Microsoft Corporation. Todos los derechos reservados.

EdgC:\Users\Thiago>cd desktop

C:\Users\Thiago\Desktop>python estadosdefinitivos.py
[NUEVO] Creando el proceso...
[LISTO] Esperando turno para ejecutar...
[EJECUTANDO] Procesando tarea...
Resultado de suma: 49999995000000
[BLOQUEADO] Esperando por entrada/salida (simulado con sleep)...
[TERMINADO] Proceso finalizado.

--- Tiempos de transición ---
Nuevo -> Listo: 0.00 s
Listo -> Ejecutando: 0.24 s
Ejecutando -> Bloqueado: 5.01 s
Bloqueado -> Terminado: 0.00 s

C:\Users\Thiago\Desktop>
```

Para definir el estados de procesos, utilice una aplicación hecha en Python, ejecutada directamente desde el cmd.

El comando Cd Desktop, direcciona hacia el escritorio donde tenia guardado el archivo.

Estados de un Proceso

1. **Nuevo (New)** → El proceso se acaba de crear.
2. **Listo (Ready)** → Está en espera para ser ejecutado por la CPU.
3. **Ejecutando (Running)** → El proceso está siendo procesado por la CPU.
4. **Bloqueado (Blocked/Waiting)** → El proceso está esperando un recurso (E/S, señal, etc.).
5. **Terminado (Terminated)** → El proceso finalizó su ejecución.

Nuevo: Se crea el proceso

Listo: Esperando su turno para ejecutar

Ejecutando: Procesando Tarea

Bloqueado: Esperando por entrada-salida

Terminado: Proceso Finalizado

De nuevo a listo, tardo 0.00 segundos, de listo a ejecutando tardo 0.24segundos, de ejecutando a bloqueado tardo 5.01 segundos. Por ultimo de bloqueado a terminado tardo 0.00 segundos

```
import time
```

```
import os
```

```
def estado_nuevo():
```

```
    print("[NUEVO] Creando el proceso...")
```

```
def estado_listo():
```

```
    print("[LISTO] Esperando turno para ejecutar...")
```

```
def estado_ejecutando():
```

```
    print("[EJECUTANDO] Procesando tarea...")
```

```
    suma = 0
```

```
    for i in range(10_000_000):
```

```
        suma += i
```

```
    print(f"Resultado de suma: {suma}")
```

```
def estado_bloqueado():
```

```
    print("[BLOQUEADO] Esperando por entrada/salida (simulado con sleep)...")
```

