Trabajo Práctico 1: *Scheduling*

Sistemas Operativos

Departamento de Computación

FCEyN - UBA

Segundo Cuatrimestre 2025

Introducción

Este trabajo práctico tiene como objetivo consolidar conocimientos teóricos y prácticos relacionados con políticas de *scheduling* o planificación de procesos. Para esto, usaremos un simulador del funcionamiento de un *scheduler* bajo un esquema *Multi-level feedback queue*. A través de las consignas, abordaremos aspectos implementativos y experimentales que deberán describirse detalladamente en un informe, justificando las decisiones tomadas y las hipótesis planteadas.

Simulador

Dentro del archivo comprimido del TP encontrarán un archivo de extensión *.py* que contiene un programa **simulador** de *Multi-level feedback queue* que nos ayudará a entender cómo funciona este esquema de *scheduling* en la práctica. Para detalles de uso, **consultar el apéndice A** donde encontrarán una guía completa para la utilización del simulador.

Consignas

Resolver **de forma individual** los siguientes ejercicios. Para cada caso, generar tests y realizar gráficos representativos de los resultados obtenidos.

1. Para familiarizarte con el simulador, generar dos problemas con datos aleatorios *(notar que es una posibilidad nativa del simulador)* deshabilitando E/S. Incluir en el informe una explicación breve de los resultados obtenidos y el diagrama de Gantt para uno de los dos problemas.
   Nota: Te podés ayudar limitando la duración de las tareas a un valor razonable y corto (entre 20 y 30ms por ejemplo).

2. El simulador actual provee la posibilidad de conocer el **responseTime** y **turnaround** para cada uno de los trabajos. **Extender** las funcionalidades del simulador para que calcule también las métricas **Throughput** y **Waiting time** individual y promedio. Se debe imprimir el **Throughput** calculado cada 2 unidades de tiempo, mostrando cómo evoluciona a lo largo de la ejecución. Incluir en el informe una descripción breve de la implementación.

3. Un programa *malicioso* puede aprovecharse y usar el mecanismo de llamada a E/S para acaparar recursos del CPU. Simular un escenario con dos trabajos en donde ocurra esta situación. Justificar en el informe porqué se da esta situación.

4. Considerando la siguiente lista de trabajos (con duración de 5ms para cada operación de E/S):

   - $J_1 = (0, 30, 0)$: intensivo en CPU (llega en $t = 0$, corre 30ms, sin E/S).
   - $J_2 = (0, 8, 2)$: interactivo (llega en $t = 0$, corre 8ms, pide E/S cada 2ms).
   - $J_3 = (10, 12, 0)$: uso mediano de CPU (llega en $t = 10$, corre 12ms, sin E/S).

   Hallar dos configuraciones para el simulador: una que sea beneficiosa para el trabajo interactivo (y perjudicial para el resto) y otra que sea exactamente inversa (es decir, perjudicial para el trabajo interactivo y beneficioso para el resto). Describir cada configuración en el informe y justificar el beneficio/perjuicio con base en las métricas que genere cada ejecución.

Requerimientos de Entrega

- El trabajo debe entregarse en un único archivo comprimido (`.zip` o `.tar.gz`) que contenga:
  - El informe, en formato PDF.
  - El archivo `.py` modificado con las implementaciones solicitadas en las consignas.
  - Todos los archivos necesarios para reproducir la experimentación incluida en el informe.
- Fecha de entrega: **28/09/2025**.
- La entrega se realizará a través del siguiente formulario: https://forms.gle/nbiJSHn6iWZZ3LV19
- **No se aceptarán entregas más allá de la fecha límite, sin excepción.**

Criterios de Evaluación

- El código que escriban deberá seguir buenas prácticas de programación, como usar nombres descriptivos para variables y funciones e incluir comentarios donde consideren necesario realizar aclaraciones.
- El código entregado deberá resolver correctamente las consignas.
- El informe deberá ser claro, prolijo, y conciso —evaluaremos la calidad de su contenido por sobre su cantidad—.

- El informe deberá explicar cada una de las decisiones tomadas en las consignas. Si en el texto del informe incluyen referencias al código, las mismas deberán ser claras.
- El informe deberá explicar adecuadamente la experimentación realizada, que justifique cada una de las respuestas (reproducible, con gráficos comparativos, que den soporte a las hipótesis, etc).

Bibliografía

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. Wiley.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems*. Pearson.
- Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*

A. Manual del Simulador MLFQ

Nota de Referencia: El simulador utilizado en este trabajo práctico corresponde a la implementación provista por el libro **Operating Systems: Three Easy Pieces (OSTEP)**, de Remzi H. Arpaci-Dusseau y Andrea C. Arpaci-Dusseau. Se otorgan todos los créditos de la implementación a sus autores.

Descripción

Este programa permite observar el comportamiento de un *scheduler Multilevel Feedback Queue (MLFQ)* presentado en el capítulo correspondiente de OSTEP. Se puede utilizar tanto para generar problemas de forma aleatoria (mediante semillas), como para construir experimentos cuidadosamente diseñados que permitan estudiar el funcionamiento de MLFQ en diferentes circunstancias.

Uso

Para ejecutar el programa:

```
prompt> ./mlfq.py
```

Para ver todas las opciones:

```
prompt> ./mlfq.py -h
```

Opciones principales

```
-s SEED        semilla para la generación aleatoria
-n NUMQUEUES   número de colas en MLFQ (si no se usa -Q)
-q QUANTUM     tamaño del quantum en ms (si no se usa -Q)
-a ALLOTMENT   cantidad de quantums por cola antes de bajar de prioridad
               (si no se usa -A)
-Q LISTA       lista de quantums por nivel de cola (ej: 10,20,40)
-A LISTA       lista de allotments por nivel de cola (ej: 1,2,4)
-j NUMJOBS     número de trabajos en el sistema
-m MAXLEN      duración máxima de un trabajo (si se generan aleatoriamente)
-M MAXIO       frecuencia máxima de E/S (si se generan aleatoriamente)
-B BOOST       cada cuántos ms se hace un priority boost
-i IOTIME      duración fija de una operación de E/S (por defecto: 5 ms)
-S             trabajos con E/S no bajan de prioridad
-I             trabajos que terminan E/S pasan al frente de la cola
-l JLIST       lista explícita de trabajos (x,y,z:...)
-c             calcula métricas automáticamente
```

Ejemplo: trabajos aleatorios

```
prompt> ./mlfq.py -j 3
```

Genera tres trabajos al azar con parámetros por defecto. Si se agrega la opción `-c`, además de la traza de ejecución se muestran tiempos de respuesta y finalización de cada trabajo.

Ejemplo: lista explícita de trabajos

```
prompt> ./mlfq.py --jlist 0,180,0:100,20,0 -q 10
```

Este comando crea dos trabajos:

- Trabajo 0: llega en $t = 0$, requiere 180 ms de CPU, nunca hace E/S.
- Trabajo 1: llega en $t = 100$, requiere 20 ms de CPU, nunca hace E/S.

Ambos son planificados en un MLFQ de 3 colas, con quantum uniforme de 10 ms.