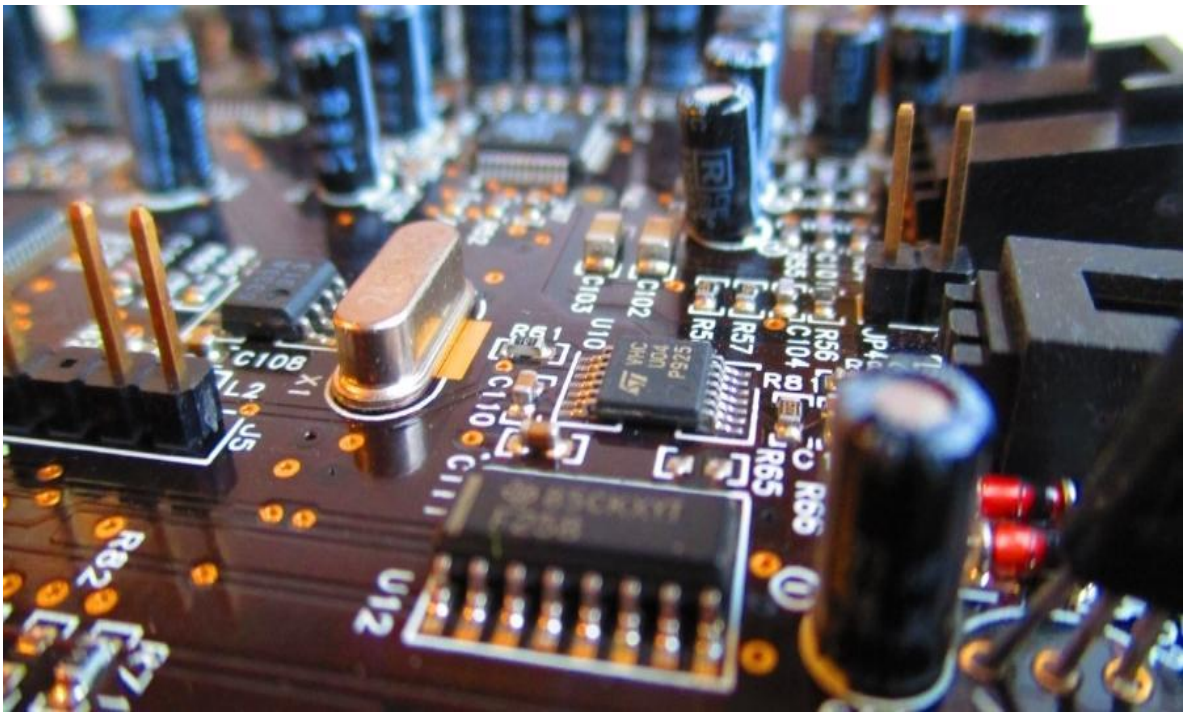
FACULTAD DE INGENIERÍA

CIUDAD UNIVERSITARIA



SISTEMAS OPERATIVOS

MAESTRO: GUNNAR EYAL WOLF ISZAEVICH

ALUMNO: YUKIOAYAX CANEK GABRIEL HERNÁNDEZ

BLOQUE: 136

GRUPO: 8

SEMESTRE 2026-1

Comparación de planificadores

En un sistema operativo la CPU es un recurso compartido y limitado, el planificador decide qué proceso corre y por cuánto tiempo, de esas decisiones dependen la capacidad de respuesta, el aprovechamiento del procesador y la equidad entre trabajos, ningún algoritmo es mejor en todo pues cada uno favorece objetivos distintos y se comporta de manera diferente según la carga, por eso el propósito de este trabajo es construir un simulador sencillo que permita contrastar bajo condiciones controladas varios mecanismos clásicos de planificación (FCFS, RR, SPN y MLFQ) y observar sus comportamientos.
La comparación no se hace con un único conjunto de procesos porque puede engañar, se generan varias cargas independientes con tiempos de llegada y de servicio distintos y para cada algoritmo se miden tres métricas estándar: tiempo de retorno promedio (T), tiempo de espera promedio (E) y penalización promedio (P=T/s), además, el simulador dibuja la secuencia de ejecución por ticks y marca con "-" los huecos de CPU ociosa, lo que facilita verificar a simple vista que la ejecución es consistente con la teoría, con esto se busca entender cómo cambian T, E y P cuando varía la carga y qué casos favorecen a cada planificador.
Se requiere un programa que permita comparar de forma justa y repetible varios planificadores de CPU en una sola máquina (FCFS, RR, SPN y MLFQ), el programa debe tomar conjuntos de procesos definidos por un identificador, un tiempo de llegada y un tiempo de servicio en ticks, simular con cada algoritmo exactamente la misma carga y producir para cada uno el esquema de ejecución por unidad de tiempo y los promedios de tres métricas, tiempo de retorno (T), tiempo de espera (E) y penalización (P=T/s), la simulación debe considerar huecos cuando la CPU queda ociosa antes de la llegada de algún proceso y reflejarlos explícitamente en la línea de tiempo con el carácter "-", porque esos huecos afectan T, E y P.
Se implementó un código en Java para resolver el ejercicio, para probarlo primero se necesita Java instalado, para esto se recomienda visitar el siguiente enlace (https://www.youtube.com/watch?v=Pzl41Iw4T04) una vez realizado esto, es necesario abrir la terminal y sitúate en la carpeta donde están los archivos .java (usa cd hasta esa carpeta), compila todo:

