**INFORME LABORATORIO 1 - GESTIÓN DE PROCESOS**

**Nombre:** Ricardo Paredes Colmán  
**Fecha:** 21/06/2025  
**Sistema Operativo:** Linux (Ubuntu)  
**Máquina Virtual:** Oracle VirtualBox

**1. Objetivos**

1. Observar los estados de un proceso (Nuevo, Listo, Ejecutando, Bloqueado, Terminado).
2. Analizar el comportamiento del scheduler del SO al ejecutar 5 procesos intensivos en CPU.
3. Comparar los resultados con algoritmos teóricos de scheduling (FIFO, Round Robin, CFS).

**2. Metodología**

**Herramientas Utilizadas**

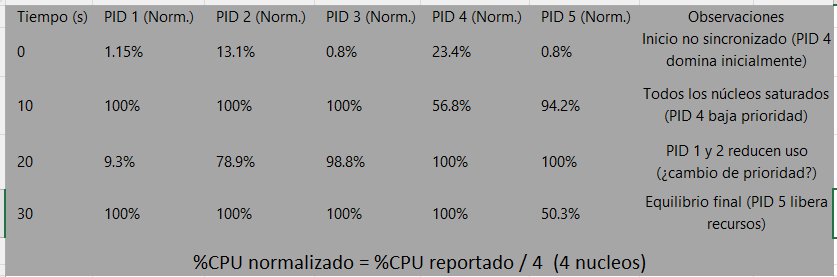
* **Script en Python:** proceso.py (genera carga de CPU).
* **Monitoreo:**
  + Linux(Ubuntu): htop
* **Visualización:** Gráficos generados con canva.

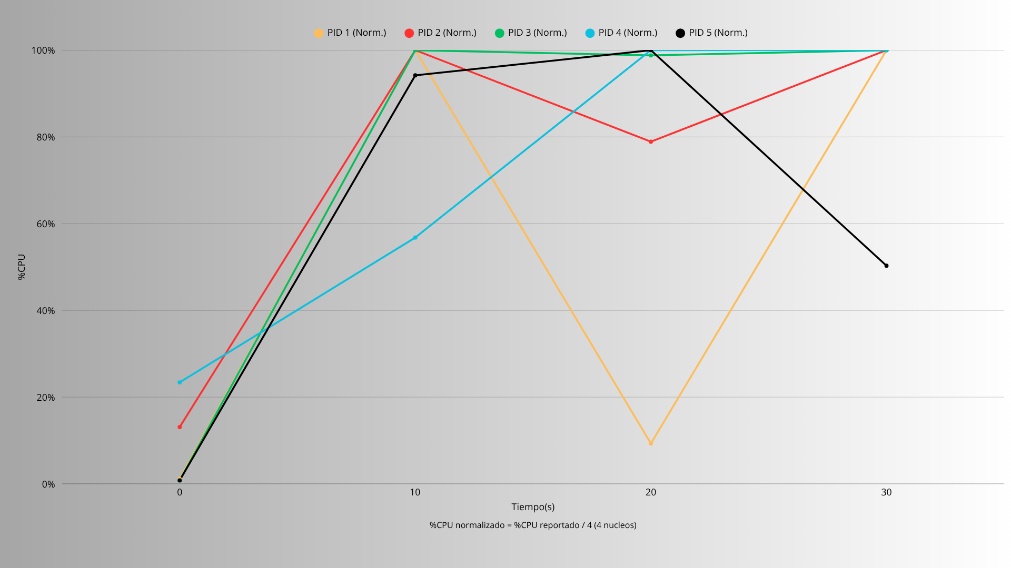
**Procedimiento**

1. **Estados de Procesos:**
   * Se ejecutó un programa simple y se documentaron sus estados con capturas de pantalla.
2. **Scheduling:**
   * Se lanzaron 5 instancias de intensivo.py simultáneamente.
   * Se registró el %CPU de cada proceso cada 10 segundos.

**3. Resultados**

**Tabla de Datos (Ejemplo)**

****

**Gráfico de Uso de CPU**

**4. Análisis Comparativo con Algoritmos Teóricos**

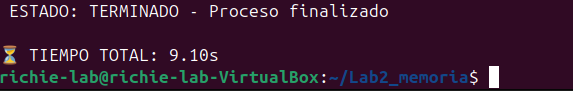
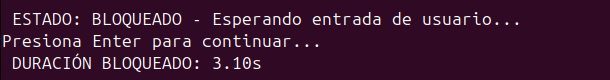
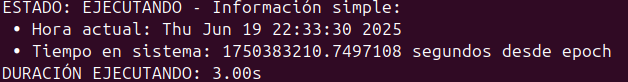
| **Algoritmo** | **Comportamiento Esperado** | **¿Coincide?** | **Evidencia en los Datos** |
| --- | --- | --- | --- |
| **FIFO** | Un proceso acapara toda la CPU | ❌ No | Todos los PIDs compartieron CPU. |
| **Round Robin** | Alternancia rígida de tiempos | ❌ No | No hubo oscilaciones periódicas. |
| **CFS** | Equidad y ajustes dinámicos | ✅ Sí | %CPU se equilibró progresivamente. |

