Nombre: Laura Valentina Bernal Lozada

Código: 2437088 - 2724

MLQ (MULTILEVEL QUEUE) SCHEDULING

1. ¿Cómo funciona el algoritmo MLQ?

El algoritmo de planificación por colas multinivel (Multilevel Queue Scheduling, MLQ) es una técnica de planificación de CPU) que organiza los procesos en múltiples colas, donde cada cola tiene su propia política de planificación interna (por ejemplo, Round Robin, FCFS, etc.) y un nivel de prioridad fijo con respecto a las demás.

Cada proceso se asigna a una cola según su tipo o prioridad, y las colas se ejecutan por niveles de prioridad, es decir, las colas superiores se ejecutan antes que las inferiores. La idea principal del MLQ es dividir los procesos del sistema en clases o tipos que requieran diferentes formas de tratamiento. Por ejemplo, algunos procesos pueden ser interactivos (que necesitan respuesta rápida), mientras que otros pueden ser por lotes (que pueden esperar más tiempo).

Este método permite manejar diferentes tipos de procesos en un mismo sistema, equilibrando la eficiencia y la justicia en la asignación del CPU.

En esta implementación se utilizó el siguiente esquema de colas:

Nivel de Cola	Política de Planificación	Quantum	Prioridad
Cola 1	Round Robin (RR)	3	Alta
Cola 2	Round Robin (RR)	5	Media
Cola 3	First Come, First Served (FCFS)	—	Baja

Cada proceso se caracteriza por los siguientes atributos:

- **Etiqueta (Label)**: nombre del proceso (A, B, C, …)
- **Burst Time (BT)**: tiempo total de CPU requerido.
- **Arrival Time (AT)**: tiempo en que el proceso llega al sistema.
- **Queue (Q)**: número de la cola a la que pertenece.

El algoritmo calcula las métricas:

- **WT (Waiting Time)**: tiempo que el proceso espera en la cola antes de ser ejecutado.
- **CT (Completion Time)**: tiempo total en el que el proceso termina.
- **RT (Response Time)**: tiempo desde su llegada hasta que recibe CPU por primera vez.
- **TAT (Turnaround Time)**: tiempo total desde que llega hasta que termina.

Prioridad entre colas:
Cada cola tiene una prioridad diferente.
Las colas de prioridad más alta siempre se ejecutan antes que las colas de menor prioridad.
Es decir, la CPU atiende primero todos los procesos de la cola 1, luego los de la cola 2, y finalmente los de la cola 3.

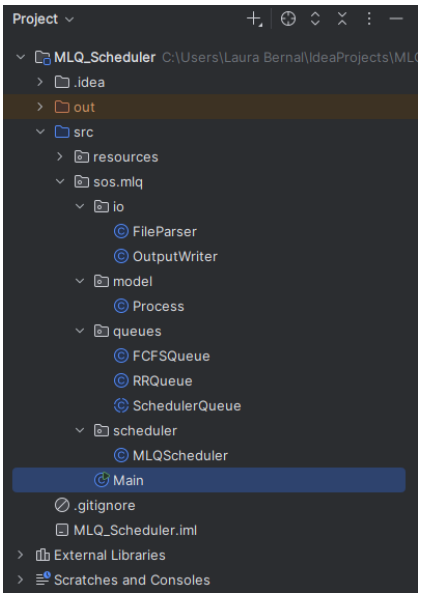
Ejecución dentro de cada cola:

- Las colas 1 y 2 utilizan **Round Robin**, lo que significa que cada proceso obtiene un tiempo limitado de CPU llamado **quantum**.
- Si un proceso no termina en su quantum, regresa al final de la misma cola para esperar su siguiente turno. Esto garantiza **justicia** y evita que un solo proceso acapare el procesador.
- La cola 3 utiliza **FCFS**, donde los procesos se ejecutan **en el orden en que llegan** y **sin interrupciones**.

No existe migración entre colas:
En este tipo de planificación, un proceso **permanece siempre en la misma cola** en la que fue asignado originalmente.
No puede cambiar de nivel, incluso si su comportamiento varía.

2. Implementación en Java

El proyecto fue desarrollado en IntelliJ IDEA usando Java 17 con el paradigma Orientado a Objetos.



