

Universidad Tecnológica Nacional  
Facultad Regional Resistencia  
Ingeniería en Sistemas de Información

## Sistemas Operativos

# **Simulador de Asignación de Memoria y Planificación de Procesos**

### **Equipo: Los Sims**

Andrés, Aldo Omar.

Bravo Pérez, Agustín Nicolás.

Peralta Ruiz, Nadine.

Trespalacios Martinez, Carlos Daniel.

Vazquez, Máximo Ezequiel.

21 de Noviembre de 2023

Ciclo 2023, Segundo Cuatrimestre

Resistencia, Chaco, Argentina

# Índice

<b>Introducción .....</b>	<b>1</b>
<b>Aspectos Técnicos .....</b>	<b>1</b>
Características del Enunciado .....	1
Decisiones de Diseño .....	1
<b>Diseño .....</b>	<b>2</b>
<b>Implementación .....</b>	<b>6</b>
Organización del Equipo .....	6
Control de Versiones .....	6
Estructura .....	7
Pruebas .....	8
<b>Uso .....</b>	<b>8</b>
<b>Conclusiones .....</b>	<b>13</b>

# Introducción

Desarrollamos este simulador de manera colaborativa y asincrónica, dividiéndonos el trabajo en equipo, y cada integrante tuvo un rol a cumplir.



Figura 1: Logo del Grupo “Los Sims”

## Aspectos Técnicos

### Características del Enunciado

- El planificador a corto plazo usa la estrategia round robin con un quantum igual a 2.
- El esquema de memoria tiene una partición exclusiva para el sistema operativo y otras tres particiones fijas:
  - 250 kB para trabajos grandes,
  - 120 kB para trabajos medianos,
  - y 60 kB para trabajos pequeños.
- La política de asignación a dicha memoria es best-fit.
- El simulador acepta una carga de trabajo con hasta 10 procesos.
- El simulador soporta un grado de multiprogramación igual a 5 como máximo, pero la memoria principal solo permite 3 procesos a la vez, por lo que se necesita usar memoria virtual.

### Decisiones de Diseño

- Al momento de activar un proceso suspendido, se lo activa siempre en la partición de mismo tamaño que tenía previamente al ser suspendido.

- Al momento de elegir una partición víctima para suspender, se elige siempre la partición de mismo tamaño que la partición que se está trayendo a memoria principal.
- La planificación round robin tiene en cuenta a los procesos listos y listos/suspendidos por igual. Si hubiera 3 procesos listos y 2 listos/suspendidos, el CPU se estaría compartiendo entre ellos 5, haciendo swap in y out para que los listos/suspendidos estén en memoria principal cuando les llegue su turno.
- Si un proceso desea ser admitido una vez alcanzado el grado de multiprogramación máximo (5), es rechazado pero se lo vuelve a intentar admitir en cada delta de tiempo hasta que otro proceso termine.
- Creamos un estado especial DENEGADO para los procesos que requieren una cantidad de memoria mayor al tamaño de la partición de memoria más grande, por lo que jamás podría ser admitido. Se lo envía al estado DENEGADO para siempre.

En particular el hecho de tener a los procesos listos y listos/suspendidos en la misma cola con la misma prioridad, asegura que un grado de multiprogramación igual a cinco sea posible, pero sacrifica el rendimiento del sistema dado que cuando el grado de multiprogramación es mayor a tres la cantidad de swap outs es muy dañina para la velocidad.

Esto pone en evidencia el compromiso entre la velocidad y la capacidad del sistema, siendo que sostener un grado de multiprogramación mayor a la cantidad de particiones, especialmente en un simulador que no tiene procesos que se bloqueen, se vuelve muy costoso debido al impacto de los swap outs.

## Diseño

Al momento de realizar los primeros bocetos de cómo implementaríamos la consigna del simulador, elaboramos el siguiente diagrama de flujo, en el que se puede ver cómo la carga de trabajo es ingresada mediante un archivo de entrada, siendo los procesos ordenados por tiempo de arribo. Luego, a medida que se van admitiendo los procesos a memoria, se planifica su uso del CPU mediante un planificador Round Robin con un quantum igual a 2 hasta que se termine la carga completa de trabajo. Una vez finalizada la simulación, se imprime por pantalla un reporte estadístico de los tiempos de espera y de retorno de cada proceso, y el promedio total.

