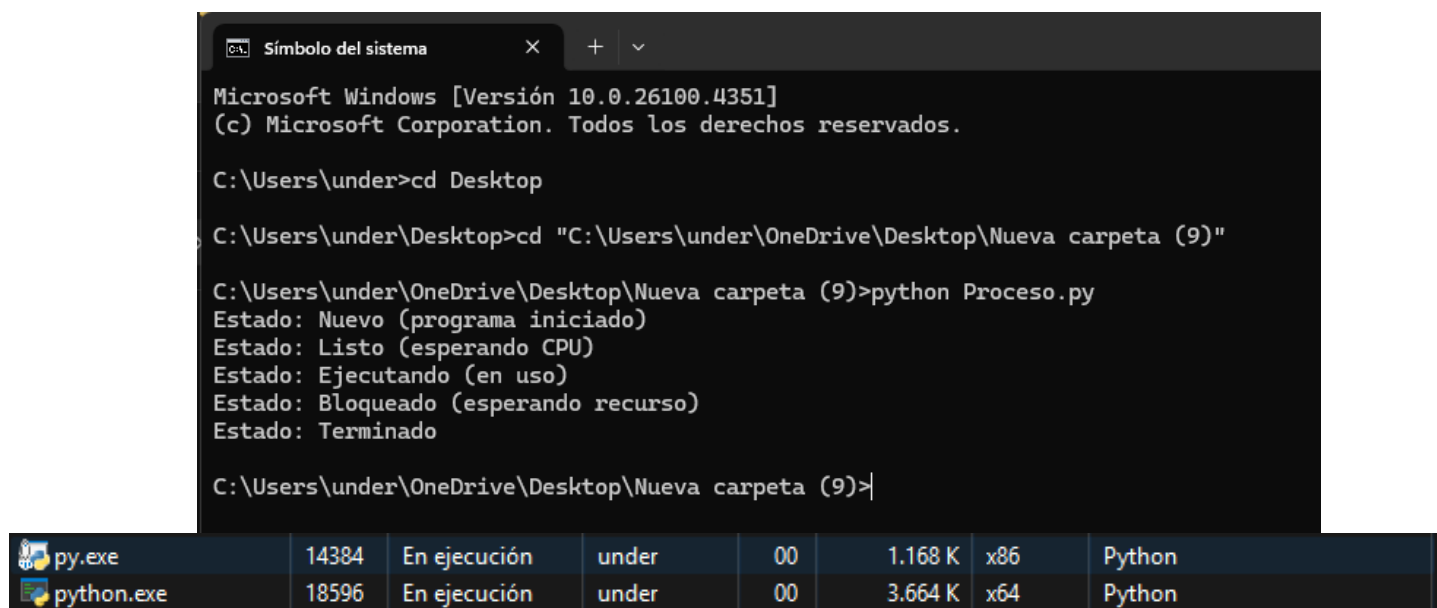


Laboratorio 1 - Gestión de Procesos

Este informe documenta los resultados del laboratorio 1, enfocado en la Gestión de Procesos.

1. Simulación de Estados de Proceso:

Se creó un script en Python llamado 'proceso.py' que simula los estados: Nuevo, Listo, Ejecutando, Bloqueado y Terminado. Se midieron los tiempos de transición entre estados y se tomo captura del mismo junto como se veía en el administrador de tareas también



