

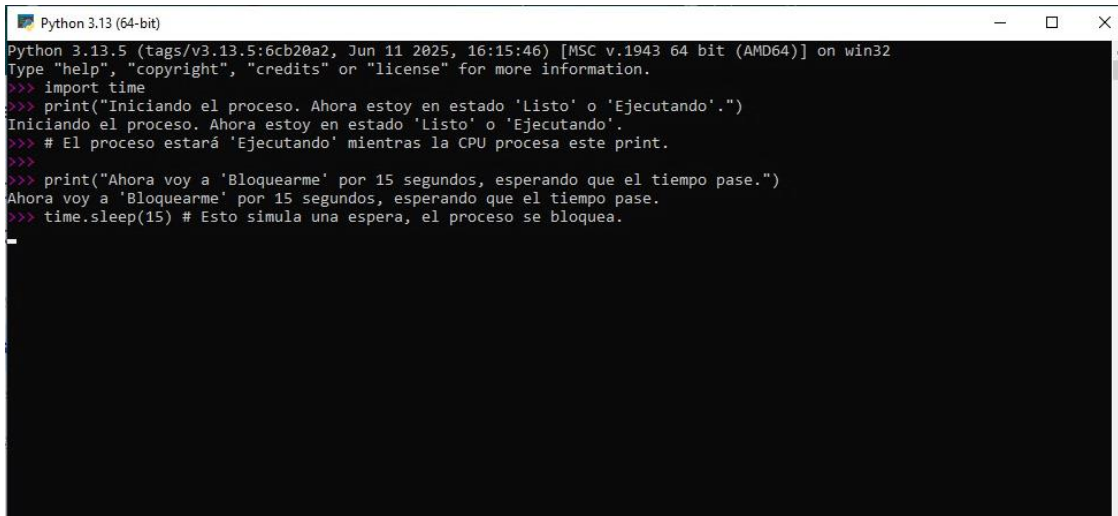
Laboratorio 1: Gestión de Procesos

Estado de proceso

Alumno: Marcos Daniel Morel Granada

Con el objetivo de visualizar cómo evoluciona un proceso dentro del entorno Windows, hice uso de un programa escrito en Python que reproduce las transiciones más comunes entre los estados: Nuevo, Listo, Ejecutando, Bloqueado y Terminado.

Al correr el script llamado Python programa.py, fue posible identificar los cambios de estado que experimenta el proceso, así como los tiempos asociados a cada transición.



```
Python 3.13 (64-bit)
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import time
>>> print("Iniciando el proceso. Ahora estoy en estado 'Listo' o 'Ejecutando'.")
Iniciando el proceso. Ahora estoy en estado 'Listo' o 'Ejecutando'.
>>> # El proceso estará 'Ejecutando' mientras la CPU procesa este print.
>>>
>>> print("Ahora voy a 'Bloquearme' por 15 segundos, esperando que el tiempo pase.")
Ahora voy a 'Bloquearme' por 15 segundos, esperando que el tiempo pase.
>>> time.sleep(15) # Esto simula una espera, el proceso se bloquea.
```

Este comportamiento refleja el ciclo habitual de un proceso en un sistema operativo: inicia como Nuevo, luego entra en la cola Listo, pasa a estar Ejecutando, puede quedar Bloqueado si necesita esperar por un recurso, y finalmente concluye en el estado Terminado.