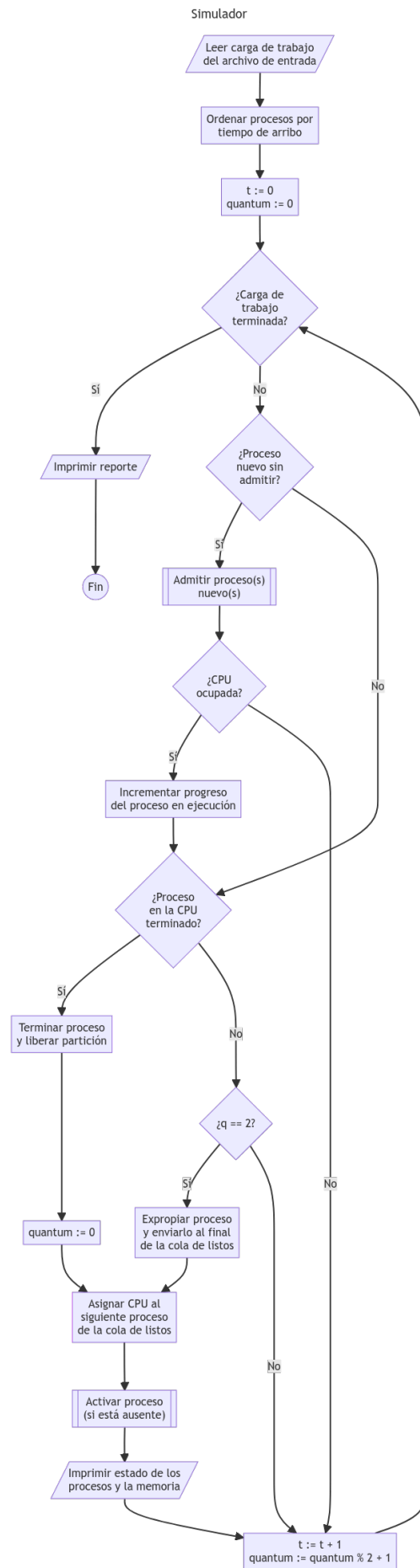


Figura 2: Diagrama del simulador.

Explayamos el proceso de asignar procesos nuevos y activar procesos en otros dos diagramas:

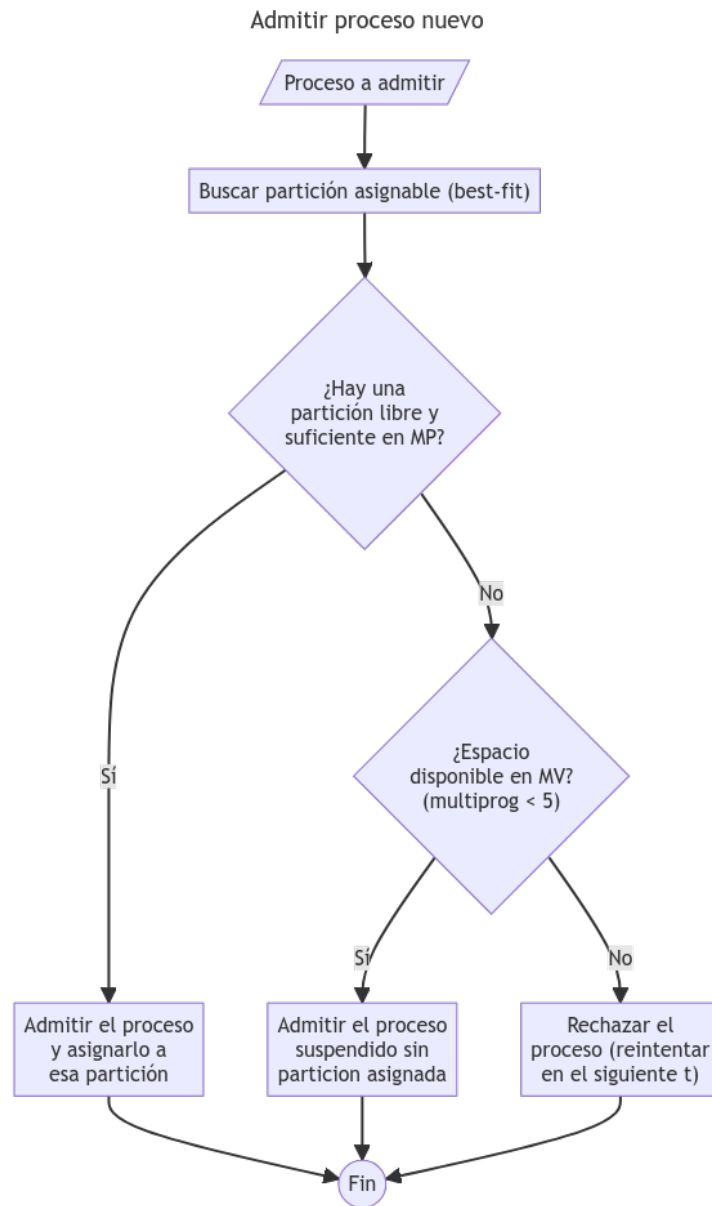


Figura 3: Diagrama de la admisión de un proceso nuevo.