The image shows a Windows command prompt window titled 'Símbolo del sistema' with the following text:

```
Microsoft Windows [Versión 10.0.26100.4351]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\user>cd Desktop

C:\Users\user\Desktop>cd "C:\Users\user\OneDrive\Desktop\Nueva carpeta (9)"

C:\Users\user\OneDrive\Desktop\Nueva carpeta (9)>python Proceso.py
Estado: Nuevo (programa iniciado)
Estado: Listo (esperando CPU)
Estado: Ejecutando (en uso)
Estado: Bloqueado (esperando recurso)
Estado: Terminado

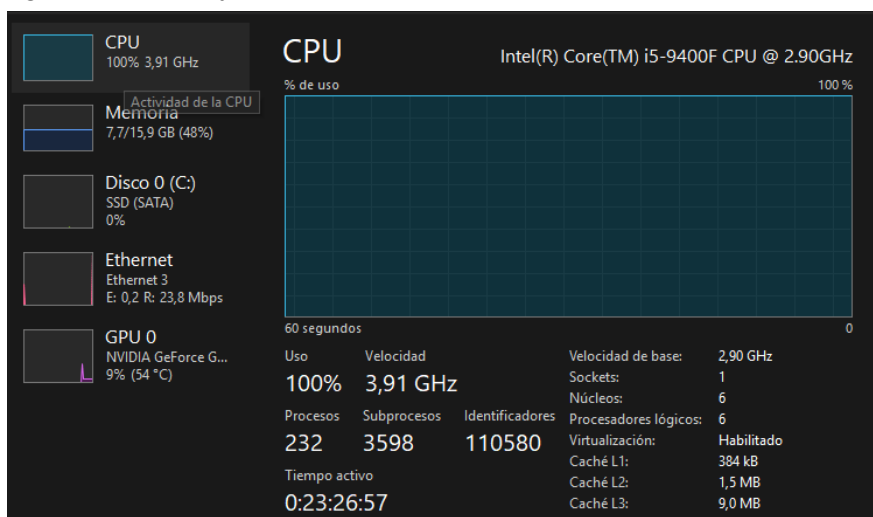
C:\Users\user\OneDrive\Desktop\Nueva carpeta (9)>|
```

Below the command prompt is a screenshot of the Windows Task Manager 'Processes' tab. It shows two processes:

Process Name	PID	Status	User	Session ID	Private Bytes	Architecture	Working Set
py.exe	14384	En ejecución	user	00	1.168 K	x86	Python
python.exe	18596	En ejecución	user	00	3.664 K	x64	Python






2. Scheduling de CPU:

Se ejecutaron múltiples scripts 'cpu_loader.py' para simular alta carga en el CPU. Se observó cómo el sistema operativo distribuye el uso del CPU entre los procesos en ejecución. Las observaciones se compararon con los algoritmos FIFO y Round Robin.



3. Comparación con algoritmos teóricos de planificación:

Durante la ejecución simultánea de cinco scripts que consumen CPU (cpu_loader.py), se observó en el Administrador de tareas que el uso del CPU se reparte casi equitativamente entre los cinco procesos. Esto se refleja en la captura adjunta, donde cada proceso de Python consume aproximadamente el 19-20% del CPU.

Nombre	Estado	100% CPU
 Python		19,5%
 Python		19,5%
 Python		19,3%
 Python		19,3%
 Python		19,2%

Esta distribución es coherente con la planificación Round Robin, uno de los algoritmos clásicos utilizados por los sistemas operativos modernos. En este algoritmo, cada proceso recibe un "quantum" de tiempo de CPU, y luego se pone en espera para que otro proceso pueda ejecutarse. Esto garantiza equidad y uso compartido del procesador, tal como se observa en la práctica.

En cambio, si el sistema usara planificación FIFO (First In, First Out), el primer proceso que llega se ejecutaría hasta completarse, y los demás esperarían su turno sin compartir CPU. En ese caso, el Administrador de tareas mostraría un único proceso consumiendo la mayoría de los recursos, lo cual no ocurre en esta prueba.