La estructura de paquetes es la siguiente:

- **Process:** modelo que representa cada proceso con sus atributos y métricas.
- **SchedulerQueue:** clase abstracta base para las colas.
- **RRQueue y FCFSQueue:** implementan las políticas de planificación.
- **MLQScheduler:** gestiona las tres colas y las ejecuta por orden de prioridad.
- **FileParser y OutputWriter:** manejan la lectura y escritura de archivos.
- **Main**: punto de entrada que conecta todo el sistema.

DESCRIPCIÓN DE CÓMO SE APLICÓ EL ALGORITMO EN LA IMPLEMENTACIÓN

En la implementación desarrollada, el sistema de planificación Multilevel Queue (MLQ) se estructura a través de tres colas jerárquicas, cada una con su propio algoritmo de planificación y nivel de prioridad. Estas colas funcionan de la siguiente manera:

Cola 1: emplea el algoritmo Round Robin (RR) con un quantum de 3, asignado a los procesos de mayor prioridad. Este enfoque permite una distribución equitativa del tiempo de CPU entre los procesos más importantes, evitando que alguno monopolice el procesador.

Cola 2: también utiliza Round Robin, pero con un quantum de 5, destinado a procesos de prioridad media. De esta forma, se permite un tiempo de ejecución ligeramente mayor, equilibrando eficiencia y justicia entre los procesos intermedios.

Cola 3: implementa el algoritmo First Come, First Served (FCFS), asignado a los procesos de menor prioridad. En esta cola, los procesos se ejecutan estrictamente en el orden de llegada, sin interrupciones ni cambios de turno.

Los datos de entrada son obtenidos desde un archivo de texto (.txt), en el cual cada línea representa un proceso con sus atributos principales: etiqueta, tiempo de ráfaga (BT), tiempo de llegada (AT) y número de cola (Q).

Cada proceso leído se clasifica automáticamente en su cola correspondiente de acuerdo con el nivel indicado en el archivo. Posteriormente, el planificador principal (MLQScheduler) ejecuta las colas de forma secuencial, respetando el orden de prioridad establecido **(Cola 1 → Cola 2 → Cola 3).**
Durante la ejecución, el programa calcula los principales indicadores de desempeño de la planificación (WT, CT, RT y TAT)

Todos estos resultados se guardan automáticamente en un archivo de salida, lo que permite analizar el comportamiento y eficiencia del algoritmo implementado.

RESUMEN DEL FLUJO DEL PROGRAMA

El flujo general del programa está basado en los principios de la programación orientada a objetos (POO). Primero, el sistema inicia con la lectura del archivo de entrada a través de la clase **FileParser,** la cual se encarga de interpretar los datos y convertir cada línea en un objeto de tipo **Process.**

Una vez creados los objetos, la clase **MLQScheduler** actúa como el controlador central del planificador. Esta clase distribuye los procesos en sus respectivas colas **(RRQueue, RRQueue, y FCFSQueue)** y se encarga de coordinar la ejecución de cada una en orden de prioridad.

Después de que todas las colas finalizan su ejecución, los resultados son procesados por la clase **OutputWriter**, que genera un archivo de texto con el detalle de cada proceso **(tiempos WT, CT, RT, TAT)** y los promedios globales de las métricas.

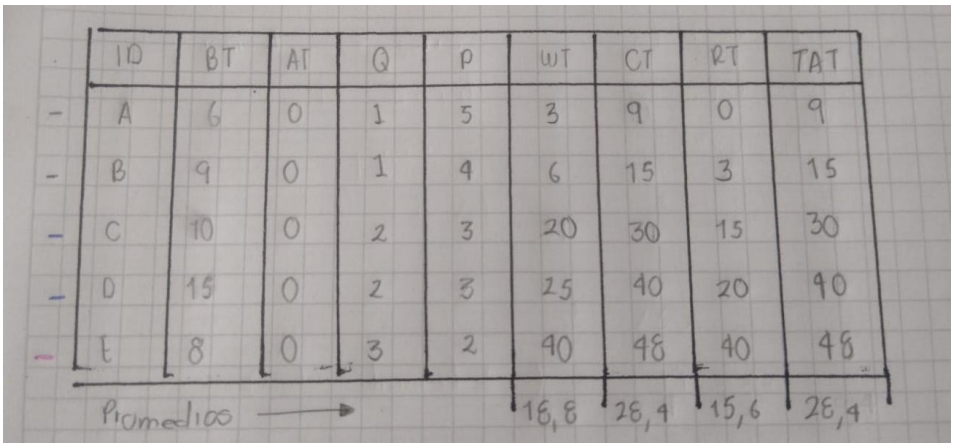
Este flujo de trabajo garantiza que los procesos más prioritarios reciban atención preferente del CPU, mientras que los de menor prioridad también sean atendidos de forma ordenada. A su vez, la separación por clases y la jerarquía de colas permiten una implementación clara, escalable y fiel al comportamiento esperado del algoritmo MLQ.