```
    time.sleep(5)
```

```
def estado_terminado():
```

```
    print("[TERMINADO] Proceso finalizado.")
```

```
# Registro de tiempos
```

```
inicio = time.time()
```

```
estado_nuevo()
```

```
nuevo_time = time.time()
```

```
estado_listo()
```

```
listo_time = time.time()
```

```
estado_ejecutando()
```

```
ejecutando_time = time.time()
```

```
estado_bloqueado()
```

```
bloqueado_time = time.time()
```

```
estado_terminado()
```

```
terminado_time = time.time()
```

```
# Tiempos
```

```
print("\n--- Tiempos de transición ---")
```

```
print(f"Nuevo -> Listo: {listo_time - nuevo_time:.2f} s")
```

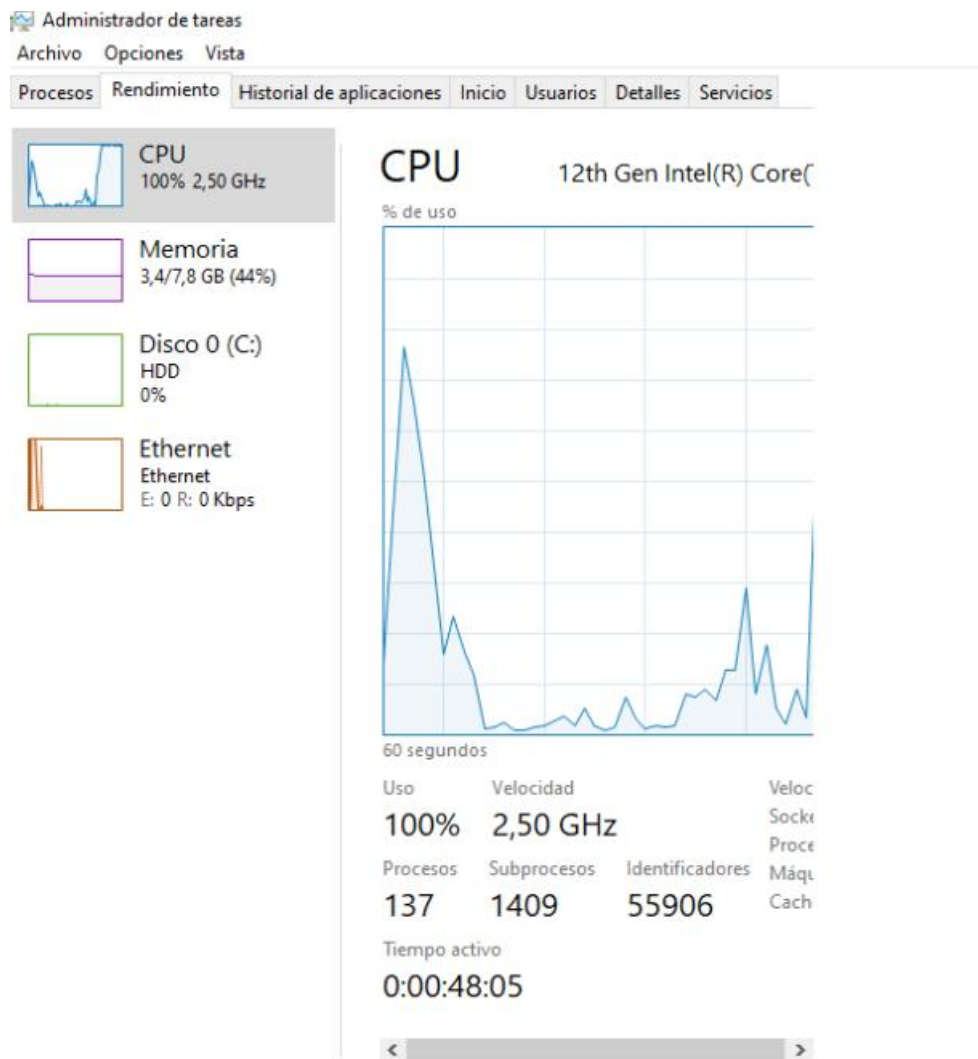
```
print(f"Listo -> Ejecutando: {ejecutando_time - listo_time:.2f} s")
```

```
print(f"Ejecutando -> Bloqueado: {bloqueado_time - ejecutando_time:.2f} s")
```