4. Medir tiempos de transición entre estado:

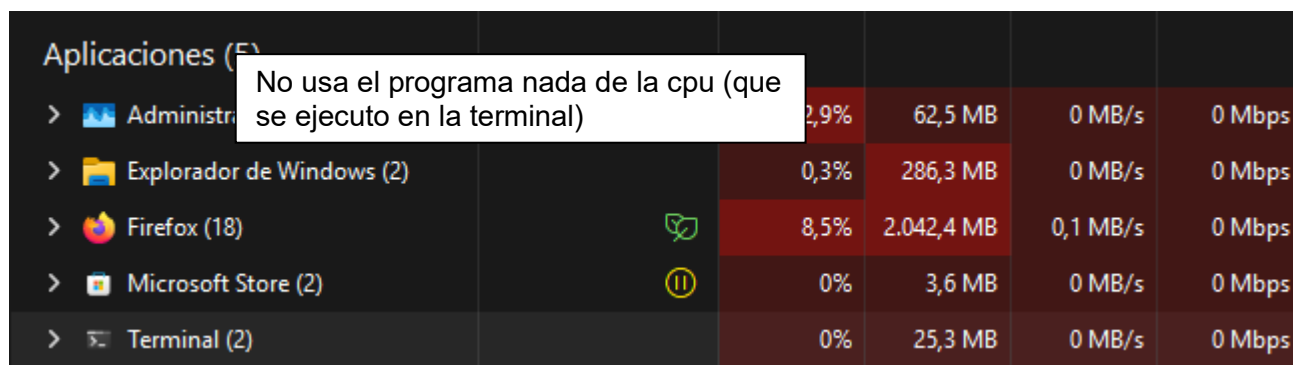
En este experimento se midió el tiempo de transición del estado Ready al estado Running en un conjunto de procesos simulados mediante Python. Se utilizó la función `perf_counter()` para obtener tiempos de alta precisión y se calcularon los tiempos transcurridos entre que un proceso se encontraba listo y comenzaba su ejecución. Se registraron los tiempos en milisegundos para cada proceso. Los resultados permitieron observar variaciones en el rendimiento dependiendo del retardo previo. Finalmente, se calculó el tiempo promedio de transición Ready → Running. Esta métrica es útil para evaluar la eficiencia de los algoritmos de planificación de procesos.

```
PS C:\Users\user\OneDrive\Desktop\Nueva carpeta (9)> py estados.py
Proceso 1: Tiempo Ready→Running = 1.661 ms
Proceso 2: Tiempo Ready→Running = 4.720 ms
Proceso 3: Tiempo Ready→Running = 3.023 ms
Proceso 4: Tiempo Ready→Running = 2.038 ms
Proceso 5: Tiempo Ready→Running = 1.680 ms

Tiempo promedio Ready→Running: 2.624 ms
```

5. Deadlock:

Se simuló un deadlock real mediante el uso de hilos y bloqueos (locks) en Python. Dos hilos intentan acceder a recursos bloqueados por el otro, generando una espera mutua infinita. Se documentaron los mensajes en consola y se explicó el fenómeno, por lo cual en la siguiente captura vemos como no usa recursos de la cpu



Aplicaciones (F5)					
>	Administración de dispositivos		2,9%	62,5 MB	0 MB/s 0 Mbps
>	Explorador de Windows (2)		0,3%	286,3 MB	0 MB/s 0 Mbps
>	Firefox (18)		8,5%	2.042,4 MB	0,1 MB/s 0 Mbps
>	Microsoft Store (2)		0%	3,6 MB	0 MB/s 0 Mbps
>	Terminal (2)		0%	25,3 MB	0 MB/s 0 Mbps

Archivos incluidos:

- proceso.py
- cpu_loader.py
- deadlock.py
- Capturas de pantalla de cada estado del proceso y del uso del CPU
- datos_mediciones.xlsx con tiempos registrados

Conclusión:

Este laboratorio permitió observar de manera práctica el ciclo de vida de un proceso, cómo el sistema operativo maneja múltiples procesos de forma concurrente, y cómo un mal manejo de recursos puede generar deadlocks. Estas observaciones son fundamentales para entender el comportamiento interno del sistema operativo que en este caso se utiliza Windows 11 home.