3. Entradas y Resultados Esperados

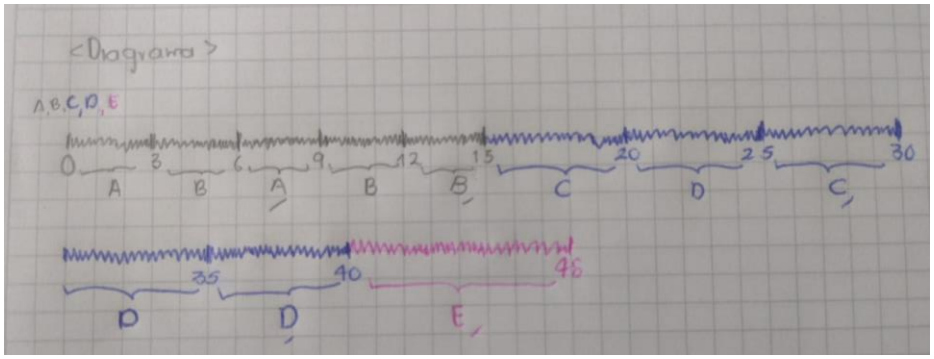
A continuación, se muestran los tres archivos de entrada utilizados y los resultados esperados en su implementación

```
# Archivo: mlq001.txt
# etiqueta; burst time (BT); arrival time (AT); Queue (Q); Priority (5 > 1)
A; 6; 0; 1; 5
B; 9; 0; 1; 4
C; 10; 0; 2; 3
D; 15; 0; 2; 3
E; 8; 0; 3; 2
```

RESULTADO ESPERADO EN LA IMPLEMENTACIÓN



ID	BT	AT	Q	P	WT	CT	RT	TAT
A	6	0	1	5	3	9	0	9
B	9	0	1	4	6	15	3	15
C	10	0	2	3	20	30	15	30
D	15	0	2	3	25	40	20	40
E	8	0	3	2	40	48	40	48
Promedios →					18,8	28,4	15,6	28,4