```
print(f"Bloqueado -> Terminado: {terminado_time - bloqueado_time:.2f} s")
```

Este código en Python simula los diferentes estados de un proceso en un sistema operativo: **Nuevo** (creación), **Listo** (espera para ejecutar), **Ejecutando** (realiza una tarea intensiva de CPU sumando números), **Bloqueado** (espera simulada con `time.sleep(5)` como si hiciera E/S) y **Terminado** (finalización). Mide y muestra los tiempos entre cada transición de estado, donde:

Scheduling del sistema operativo



Administrador de tareas

Archivo Opciones Vista

Procesos Rendimiento Historial de aplicaciones Inicio Usuarios Detalles Servicios

| Nombre | Estado | 100% CPU | 43% Memoria | 0% Disco | 0% Red | Consumo de ... | Tendencia de ... |
|---------------------------|--------|----------|-------------|----------|----------|----------------|------------------|
| Aplicaciones (4) | | | | | | | |
| > Administrador de tareas | | 0,3% | 17,3 MB | 0 MB/s | 0 Mbps | Muy baja | Muy baja |
| > Microsoft Edge (8) | | 0,4% | 378,0 MB | 0 MB/s | 0,1 Mbps | Muy baja | Muy baja |
| > Python (32 bits) (3) | | 32,1% | 10,7 MB | 0 MB/s | 0 Mbps | Alto | Moderado |
| > Python (32 bits) (3) | | 17,1% | 10,7 MB | 0 MB/s | 0 Mbps | Moderado | Bajo |

1. Procesos Python consumiendo ~32% y ~17% de CPU:

- Indica que **no están usando toda la CPU** (solo fracciones de un núcleo).
- Windows está **repartiendo el tiempo de CPU** entre tus procesos y otros (como Edge, el Administrador de tareas, etc.).
- El **100% de uso** en cpualmax.png sugiere que **al menos un núcleo está saturado** (probablemente por tu bucle de sumas en Python).

2. Varios procesos en ejecución (137 procesos, 1409 hilos):

- Windows maneja **multi-tarea real**, donde el scheduler (planificador) decide cuándo cada proceso corre.
- Tus scripts de Python **compiten** con otros procesos por tiempo de CPU.

Comparación con Algoritmos Teóricos

1. FIFO (First-In, First-Out)

- **Cómo funciona:**
 - El primer proceso que llega **ocupa la CPU hasta terminar**.
 - Si un proceso se bloquea (espera E/S), la CPU pasa al siguiente.
- **Comparación:**
 - **No es lo que ocurre en tus imágenes**, porque Windows no deja que un solo proceso (como Python) acapare la CPU.
 - Si fuera FIFO, **uno de tus procesos Python estaría al 100%** y los demás esperarían.
 - **En tu simulación actual**, tu código se parece a FIFO porque avanza secuencialmente.

2. Round Robin (RR)

- **Cómo funciona:**
 - Cada proceso recibe un **quantum** (ej. 10-100ms) de CPU antes de ser pausado y puesto al final de la cola.
 - **Justo y equilibrado**, pero puede haber mucho *context switching* (cambio entre procesos).
- **Comparación:**
 - **Windows actúa más como Round Robin** (con prioridades).
 - Tus procesos Python **no llegan al 100%** porque el SO les da pequeños tiempos de CPU y luego los cambia por otros.
 - El **Administrador de Tareas** muestra cómo varios procesos comparten la CPU (como Edge con 0.4%, Python con 32%, etc.).

Creacion de Deadlok

```
Microsoft Windows [Versión 10.0.19045.3803]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Thiago>cd desktop

C:\Users\Thiago\Desktop>python deadlok.py
Thread 1: intentando adquirir Lock 1
Thread 1: adquirió Lock 1
Thread 2: intentando adquirir Lock 2
Thread 2: adquirió Lock 2
Thread 2: intentando adquirir Lock 1
Thread 1: intentando adquirir Lock 2
```

Procesador de comandos de Wi... 0% 12,0 MB 0 MB/s 0 Mbps Muy baja Muy baja

Durante la ejecución deadlock, se utilizó nuevamente un programa hecho en Python, Este código demuestra claramente un **deadlock clásico** debido a la adquisición cruzada de locks. Vamos a desglosarlo:

Cómo se Produce el Deadlock

1. **Thread 1:**
 - Adquiere lock1 → Espera 1s → Intenta adquirir lock2
2. **Thread 2:**
 - Adquiere lock2 → Espera 1s → Intenta adquirir lock1
3. **Resultado:**
 - Thread 1 tiene lock1 y espera por lock2 (que tiene Thread 2).
 - Thread 2 tiene lock2 y espera por lock1 (que tiene Thread 1).
 - **Ambos se bloquean indefinidamente**

Conclusión

- **Error clave:** Adquisición cruzada de locks ($\text{lock1} \rightarrow \text{lock2}$ vs $\text{lock2} \rightarrow \text{lock1}$).
- **Solución preferida: Orden consistente** (primero lock1, luego lock2 en todos los threads).
- **Alternativas:** Timeouts o RLock para casos complejos

Conclusion General del Laboratorio

Este laboratorio demostró que, mientras en la simulación los procesos avanzan secuencialmente (como en FIFO), en la realidad los sistemas operativos usan algoritmos como Round Robin para distribuir el tiempo de CPU entre múltiples procesos de manera más eficiente, como se observó en el Administrador de Tareas donde los procesos Python compartían la CPU; además, se evidenció que los deadlocks -bloqueos mutuos causados por una mala gestión de recursos compartidos, como se vio en el código con los locks cruzados- son errores de programación que deben prevenirse mediante órdenes consistentes de adquisición de recursos o mecanismos de timeout, destacando así la importancia de diseñar sistemas concurrentes con cuidado para evitar bloqueos y maximizar la eficiencia del sistema operativo.