```
javac *.java
```

Debido a la solución dada, existen diversas formas de compilar y ejecutar el programa, por ende, se darán formas generales y una canónica (pues servirá para comprobar la salida como se estipuló en el repositorio), de tal forma que para comprobar dicha salida se debe poner en la terminal:

```
java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from
"A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"
```

O para varias rondas:

```
java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from
"A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from
"A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from
"A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"
```

En general, la forma general que compila el programa es el siguiente:

java Simulador --alg ALGOS [--rr QS] [--mlfq NIVELES [--mlfq-aging K] [--mlfq-no-preempt]] FUENTE

Donde --alg ALGOS indica la lista de algoritmos separados por comas y los valores validos son fcfs, spn, rr y mlfq; si aparece rr es obligatorio --rr QS donde QS es una lista de cuantums en ticks separados por comas, por ejemplo --rr 1,4; si aparece mlfq es obligatorio --mlfq NIVELES donde NIVELES es la lista de quantums por nivel separados por comas, por ejemplo --mlfq 1,2,4, y opcionalmente se puede anadir --mlfq-aging K para indicar el envejecimiento en ticks y --mlfq-no-preempt para desactivar la expropiacion dentro de cada nivel; finalmente FUENTE define de donde sale la carga y debe elegirse una sola opcion: --from "A:0,3;B:1,5;..." donde cada proceso se escribe como ID:llegada,servicio en ticks separados por ; (tambien se admite t=3 como alias de servicio), o --from-file archivo.txt donde cada linea del archivo es una ronda completa con el mismo formato anterior, o --random -n N [-s SEED] para generar N rondas aleatorias reproducibles con semilla y parametros adicionales del generador como --gen-n-min a --gen-n-max b para el numero de procesos por ronda, --gen-serv-min x --gen-serv-max y y --gen-serv-dist uniform|exp:mu para los servicios, --gen-arr-dist uniform|exp:lambda junto con --gen-gap G para espaciar llegadas o --gen-max-lleg M para fijar una llegada maxima; en todos los casos la misma carga de cada ronda se usa con todos los algoritmos y la salida muestra primero el encabezado con los procesos y el total de servicio y despues, por algoritmo, las metricas T, E, P seguidas de la traza por tick marcando con - la CPU ociosa.