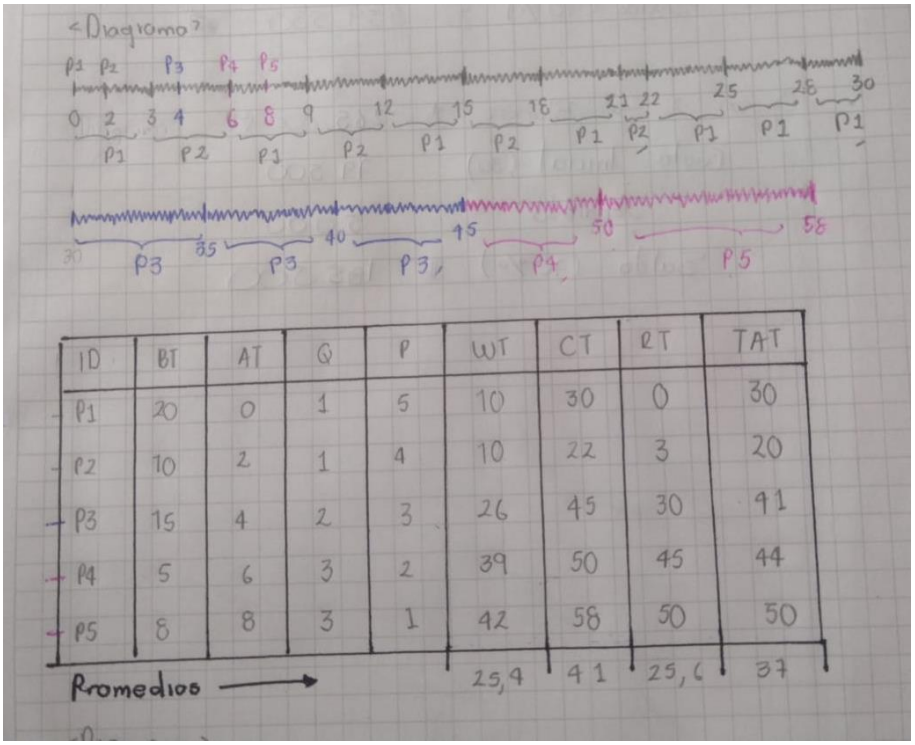


RESULTADOS OBTENIDOS EN LA IMPLEMENTACIÓN

```
# Etiqueta; BT; AT; Q; WT; CT; RT; TAT
A;6;0;1;3;9;0;9
B;9;0;1;6;15;3;15
C;10;0;2;20;30;15;30
D;15;0;2;25;40;20;40
E;8;0;3;40;48;40;48

WT=18,80; CT=28,40; RT=15,60; TAT=28,40;
```

```
# Archivo: mlq002.txt
# 09102024
# etiqueta; burst time (BT); arrival time (AT); Queue (Q); Priority (5
> 1)

p1; 20; 0; 1; 5
p2; 10; 2; 1; 4
p3; 15; 4; 2; 3
p4; 5; 6; 3; 2
p5; 8; 8; 3; 1
```

RESULTADO ESPERADO EN LA IMPLEMENTACIÓN



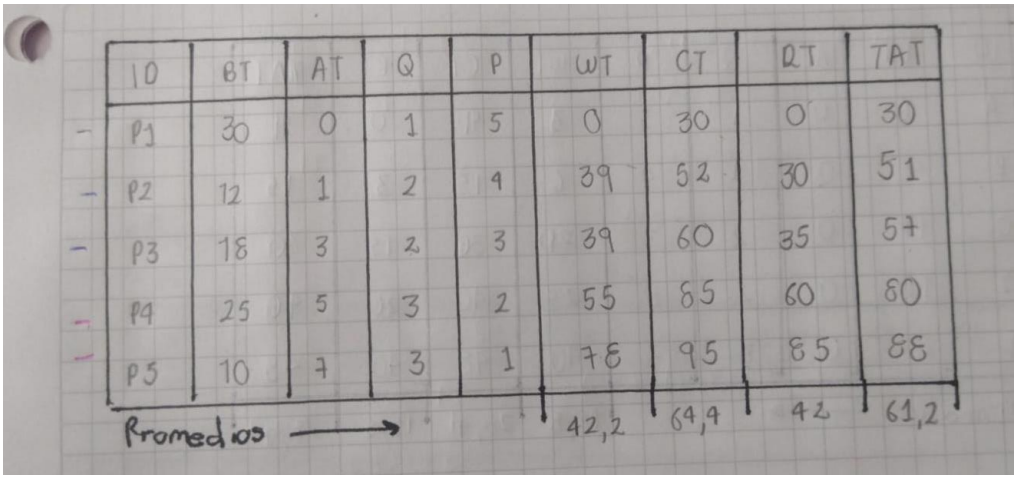
ID	BT	AT	Q	P	WT	CT	RT	TAT
P1	20	0	1	5	10	30	0	30
P2	10	2	1	4	10	22	3	20
P3	15	4	2	3	26	45	30	41
P4	5	6	3	2	39	50	45	44
P5	8	8	3	1	42	58	50	50
Promedios →					25,4	41	25,6	37