Para poder admitir un proceso, debe haber una partición libre y con suficiente capacidad para alojarlo, caso contrario, si no se tiene un grado de multiprogramación superior a 5 el proceso será suspendido sin partición asignada. Si el grado de multiprogramación es mayor a 5 entonces se rechaza el proceso, y luego se volverá a intentar admitirlo.

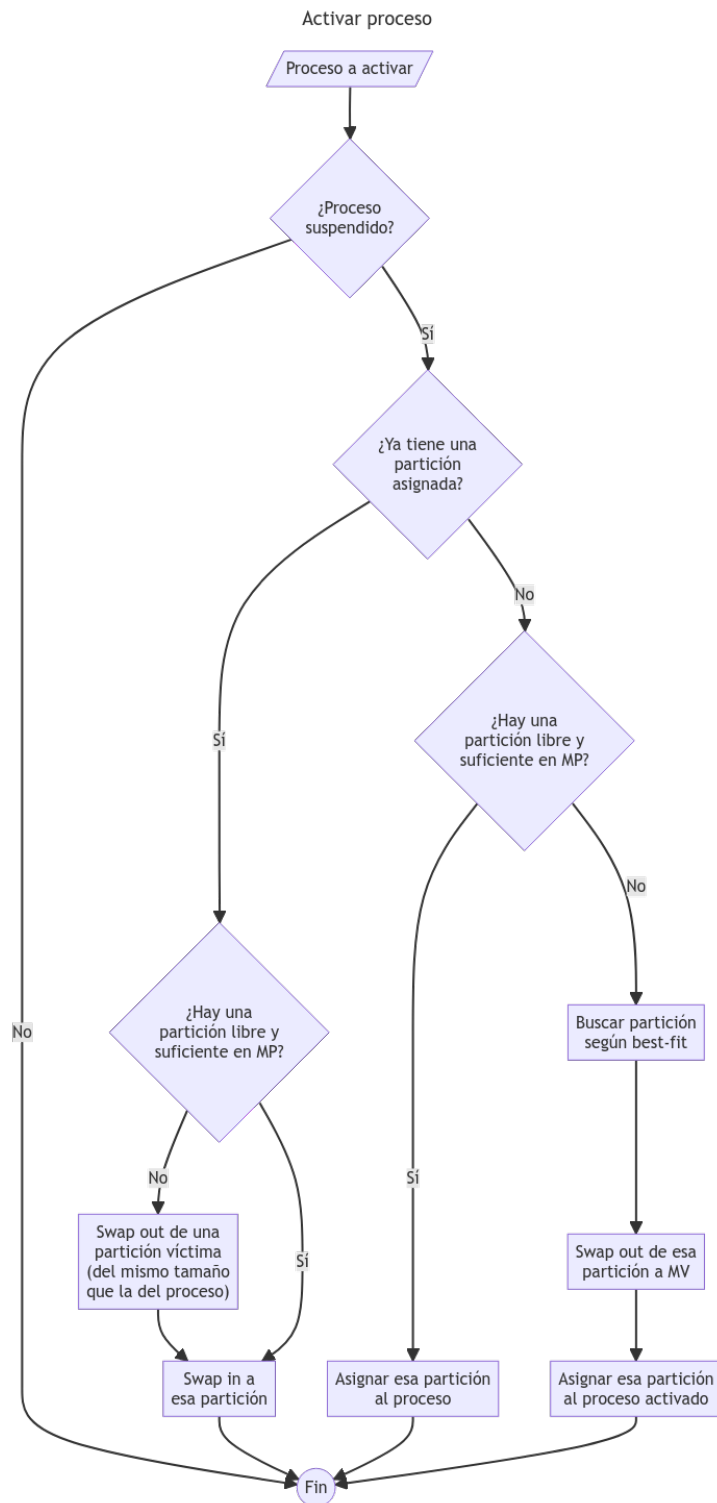


Figura 4: Diagrama de la activación de un proceso.