Como se puede apreciar, es un poco tardío la forma en la que se ejecuta este programa, por ende, se deja a continuación comandos de compilación para que se prueben de forma más cómoda:

java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"

java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"

java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from-file rondas.txt

java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --random -n 5 -s 123 --gen-n-min 4 --gen-n-max 8 --gen-serv-min 1 --gen-serv-max 10 --gen-serv-dist uniform --gen-arr-dist uniform --gen-gap 6

java Simulador --alg fcfs,spn,rr,mlfq --rr 1,2,8 --mlfq 2,4,8 --mlfq-aging 20 --random -n 5 -s 7 --gen-n-min 5 --gen-n-max 9 --gen-serv-min 1 --gen-serv-max 12 --gen-serv-dist exp:4.5 --gen-arr-dist uniform --gen-gap 8

java Simulador --alg fcfs,spn,rr,mlfq --rr 2,6 --mlfq 2,4,8 --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"

java Simulador --alg fcfs,spn,rr,mlfq --rr 4 --mlfq 1,2,4 --mlfq-no-preempt --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"

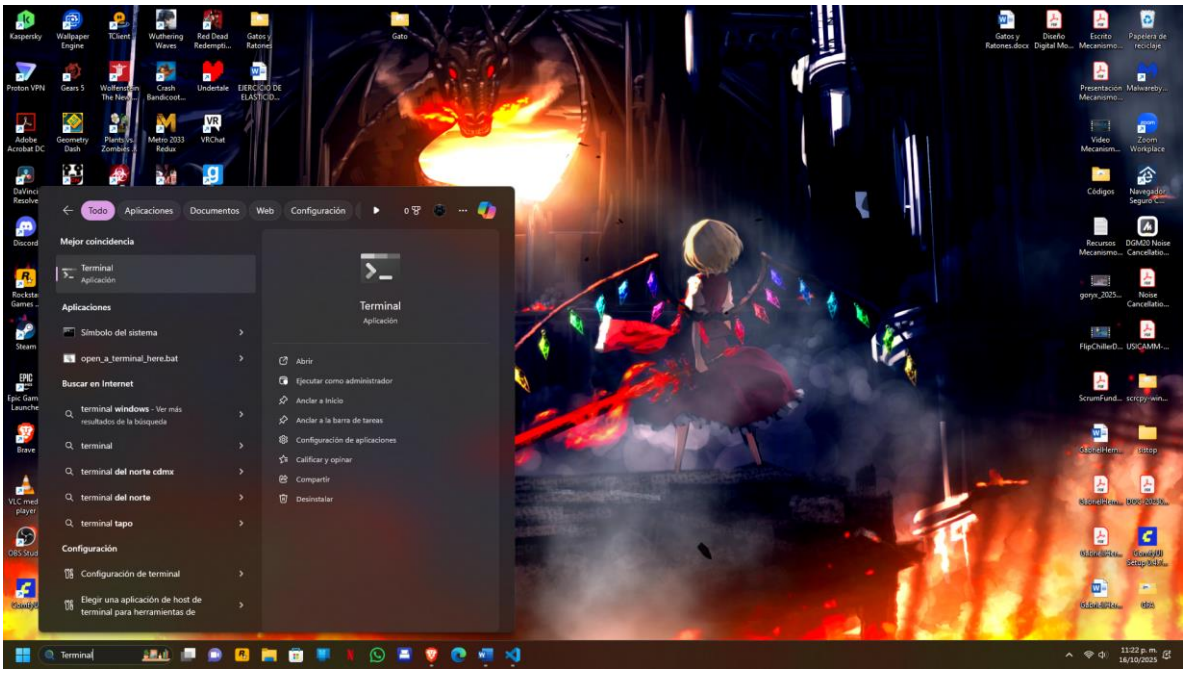
java Simulador --alg rr --rr 1,4,8 --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"

java Simulador --alg mlfq --mlfq 1,2,3,5 --mlfq-aging 25 --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"

java Simulador --alg spn --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"

java Simulador --alg fcfs --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"

Por último, para comandos como java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from-file rondas.txt es necesario tener un archivo de texto creado, puede ser usado el archivo ya creado o crear uno similar.