RESULTADOS OBTENIDOS EN LA IMPLEMENTACIÓN

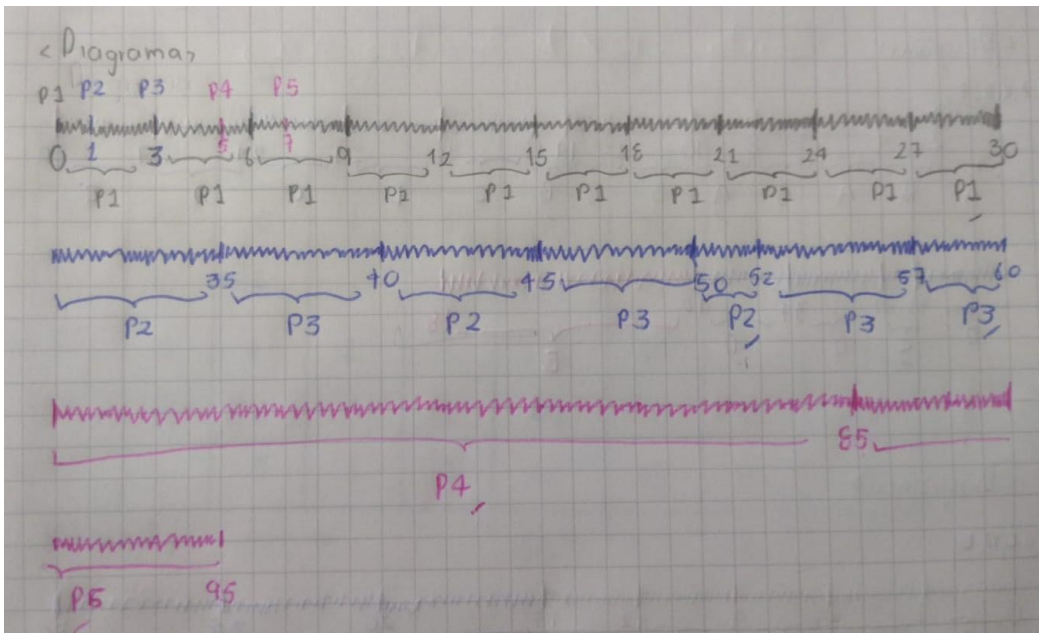
```
# Etiqueta; BT; AT; Q; WT; CT; RT; TAT
p1;20;0;1;10;30;0;30
p2;10;2;1;10;22;3;20
p3;15;4;2;26;45;30;41
p4;5;6;3;39;50;45;44
p5;8;8;3;42;58;50;50

WT=25,40; CT=41,00; RT=25,60; TAT=37,00;
```

```
# Archivo: mlq003.txt
# 02102024
# etiqueta; burst time (BT); arrival time (AT); Queue (Q); Priority (5
> 1)
p1; 30; 0; 1; 5
p2; 12; 1; 2; 4
p3; 18; 3; 2; 3
p4; 25; 5; 3; 2
p5; 10; 7; 3; 1
```

RESULTADO ESPERADO EN LA IMPLEMENTACIÓN



ID	BT	AT	Q	P	WT	CT	RT	TAT
P1	30	0	1	5	0	30	0	30
P2	12	1	2	4	39	52	30	51
P3	18	3	2	3	39	60	35	57
P4	25	5	3	2	55	85	60	80
P5	10	7	3	1	78	95	85	88
Promedios →					42,2	64,4	42	61,2



RESULTADOS OBTENIDOS EN LA IMPLEMENTACIÓN

```
# Etiqueta; BT; AT; Q; WT; CT; RT; TAT
p1;30;0;1;0;30;0;30
p2;12;1;2;39;52;30;51
p3;18;3;2;39;60;35;57
p4;25;5;3;55;85;60;80
p5;10;7;3;78;95;85;88

WT=42,20; CT=64,40; RT=42,00; TAT=61,20;
```

4. Análisis de los Resultados

1. **Los procesos de las colas con mayor prioridad (Q1 y Q2)** siempre se ejecutan antes que los de colas inferiores.
2. **El uso de RR en las colas superiores** distribuye de manera justa el tiempo de CPU entre los procesos, permitiendo que todos avancen parcialmente antes de finalizar.
3. **La cola FCFS (Q3)** ejecuta los procesos restantes de forma secuencial, sin interrupciones, cuando las otras colas están vacías.
4. **El tiempo de respuesta (RT)** es menor en las colas superiores debido al uso de RR, que otorga CPU rápidamente a los procesos recién llegados.

VIDEO EXPLICATIVO DE LA IMPLEMENTACIÓN USANDO POO:

https://drive.google.com/file/d/1Fo2f7iNVzBMYb6zulJ3yReBWWhW0iWxJ/view?usp=sharing

LINK AL REPOSITORIO PÚBLICO DE GITHUB DE LA IMPLEMENTACIÓN:

https://github.com/LauraBernal18/MLQ_Scheduler