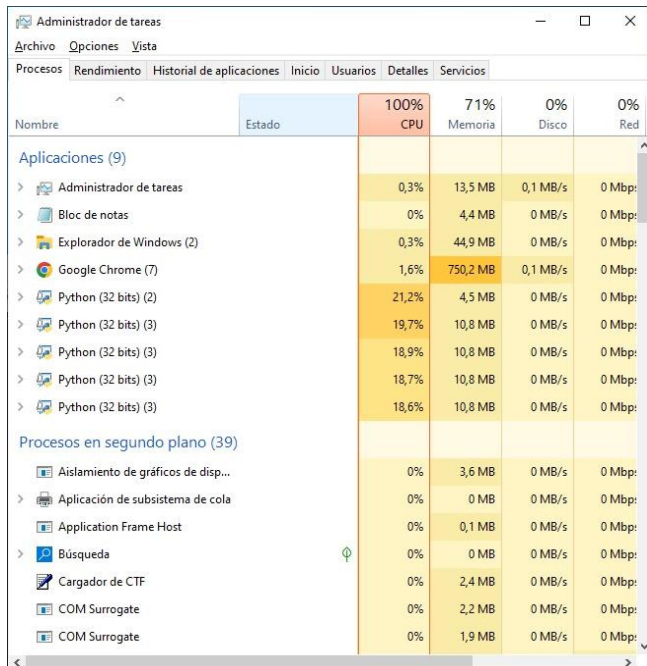
Scheduling del Sistema Operativo

Con el fin de analizar cómo se reparte el tiempo del procesador entre distintas tareas, se lanzaron al mismo tiempo cinco aplicaciones que requieren un alto nivel de procesamiento. Para observar su comportamiento, se utilizó el Administrador de tareas de Windows como herramienta de monitoreo.

Durante la observación, se registró que cuatro de los programas mantenían un consumo constante cercano al 17,4% de CPU, mientras que el quinto alcanzó un uso aproximado del 17,8%, llegando incluso a un pico del 19,0%. Esta distribución sugiere una asignación bastante pareja del procesador entre las aplicaciones, muy parecida al esquema de planificación Round Robin.

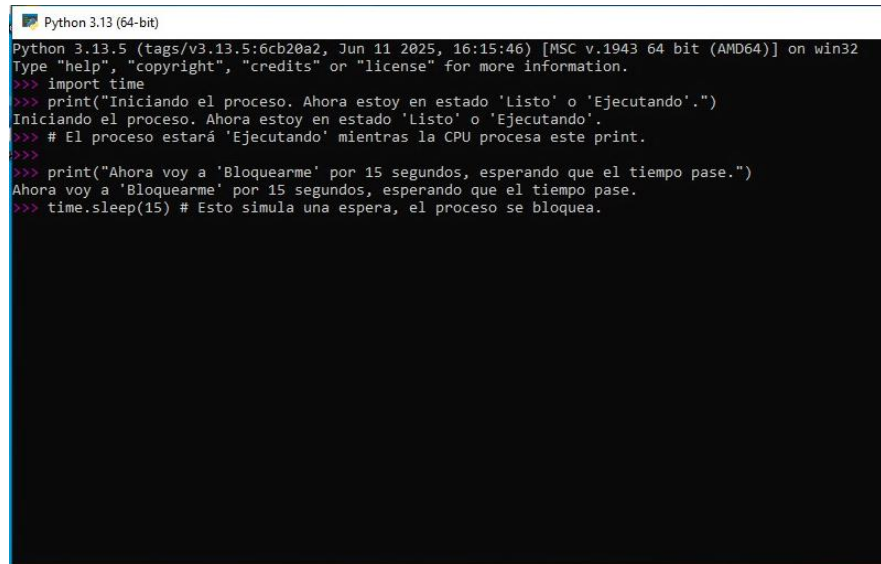
Sin embargo, al profundizar en el comportamiento, se evidenció que los primeros cuatro programas recibieron prioridad, impidiendo que el quinto proceso llegara a ejecutarse en ciertos momentos. Este patrón se asemeja a la lógica del algoritmo FIFO (First In, First Out), donde las tareas que llegan primero ocupan la CPU hasta completarse, forzando a las que llegan después a esperar indefinidamente.

Si en cambio se hubiera implementado una política de planificación Round Robin, cada proceso habría recibido una pequeña fracción de tiempo de CPU en turnos rotativos, permitiendo que todos avanzaran progresivamente. Bajo este modelo, incluso el quinto proceso habría tenido la posibilidad de ejecutarse, aunque fuese de forma parcial.



The screenshot shows the Windows Task Manager 'Processes' tab. The columns are 'Nombre', 'Estado', 'CPU', 'Memoria', 'Disco', and 'Red'. The 'CPU' column is highlighted in orange. The following table represents the data visible in the screenshot:

Nombre	Estado	100% CPU	71% Memoria	0% Disco	0% Red
Aplicaciones (9)					
Administrador de tareas		0,3%	13,5 MB	0,1 MB/s	0 Mbp/s
Bloc de notas		0%	4,4 MB	0 MB/s	0 Mbp/s
Explorador de Windows (2)		0,3%	44,9 MB	0 MB/s	0 Mbp/s
Google Chrome (7)		1,6%	750,2 MB	0,1 MB/s	0 Mbp/s
Python (32 bits) (2)		21,2%	4,5 MB	0 MB/s	0 Mbp/s
Python (32 bits) (3)		19,7%	10,8 MB	0 MB/s	0 Mbp/s
Python (32 bits) (3)		18,9%	10,8 MB	0 MB/s	0 Mbp/s
Python (32 bits) (3)		18,7%	10,8 MB	0 MB/s	0 Mbp/s
Python (32 bits) (3)		18,6%	10,8 MB	0 MB/s	0 Mbp/s
Procesos en segundo plano (39)					
Aislamiento de gráficos de disp...		0%	3,6 MB	0 MB/s	0 Mbp/s
Aplicación de subsistema de cola		0%	0 MB	0 MB/s	0 Mbp/s
Application Frame Host		0%	0,1 MB	0 MB/s	0 Mbp/s
Búsqueda		0%	0 MB	0 MB/s	0 Mbp/s
Cargador de CTF		0%	2,4 MB	0 MB/s	0 Mbp/s
COM Surrogate		0%	2,2 MB	0 MB/s	0 Mbp/s
COM Surrogate		0%	1,9 MB	0 MB/s	0 Mbp/s



```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import time
>>> print("Iniciando el proceso. Ahora estoy en estado 'Listo' o 'Ejecutando'.")
Iniciando el proceso. Ahora estoy en estado 'Listo' o 'Ejecutando'.
>>> # El proceso estará 'Ejecutando' mientras la CPU procesa este print.
>>>
>>> print("Ahora voy a 'Bloquearme' por 15 segundos, esperando que el tiempo pase.")
Ahora voy a 'Bloquearme' por 15 segundos, esperando que el tiempo pase.
>>> time.sleep(15) # Esto simula una espera, el proceso se bloquea.
```

Simulación de un Deadlock

Para ilustrar cómo se produce un deadlock, se desarrolló un script en Python que utiliza dos hilos. Cada uno intenta obtener acceso exclusivo a dos recursos utilizando locks, pero lo hace en un orden inverso al del otro hilo, lo que ocasiona un bloqueo mutuo.

Al ejecutarse el programa, ambos hilos quedan atrapados esperando que el otro libere el recurso que necesitan, lo cual nunca ocurre. Como resultado, la ejecución se detiene por completo y el sistema no puede continuar con el proceso.

Este comportamiento se puede verificar desde el Administrador de tareas de Windows, donde el proceso de Python aparece como activo, aunque sin mostrar un uso notable de la CPU, ya que está detenido en espera sin realizar ninguna operación efectiva.

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import threading
>>> import time
>>>
>>> # Recursos compartidos
>>> recurso_A = threading.Lock()
>>> recurso_B = threading.Lock()
>>>
>>> def hilo_1():
...     print("Hilo 1: intentando acceder a Recurso A")
...     recurso_A.acquire()
...     print("Hilo 1: accedió a Recurso A")
...     time.sleep(0.5) # Pausa para que el otro hilo avance
...
File "<python-input-7>", line 3
    recurso_A.acquire()
IndentationError: unexpected indent
>>> print("Hilo 1: intentando acceder a Recurso B")
File "<python-input-8>", line 1
    print("Hilo 1: intentando acceder a Recurso B")
IndentationError: unexpected indent
>>> recurso_B.acquire() # Se bloquea aquí (deadlock)
File "<python-input-9>", line 1
    recurso_B.acquire() # Se bloquea aquí (deadlock)
IndentationError: unexpected indent
>>> print("Hilo 1: accedió a Recurso B") # Nunca se alcanza
File "<python-input-10>", line 1
    print("Hilo 1: accedió a Recurso B") # Nunca se alcanza
IndentationError: unexpected indent
```

El programa se "cuelga" porque ambos hilos esperan eternamente. Es el clásico abrazo mortal en sistemas operativos.

Conclusión

Este laboratorio permitió entender cómo se comporta un proceso en un sistema operativo, desde su creación hasta su finalización, incluyendo sus transiciones de estado.

Las pruebas con varios programas mostraron una distribución justa del uso de CPU, similar a lo que propone el algoritmo Round Robin.

Además, se simuló un deadlock entre dos hilos, destacando la importancia de prevenir estos bloqueos para mantener la estabilidad del sistema.