Ejecución de una ronda:

```
PS D:\Escritorio\Comparación de planificadores> javac *.java
PS D:\Escritorio\Comparación de planificadores> java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"
- Primera ronda:
  A: 0, t=3; B: 1, t=5; C: 3, t=2; D: 9, t=5; E: 12, t=5
      (tot:20)
FCFS:                           T=6.2, E=2.2, P=1.74
AAABBBBBCCDDDDDEEEEE
SPN:                            T=5.6, E=1.6, P=1.32
AAACCBBBBBDDDDDEEEEE
RR1:                            T=6.8, E=2.8, P=1.77
AABABCBCBBDDDEDEDEEE
RR4:                            T=7.2, E=3.2, P=1.88
AAABBBBCCBDDDDEEEEDE
MLFQ[q=1,2,4;aging=0;preempt=on]: T=6.6, E=2.6, P=1.73
ABAACBBCBBDDDEDDEEEE
```

Ejecución de varias rondas:

```
PS D:\Escritorio\Comparación de planificadores> java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5" --from "A:0,3;B:1,5;C:3,2;D:9,5;E:12,5"
- Primera ronda:
  A: 0, t=3; B: 1, t=5; C: 3, t=2; D: 9, t=5; E: 12, t=5
      (tot:20)
FCFS:                           T=6.2, E=2.2, P=1.74
AAABBBBBCCDDDDDEEEEE
SPN:                            T=5.6, E=1.6, P=1.32
AAACCBBBBBDDDDDEEEEE
RR1:                            T=6.8, E=2.8, P=1.77
AABABCBCBBDDDEDEDEEE
RR4:                            T=7.2, E=3.2, P=1.88
AAABBBBCCBDDDDEEEEDE
MLFQ[q=1,2,4;aging=0;preempt=on]: T=6.6, E=2.6, P=1.73
ABAACBBCBBDDDEDDEEEE

- Segunda ronda:
  A: 0, t=3; B: 1, t=5; C: 3, t=2; D: 9, t=5; E: 12, t=5
      (tot:20)
FCFS:                           T=6.2, E=2.2, P=1.74
AAABBBBBCCDDDDDEEEEE
SPN:                            T=5.6, E=1.6, P=1.32
AAACCBBBBBDDDDDEEEEE
RR1:                            T=6.8, E=2.8, P=1.77
AABABCBCBBDDDEDEDEEE
RR4:                            T=7.2, E=3.2, P=1.88
AAABBBBCCBDDDDEEEEDE
MLFQ[q=1,2,4;aging=0;preempt=on]: T=6.6, E=2.6, P=1.73
ABAACBBCBBDDDEDDEEEE
```

```
- Tercera ronda:
  A: 0, t=3; B: 1, t=5; C: 3, t=2; D: 9, t=5; E: 12, t=5
      (tot:20)
FCFS:                           T=6.2, E=2.2, P=1.74
AAABBBBBCCDDDDDEEEEE
SPN:                            T=5.6, E=1.6, P=1.32
AAACCBBBBBDDDDDEEEEE
RR1:                            T=6.8, E=2.8, P=1.77
AABABCBCBBDDDEDEDEEE
RR4:                            T=7.2, E=3.2, P=1.88
AAABBBBCCBDDDDEEEEDE
MLFQ[q=1,2,4;aging=0;preempt=on]: T=6.6, E=2.6, P=1.73
ABAACBBCBBDDDEDDEEEE

- Cuarta ronda:
  A: 0, t=3; B: 1, t=5; C: 3, t=2; D: 9, t=5; E: 12, t=5
      (tot:20)
FCFS:                           T=6.2, E=2.2, P=1.74
AAABBBBBCCDDDDDEEEEE
SPN:                            T=5.6, E=1.6, P=1.32
```