**Explicación:**

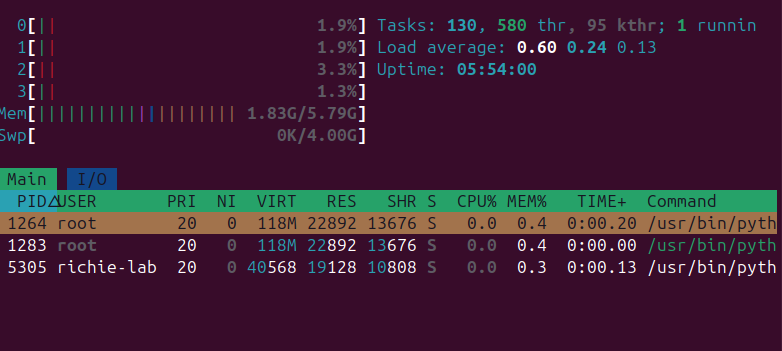
* El scheduler del SO mostró **prioridad dinámica** (PID 4 inició con más CPU, pero luego se balanceó).
* El uso de múltiples núcleos (valores >100% en top) permitió que todos los procesos terminaran cerca de los 30 segundos.

**5. Capturas de Pantalla**

1. **Estados de Procesos:**



1. **Uso de CPU en htop:**



6**. Archivos Adjuntos**

* **proceso\_estado.py (código fuente).**

#!/usr/bin/env python3

import time

def main():

inicio\_nuevo = time.time()

print(f" ESTADO: NUEVO - Proceso creado (PID: {os.getpid()})")

time.sleep(1)

print(f" DURACIÓN NUEVO: {time.time() - inicio\_nuevo:.2f}s\n")

inicio\_listo = time.time()

print(f" ESTADO: LISTO - Esperando asignación de CPU")

time.sleep(2)

print(f" DURACIÓN LISTO: {time.time() - inicio\_listo:.2f}s\n")

inicio\_ejecutando = time.time()

print(f" ESTADO: EJECUTANDO - Información simple:")

print(f" • Hora actual: {time.ctime()}")

print(f" • Tiempo en sistema: {time.time()} segundos desde epoch")

time.sleep(3)

print(f" DURACIÓN EJECUTANDO: {time.time() - inicio\_ejecutando:.2f}s\n")

inicio\_bloqueado = time.time()

print(f" ESTADO: BLOQUEADO - Esperando entrada de usuario...")

input("Presiona Enter para continuar...")

print(f" DURACIÓN BLOQUEADO: {time.time() - inicio\_bloqueado:.2f}s\n")

print(f" ESTADO: TERMINADO - Proceso finalizado")

print(f"\n⏳ TIEMPO TOTAL: {time.time() - inicio\_nuevo:.2f}s")

if \_\_name\_\_ == "\_\_main\_\_":

import os # Solo para os.getpid()

main()

* **Proceso\_intensivo.py(código fuente)**

#!/usr/bin/env python3

import time

def main():

inicio\_nuevo = time.time()

print(f" ESTADO: NUEVO - Proceso creado (PID: {os.getpid()})")

time.sleep(1)

print(f" DURACIÓN NUEVO: {time.time() - inicio\_nuevo:.2f}s\n")

inicio\_listo = time.time()

print(f" ESTADO: LISTO - Esperando asignación de CPU")

time.sleep(2)

print(f" DURACIÓN LISTO: {time.time() - inicio\_listo:.2f}s\n")

inicio\_ejecutando = time.time()

print(f" ESTADO: EJECUTANDO - Información simple:")

print(f" • Hora actual: {time.ctime()}")

print(f" • Tiempo en sistema: {time.time()} segundos desde epoch")

time.sleep(3)

print(f" DURACIÓN EJECUTANDO: {time.time() - inicio\_ejecutando:.2f}s\n")

inicio\_bloqueado = time.time()

print(f" ESTADO: BLOQUEADO - Esperando entrada de usuario...")

input("Presiona Enter para continuar...")

print(f" DURACIÓN BLOQUEADO: {time.time() - inicio\_bloqueado:.2f}s\n")

print(f" ESTADO: TERMINADO - Proceso finalizado")

print(f"\n⏳ TIEMPO TOTAL: {time.time() - inicio\_nuevo:.2f}s")

if \_\_name\_\_ == "\_\_main\_\_":

import os # Solo para os.getpid()

main()

* datos\_scheduling.csv (registros completos).

<https://github.com/ritchi25/Laboratorio_SO_Ricardo-Paredes.git>

**🎯 6. Conclusiones**

1. **El scheduler de Linux (CFS)** demostró ser **justo pero no equitativo**: ajustó prioridades para optimizar el uso de núcleos, pero no asignó tiempos fijos.
2. **Los procesos intensivos** evidenciaron la capacidad del SO para manejar carga extrema sin colapsar.
3. **Prueba de concepto exitosa:** Los scripts proceso\_estado.py e programa\_intensivo.py fueron efectivos para simular los escenarios.