Un proceso debe ser activado cuando es asignado la CPU. Si está en memoria principal no es necesario hacer nada, pero si este se encuentra en la memoria virtual se debe primero verificar que exista una partición libre a la que pueda asignarse dicho proceso. De ser así, se hace un swap in a la misma. Sino, se debe hacer un swap out de alguna partición víctima antes de realizar el swap in. La partición víctima siempre es la partición de mismo tamaño (misma dirección de inicio) que de la partición suspendida que

se está trayendo a memoria principal. Si además ese proceso no tenía una partición asignada, dado que ws un proceso suspendido entrando por primera vez a la CPU, entonces se le asigna la partición encontrada.

## Implementación

### Organización del Equipo

Utilizamos un tablero kanban de la plataforma de Trello para organizarnos de una mejor manera en la división de responsabilidades y de avances del trabajo. Esto nos permitía ver en todo momento el estado en el que se encontraba el progreso de nuestro trabajo.

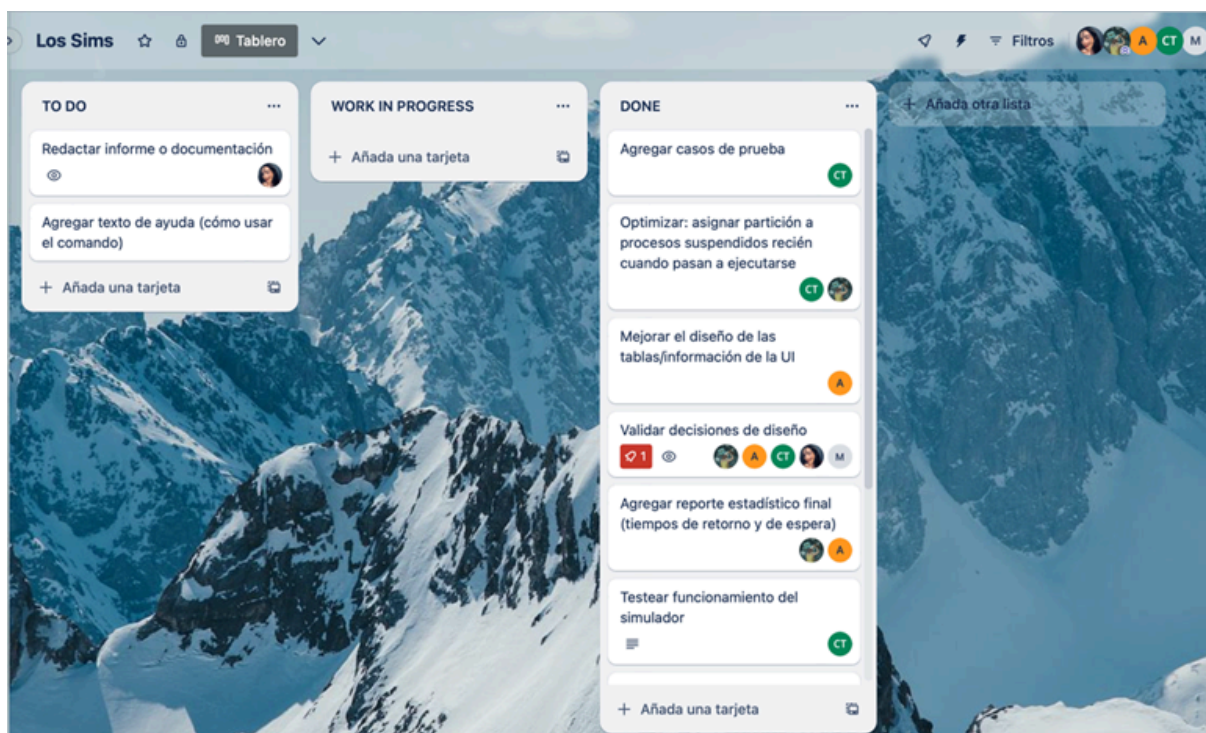


Figura 5: Tablero kanban online en Trello del equipo.

### Control de Versiones

Para facilitar el desarrollo colaborativo del trabajo usamos a GitHub como repositorio remoto, para tener un mejor detalle de los cambios que se fueron realizando en cada momento, con acceso a versiones anteriores en caso de hubiésemos cometido algún error.



```

* 02aebd2 - (HEAD -> main, origin/main, origin/HEAD) actualizados diagramas en base a la optimización de procesos suspendidos
Agustín Bravo Pérez>
* f1d657e - workload fuera de carpeta de procesos eliminado (27 hours ago) <Carlos Daniel>
* 764bd5d - Carpeta para la carga de trabajos (27 hours ago) <Carlos Daniel>
* 132691d - Se admiten procesos a suspendidos sin asignarles memoria antes de ser necesario (2 days ago) <Carlos Daniel>
* a95f0cd - corregidas comillas anidadas (5 days ago) <Aldo>
* 7ac7d12 - tabla de memoria ahora muestra valores en bytes (en lugar de bits) (6 days ago) <Agustín Bravo Pérez>
* 4722a76 - agregada barra de progreso a la tabla de la carga de trabajo (6 days ago) <Agustín Bravo Pérez>
* ddf37d - agregados colores a la información por pantalla (7 days ago) <Agustín Bravo Pérez>
* 103f731 - Decisión de crear el estado 'DENEGADO' documentada (8 days ago) <Agustín Bravo Pérez>
* 7cb3dea - Merge pull request #1 from agustinbravo/estadistico (8 days ago) <Agustín Bravo Pérez>
//
* fc5e1d9 - (estadistico) mejorada la información y formato del reporte estadístico final (8 days ago) <Agustín Bravo Pérez>
* 0f5a207 - Cálculo estadístico (2 weeks ago) <Aldo>
//
* 4d4cc0b - agregado logo del equipo al README.md (5 weeks ago) <Agustín Bravo Pérez>
* 9071c5c - aclaración de aspectos técnicos y decisiones de diseño (5 weeks ago) <Agustín Bravo Pérez>
* 40a2df1 - main.py modularizado en un archivo distinto para cada clase (6 weeks ago) <Agustín Bravo Pérez>
* f408ea8 - Merge branch 'main' of https://github.com/agustinbravo/utn-sims (6 weeks ago) <Agustín Bravo Pérez>
//
* 6147614 - Update main.py (6 weeks ago) <peraltanadine>
* 01829d1 - agregada memoria virtual (y simulador funcionando bien) (6 weeks ago) <Agustín Bravo Pérez>
* 712deeb - eliminado simulador.py (integrado a main.py) (6 weeks ago) <Agustín Bravo Pérez>
* 77f8d1f - renombrado simulador2.0 a simulador.py (6 weeks ago) <Agustín Bravo Pérez>
//
* 72a1114 - implementacion_de_la_carga_de_trabajo (6 weeks ago) <Carlos Daniel>
* b8e118b - implementacion_de_la_carga_de_trabajo (6 weeks ago) <Carlos Daniel>
* 77f75ea - Update simulador2.0 (6 weeks ago) <Maximo-Vazquez>
* 2d3b676 - Merge branch 'main' of https://github.com/agustinbravo/utn-sims (6 weeks ago) <Agustín Bravo Pérez>
//
* 1d3ed61 - Create simulador2.0 (7 weeks ago) <Maximo-Vazquez>
* 5e92487 - diagrama de flujo inicial (6 weeks ago) <Agustín Bravo Pérez>
//
* 4fcae9 - leer la carga de trabajo desde un archivo csv (7 weeks ago) <Agustín Bravo Pérez>
* e52b690 - Update README.md (8 weeks ago) <Agustín Bravo Pérez>
* 211d015 - Initial commit (8 weeks ago) <Agustín Bravo Pérez>

```

Figura 6: Historial de commits del repositorio.

## Estructura

Modelamos el simulador en cuatro clases de la siguiente manera:

### Modelo de Clases del Simulador

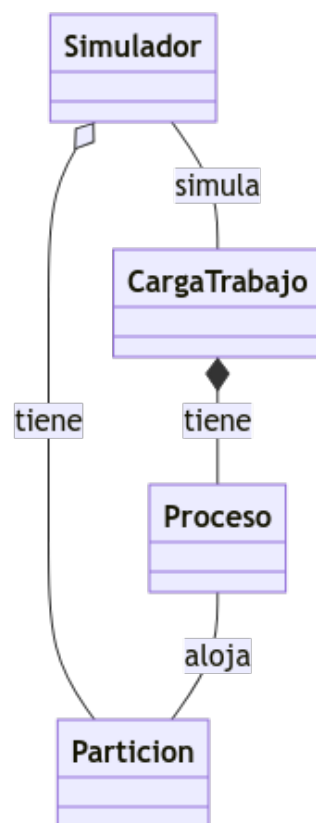


Figura 7: Diagrama de clases.

A grandes rasgos, la `CargaTrabajo` es un conjunto de `Procesos` tomados del archivo de entrada, que serán simulados por la clase `Simulador`, la cual representa su memoria de particiones fijas con un arreglo de `Particiones`, en la cual aloja los procesos para ser ejecutados.

La clase `Simulador` es la encargada de interconectar el resto de las clases, de implementar la planificación de la CPU y de la administración de la memoria. Tenemos además un archivo `main.py` que sirve como punto de entrada de la aplicación, y que va avanzando el simulador a medida que el usuario presiona `Enter` por consola.

El simulador fue desarrollado en **Python**, un lenguaje interpretado fácil de usar. Sus únicas dos dependencias son `colorama`, una librería para imprimir texto colorido por pantalla, y `tabulate`, para mostrar tablas con formato.

## Pruebas

Tenemos en la carpeta `/workloads` del proyecto cuatro cargas de trabajo con las cuales probamos distintas funcionalidades del simulador:

- **livianos.csv**: es la carga de trabajo con ocho procesos que se usó en un ejercicio de las guías prácticas del cursado. Es estándar y no tiene ningún caso especial o particular.
- **muchos.csv**: prueba el funcionamiento del simulador bajo una carga de diez procesos, el máximo que debe soportar.
- **pesados.csv**: son siete procesos de los cuales el número siete requiere 258 kB de memoria, por lo que será denegado y nunca podrá ser admitido a una partición. Prueba que el simulador termine incluso cuando no puede admitir todos los procesos.
- **admitido\_sin\_part.csv**: es una pequeña carga de trabajo que prueba la optimización realizada de asignar una partición de memoria a un proceso suspendido recién cuando va a ocupar la CPU por primera vez. Lo que se puede ver es que al proceso cuatro, que entra suspendido a la cola, al llegar a la CPU le termina asignando una partición distinta (más grande) a la que le hubiera asignado si lo alojaba ni bien que era admitido, lo cual termina ahorrando unos swap outs. Esto es un ejemplo de la optimización mejorando la performance del simulador, pero es una mejora situacional, ocurriendo solo en estos casos.

## Uso

Dado que el simulador está escrito en Python, siendo `main.py` el archivo de entrada, para ejecutarlo solamente es necesario tener un intérprete de Python.

Con el comando `python main.py` llamamos al programa, pero obtenemos el siguiente error:

```

anbra@LAPTOP-D9C4NLLH MINGW64 ~/Desktop/UTN/2doNivel/Sistemas Operativos/utn-sims (main)
$ python main.py
Falta el segundo parámetro que es el archivo con la carga de trabajo.
La carga de trabajo debe ser un CSV con las columnas 'PID;TI;TAN;Mem(kB)'.
Ej:  1;0;5;15
     2;0;4;20
     3;0;10;12
     4;1;3;5
NOTA: Procesos que requieran más de 250 kB de memoria serán denegados.

anbra@LAPTOP-D9C4NLLH MINGW64 ~/Desktop/UTN/2doNivel/Sistemas Operativos/utn-sims (main)
$ █

```

Figura 8: Texto de ayuda en la *command line interface*.

Esto se debe a que el simulador requiere un archivo de entrada que especifique la carga de trabajo a simular. Si ahora ejecutamos `python main.py workloads/livianos.csv`:

Antes de comenzar la simulación, vemos la carga de trabajo inicial:

```

anbra@LAPTOP-D9C4NLLH MINGW64 ~/Desktop/UTN/2doNivel/Sistemas Operativos/utn-sims (main)
$ python main.py workloads/livianos.csv
Carga de trabajo:

```

PID	TA	TI	Mem (kB)	Estado	Progreso (%)
1	0	5	15	Nuevo	_____ 0.0
2	0	4	20	Nuevo	_____ 0.0
3	0	10	12	Nuevo	_____ 0.0
4	1	3	5	Nuevo	_____ 0.0
5	2	2	3	Nuevo	_____ 0.0
6	3	10	70	Nuevo	_____ 0.0
7	4	5	25	Nuevo	_____ 0.0
8	5	5	10	Nuevo	_____ 0.0

```

Presione q + Enter para terminar el programa.
Presione Enter para comenzar la simulación... █

```

Figura 9: Carga de trabajo inicial.

Podemos presionar `q + Enter` en cualquier momento para salir, o `Enter` para comenzar o avanzar la simulación:

Estado del procesador y de la cola de listos: [CPU: P1] <-- P2 <-- P3  
(t = 0; quantum = 0)

Tabla de memoria: (valores en bytes)

Partición	Presente	Proceso	Dir. Inicio	Tamaño	Mem. en uso	Frag. Interna
1	Sí	P3	102400	256000	12288	243712
2	Sí	P2	358400	122880	20480	102400
3	Sí	P1	481280	61440	15360	46080

Carga de trabajo: (grado de multiprogramación = 3)

PID	TA	TI	Mem (kB)	Estado	Progreso (%)
1	0	5	15	Ejecutando	_____ 0.0
2	0	4	20	Listo	_____ 0.0
3	0	10	12	Listo	_____ 0.0
4	1	3	5	Nuevo	_____ 0.0
5	2	2	3	Nuevo	_____ 0.0
6	3	10	70	Nuevo	_____ 0.0
7	4	5	25	Nuevo	_____ 0.0
8	5	5	10	Nuevo	_____ 0.0

Presione Enter para avanzar o q + Enter para salir...█

Figura 10: Primer instante de tiempo del simulador.

Vemos que en el instante cero se admiten los tres procesos con tiempo de arribo igual a cero, que acaban de llegar a la cola de listos. Si avanzamos unos cuantos pasos más...

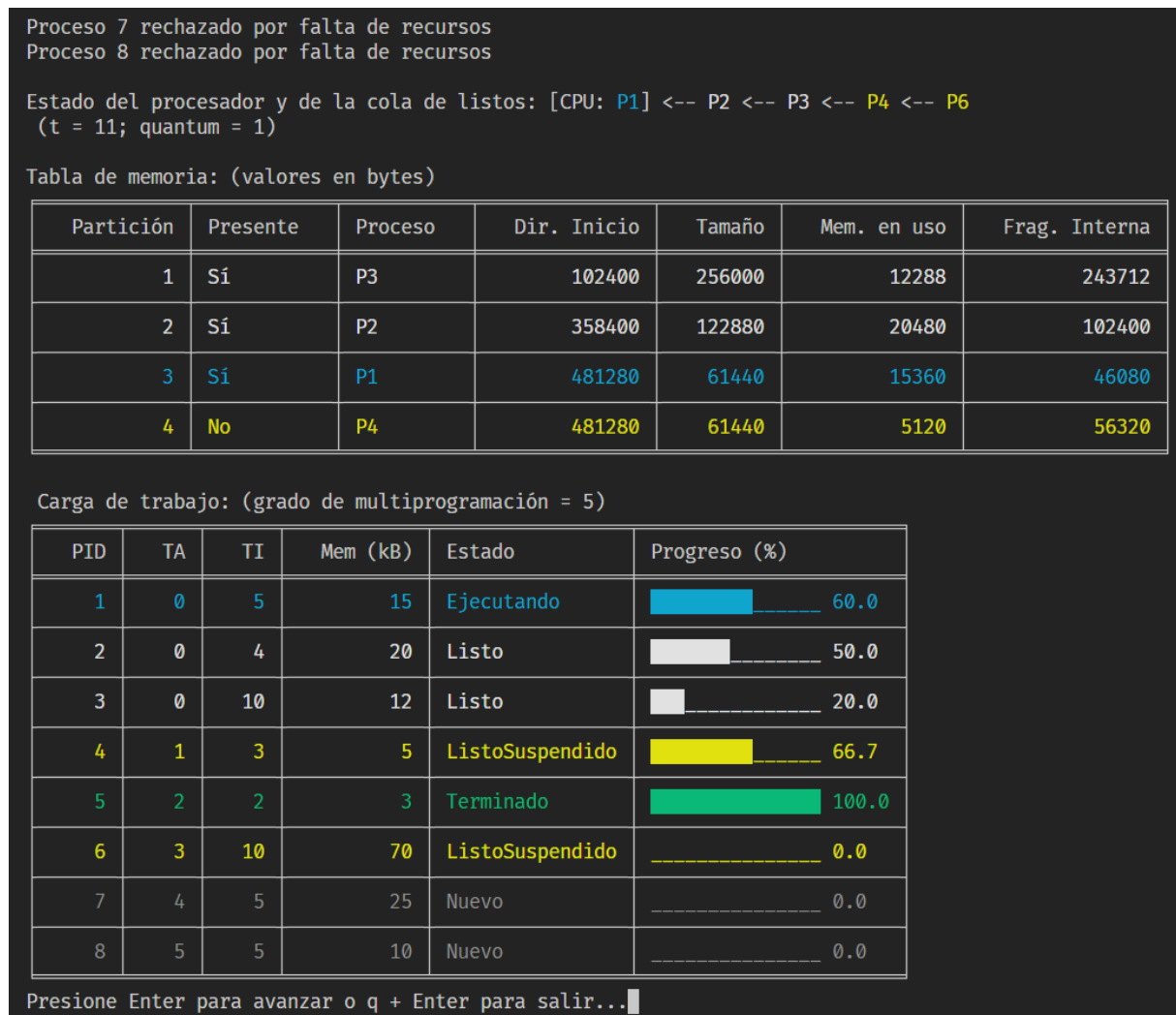


Figura 11: Progreso de la simulación en el instante t = 11.

Lo primero que vemos es que la CPU está siendo usada por el proceso P1, y la cola de listos que tiene a los procesos P2, P3, P4, P6. También se indica que estamos en el instante 11 con un quantum igual a 1.

Podemos ver en color blanco los procesos listos, en color azul el proceso en ejecución, en verde los terminados y en amarillo los suspendidos. Vemos que los procesos 7 y 8 fueron rechazados por ya haber alcanzado el grado de multiprogramación igual a 5, así que siguen en estado nuevo (color gris).

Observamos además la tabla de memoria con las tres particiones fijas de la memoria principal, junto a una partición virtualizada, la cual está en amarillo porque aloja al proceso suspendido.

Si continuamos hasta terminar toda la carga de trabajo:

Estado del procesador y de la cola de listos: (vacía)  
(t = 44; quantum = 0)

Tabla de memoria: (valores en bytes)

Partición	Presente	Proceso	Dir. Inicio	Tamaño	Mem. en uso	Frag. Interna
1	Sí	-	102400	256000	0	0
2	Sí	-	358400	122880	0	0
3	Sí	-	481280	61440	0	0

Carga de trabajo: (grado de multiprogramación = 0)

PID	TA	TI	Mem (kB)	Estado	Progreso (%)
1	0	5	15	Terminado	<div style="width: 100%;">100.0</div>
2	0	4	20	Terminado	<div style="width: 100%;">100.0</div>
3	0	10	12	Terminado	<div style="width: 100%;">100.0</div>
4	1	3	5	Terminado	<div style="width: 100%;">100.0</div>
5	2	2	3	Terminado	<div style="width: 100%;">100.0</div>
6	3	10	70	Terminado	<div style="width: 100%;">100.0</div>
7	4	5	25	Terminado	<div style="width: 100%;">100.0</div>
8	5	5	10	Terminado	<div style="width: 100%;">100.0</div>

Presione Enter para avanzar o q + Enter para salir...

Figura 12: Carga de trabajo terminada.

La simulación llega a su fin, imprimiendo un reporte estadístico con los tiempos de espera y tiempos de retorno. Luego la aplicación termina.

```
===== ¡Simulación terminada! =====
```

Reporte estadístico final:

Proceso	TA	TI	T. Espera	T. Retorno
1	0	5	15	20
2	0	4	10	14
3	0	10	29	39
4	1	3	14	17
5	2	2	7	9
6	3	10	24	34
7	4	5	18	23
8	5	5	18	23

Instante de tiempo final: 44  
Tiempo de espera promedio: 16.875  
Tiempo de retorno promedio: 22.375

Figura 13: Informe estadístico final del simulador.

## Conclusiones

Finalizada la experimentación del desarrollo del simulador, logramos enlazar los conceptos de teoría en relación a la administración de memoria y planificación de la CPU, con una aplicación tangible de las mismas, gracias a haber diseñado la administración las particiones de la memoria, la asignación de procesos, y el manejo de la cola de procesos listos.

Una oportunidad de optimización que encontramos fue asignar la partición de memoria a los procesos suspendidos recién cuando vayan a ejecutarse. Originalmente el simulador les asignaba memoria ni bien eran admitidos, lo cual era innecesario. Este cambio reduce la cantidad de memoria virtualizada, pero solo ayuda a disminuir la cantidad de swap outs (y por ende mejorar la velocidad) cuando el haber esperado a asignar la partición efectivamente implica asignar una partición de un tamaño que cumpla el algoritmo best-fit.

Rescatamos la importancia de probar con distintas cargas de trabajo el funcionamiento del simulador, y en general de cualquier sistema. Gracias a estas pruebas es que encontramos oportunidades de mejora, y validamos la efectividad de las optimizaciones realizadas.

Algo que nos benefició mucho al momento de la realización de este trabajo, fue el uso de Trello porque de esta manera pudimos llevar un control real del estado en el que nos encontrábamos. Esto también nos ayudó en el trabajo en equipo, porque de esta manera cada integrante del equipo sabía que tarea debía realizar para poder avanzar de manera organizada. Es por esta razón que seguiremos utilizando esta herramienta para futuros trabajos.