Compilación con archivo:

```
PS D:\Escritorio\Comparación de planificadores> java Simulador --alg fcfs,spn,rr,mlfq --rr 1,4 --mlfq 1,2,4 --from-file rondas.txt
- Primera ronda:
  A: 0, t=3; B: 1, t=5; C: 3, t=2; D: 9, t=5; E: 12, t=5
      (tot:20)
FCFS:                           T=6.2, E=2.2, P=1.74
AAABBBBBCCDDDDDEEEEE
SPN:                            T=5.6, E=1.6, P=1.32
AAACCBBBBBDDDDDEEEEE
RR1:                            T=6.8, E=2.8, P=1.77
AABABCBCBBDDDEDEDEEE
RR4:                            T=7.2, E=3.2, P=1.88
AAABBBBCCBDDDDEEEEDE
MLFQ[q=1,2,4;aging=0;preempt=on]: T=6.6, E=2.6, P=1.73
ABAACBBCBBDDDEDDEEEE

- Segunda ronda:
  A: 0, t=5; B: 3, t=3; C: 3, t=7; D: 7, t=4; E: 8, t=4
      (tot:23)
FCFS:                           T=9.8, E=5.2, P=2.23
AAAAABBBCCCCCCCDDDDEEEE
SPN:                            T=8.6, E=4.0, P=1.75
AAAAABBBDDDDEEEECCCCCCC
RR1:                            T=12.2, E=7.6, P=2.68
AAAABCABCDBECDECDECDECC
RR4:                            T=11.4, E=6.8, P=2.37
AAAABBBCCCCADDDDEEEECCC
MLFQ[q=1,2,4;aging=0;preempt=on]: T=13.8, E=9.2, P=2.96
AAABCBBDECCDDEEAACCCCDE

- Tercera ronda:
  A: 0, t=4; B: 2, t=6; C: 5, t=3
      (tot:13)
FCFS:                           T=6.7, E=2.3, P=1.67
AAAABBBBBBCCC
SPN:                            T=6.7, E=2.3, P=1.67
AAAABBBBBBCCC
RR1:                            T=7.3, E=3.0, P=1.69
AAABABCBCBCBB
RR4:                            T=7.0, E=2.7, P=1.61
AAAABBBBCCCBB
MLFQ[q=1,2,4;aging=0;preempt=on]: T=7.0, E=2.7, P=1.58
AAABACBBCCBBB
```

Puede probar las diversas combinaciones dadas o intentar crear un comando para una ejecución personalizada.

En conclusión, la solución cumple el objetivo porque compara de forma justa y reproducible los planificadores sobre la misma carga, calcula y reporta con claridad T, E y P, y además muestra la traza por tick incluyendo los huecos “-” de modo que cualquiera puede verificar a simple vista los tiempos de finalización y por tanto las métricas, el simulador no es una demo rígida, es parametrizable por línea de comandos (quantums de RR, niveles y política de MLFQ, fuente de la carga) y admite tanto cargas escritas a mano como desde archivo o generadas con semilla, lo que permite repetir exactamente los resultados o explorar variaciones cuando se necesite, el formato de salida coincide con lo pedido y evita valores predeterminados que sesguen la comparación, así que la evidencia que produce es consistente con la teoría y suficiente para observar los compromisos de cada algoritmo, por todo

eso, este programa resuelve el planteamiento, ofrece una base clara, verificable y extensible para analizar y contrastar planificadores de CPU.

Bibliografía

baeldung. (noviembre 27, 2017). *Introduction to the Java ArrayDeque*. Baeldung. Recuperado de https://www.baeldung.com/java-array-deque

baeldung. (enero 8, 2024). *A Guide To Java Regular Expressions API*. Baeldung. Recuperado de https://www.baeldung.com/regular-expressions-java

baeldung. (enero 8, 2024). *How to Read a File in Java*. Baeldung. Recuperado de https://www.baeldung.com/reading-file-in-java

baeldung. (marzo 17, 2024). *Guide to ThreadLocalRandom in Java*. Baeldung. Recuperado de https://www.baeldung.com/java-thread-local-random

Piazzolla, G. (enero 8 2024). *Guide to Java PriorityQueue*. Baeldung. Recuperado de https://www.baeldung.com/java-priorityqueue

Pradhan, P. (febrero 20, 2025). *Intro to the Apache Commons CLI*. Baeldung. Recuperado de https://www.baeldung.com/apache-commons-cli

S.A. (2020). *PLANIFICACIÓN DE PROCESOS.* DEPARTAMENTO DE CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN, UNIVERSIDAD NACIONAL DEL SUR. Recuperado de https://cs.uns.edu.ar/~so/data/apuntes/SO-2020-mod%2007.pdf

S.A. (s.f.). *JDK 25 Documentation*. ORACLE. Recuperado el 24 de octubre de 2025 de https://docs.oracle.com/en/java/javase/25/

S.A. (s.f.). *Scheduling: Introduction*. OSTEP. Recuperado el 24 de octubre de 2025 de https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf