UCU Universidad Católica del Uruguay

# Sistemas Operativos

# Primer semestre 2022

## Trabajo Obligatorio



Sistema de deliveries


Rodrigo Torres

Fabricio Benitez

Bruno Arnuti

# Marco teórico

**Proceso**

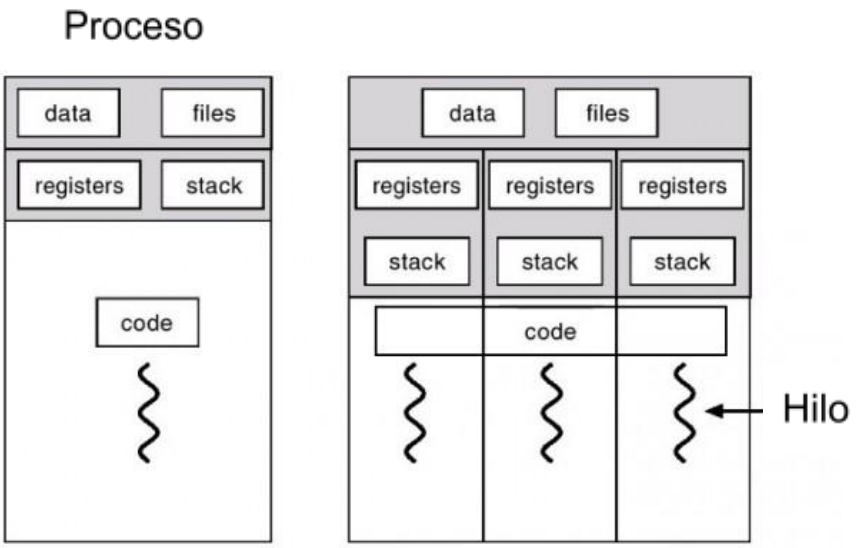
La abstracción proporcionada por el sistema operativo de un programa en ejecución es algo llamaremos un proceso. Como dijimos anteriormente, un proceso es simplemente una ejecución de un programa en cualquier instante de tiempo, podemos resumir un proceso tomando una inventario de las diferentes piezas del sistema al que accede o afecta durante el curso de su ejecución.

El proceso puede ser descrito por su estado: el contenido en la memoria en su espacio de direcciones, el contenido de los registros de la CPU (incluido el contador de programa y el puntero de pila, entre otros), e información sobre E/S (como archivos abiertos que se pueden leer o escribir).

**Hilo**

Un hilo podría definirse como una unidad básica de utilización de CPU. Contiene el "id" del hilo, su propio program counter, un conjunto de registros y una pila. La principal razón para programar en múltiples hilos es que habilita al programa a realizar varias actividades a la vez, trabajando en "cuasi-paralelo" teniendo en cuenta que se tenga un único procesador.

En la siguiente imagen podemos observar las principales diferencias entre hilo y proceso, los procesos son generalmente independientes, contienen su estado como se mencionó antes y se comunican a través de mensajes. Por otro lado, los hilos se crean dentro de un proceso, por lo que comparten la memoria dentro de ese proceso, y también el código.



**Condiciones de Carrera**

Las condiciones de carrera son situaciones en donde múltiples procesos interactúan con un mismo dato compartido, ya sea de lectura o escritura, y el resultado dependerá de quién lo ejecute y cuando exactamente lo haga.

**Regiones críticas**

Las regiones críticas nacen como una solución para el problema de las condiciones de carrera. La idea principal es la de evitar que más de un proceso interactúe con un dato compartido a la vez. Cuando un proceso entre en su zona crítica, es decir que está trabajando con una variable o un archivo en específico, los demás procesos en ejecución se "excluirán de hacer lo mismo" (exclusión mutua).

Para evitar estos problemas, los hilos (y procesos) deben utilizar algún tipo de exclusión mutua, primitivas de exclusión; hacerlo garantiza que solo un único hilo alguna vez entra en una sección crítica, evitando carreras, y resultando en productos deterministas del programa.

**Test-Set-Lock**

También conocida por las siglas "TSL", es una instrucción de hardware ideada para facilitar la creación de semáforos y demás herramientas para posibilitar la programación concurrente. TSL es una instrucción máquina para la lectura del contenido de una palabra de la memoria en un registro, y para el almacenamiento de un valor distinto de 0 en dicha palabra de memoria. Sumado a esto, en un sistema multiprocesador, ninguno de los demás procesadores tendrá acceso a la palabra hasta que termine la instrucción. "La CPU que ejecuta la instrucción TSL bloquea el bus de memoria para impedir que otras CPUs accedan a la memoria hasta que termine."

**Semáforos**

Los semáforos son una herramienta o mecanismo de sincronización de procesos, que permiten al programador influir y asistir al sistema operativo en su toma de decisiones, así ayudando a sincronizar la ejecución de múltiples procesos. Un semáforo es un objeto con un valor entero que podemos manipular con dos rutinas.
Puede tener valor de 0, si aún no se guardaron señales de despertar o un valor positivo si estuvieran pendientes una o más señales de despertar. Una de dos operaciones, *"down"*, comprueba si este valor es mayor que 0. En caso afirmativo, disminuye su valor, o en otras palabras consume una señal de despertar, y continua. En caso de que el valor fuera 0, el proceso se pone a "dormir" sin completar la operación, por el momento. Por otro lado, la operación *"up"* incrementa el valor. Si hubiera procesos en ese semáforo, esperando a completar una operación down anterior, el sistema selecciona a uno de ellos y le permite continuar.

**Mutex**

El mutex es, por decirlo de alguna forma, una versión simplificada de los semáforos, sin la habilidad de "contar". Se utilizan para administrar la exclusión mutua para cierto recurso compartido. Se conforman por dos procedimientos: *mutex_lock* y *mutex_unlock*; En el primero, el hilo solicita entrar a su sección crítica. Si el mutex se encuentra abierto, se le concederá el acceso. En caso contrario, el hilo se bloqueará hasta que el responsable de haber cerrado el Mutex salga de

la sección crítica y lo vuelva a abrir utilizando el segundo procedimiento. El principal beneficio de utilizar Mutex, es que en este no existe la "espera ocupada", situación en la que un hilo queda en espera, mientras reevalúa constantemente si se le permite continuar. Cuando *mutex_lock* no puede adquirir el mutex, entrega el procesador a otro hilo, sin reevaluar la condición de espera sino hasta que vuelve a ocupar el procesador.

**Planificador de procesos**

El planificador de procesos es aquel que, teniendo varios procesos a la espera de ejecutarse, compitiendo todos por un mismo procesador, elige cuál de ellos será el siguiente en ejecutarse. Existen una gran variedad de formas de ordenar o de elegir cuál proceso se ejecutará antes de otro, cada una de ellas considerando diferentes aspectos.

Antes de entrar en el rango de posibles políticas, es importante saber que: Determinar la carga de trabajo es una parte fundamental de la creación de políticas, y cuanto más sepa sobre la carga de trabajo, más afinada podrá estar su política.

En los tiempos de la computación por lotes, se desarrollaron varios planificadores no apropiativos; dichos sistemas ejecutarían cada trabajo hasta su finalización antes de considerar ejecutar un nuevo trabajo. Prácticamente todos los planificadores modernos son apropiativos y están bastante dispuestos a detener la ejecución de un proceso para ejecutar otro. Esto implica que el planificador puede realizar un cambio de contexto, deteniendo temporalmente un proceso en ejecución y reanudando (o iniciando) otro.

**First Come, First Served (FCFS)**

Es el más fácil de comprender: el primero en llegar es el primero en procesar. Simplemente es una cola de procesos listos, esperando a procesar.

**Shortest Job First (SJB)**

Partiendo de que se conoce el tiempo de ejecución de antemano (o cuanto menos una estimación), este planificador simplemente los ejecutará en orden del de menor tiempo al de mayor. Cabe resaltar que este planificador no es apropiativo, por lo que si fuese a llegar un proceso con menor tiempo de ejecución, éste tendría que esperar a que el proceso actual terminase de ejecutarse primero.

**Shortest Remaining Time Next (SRTN)**

Este planificador es, en esencia, una versión apropiativa de Shortest Job First. Siempre seleccionará el proceso con el menor tiempo restante de ejecución. Al ser apropiativo, si un nuevo proceso entra con menor tiempo de ejecución que el restante del que se está ejecutando actualmente, éste último será reemplazado.

**Round Robin (RR)**

Este planificador se podría llegar a clasificar como "el más justo" ya que le entrega a cada uno de los procesos un tiempo igual de ejecución, les sea suficiente para completar la ejecución de una sola vez o no.

**Event Driven (ED)**

A diferencia de Round Robin, que considera que cada proceso posee la misma prioridad e importancia, event driven le asigna a cada proceso una prioridad y los ordena en base a ella.

**Highest Response Ratio Next (HRRN)**

Se trata de un planificador basado en una prioridad por envejecimiento. La propiedad es en función de tanto su tiempo estimado de ejecución como del tiempo que lleva esperando para ejecutar.

**Multiple Level Queue (MQL)**

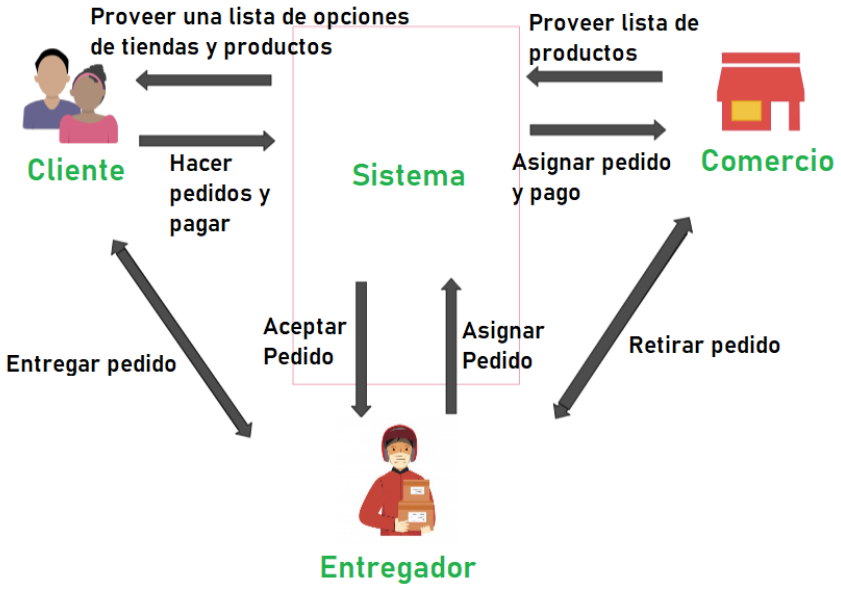
Simplificando, este planificador clasifica los procesos y los separa en diferentes colas, las cuales los ordenará utilizando el planificador que corresponda.

# Análisis del problema

En una primera instancia, podemos definir al problema que nuestro sistema se propone resolver es básicamente el de cómo llevar a cabo, de la manera más eficiente posible, el proceso que conecta a un cliente, con uno o más productos resultado de una compra que planea (o no) realizar. En otras palabras, el sistema deberá ser capaz de llevar a cabo un conjunto de tareas y procesos con el fin de vincular a potenciales clientes con potenciales productos

De esta forma, viéndolo con un enfoque de caja negra, el sistema recibirá un input, por ejemplo, una solicitud para un delivery de un conjunto de productos, y llevará a cabo los procesos necesarios (los cuales se definirán más adelante) usando ese input para producir un output correspondiente, por ejemplo, la entrega del producto solicitado.

Podemos diferenciar claramente a tres actores en el proceso anterior, sean estos; el **cliente**, el **repartidor** y el funcionario/dueño del **comercio**, cada cual con sus distintos roles y responsabilidades. A continuación, queda ilustrado en un diagrama las capacidades que anticipamos que puedan tener los distintos actores:



En nuestro sistema definimos anteriormente al pedido como un proceso (un macroproceso si se quiere), un proceso que en realidad está conformado por otra serie de procesos. Siguiendo esta línea, el pedido para nosotros sería un proceso, y no un recurso (aunque la idea de pedido luego se encapsulará en un objeto).

Un recurso para nosotros son los **repartidores** registrados, asociado a este recurso viene también una serie de procesos, por ejemplo: el que se encarga de asignar un pedido a un entregador disponible. Nos interesa entonces contar con un gran número de repartidores, ya que cuanto mayor sea el número de repartidores, mayor será la eficacia (debido a la mayor posibilidad de múltiples envíos en simultáneo) del sistema para el envío de pedidos al hogar de los clientes, además de poder contar con una posible mayor cobertura de zonas.

Otro recurso pueden ser los **comercios**, es decir, aquellos negocios que ofrecen productos en la plataforma. Sin estos, sería impensable la operación del sistema. Nos interesaría tener muchos negocios registrados ya que supondría una mayor variedad de productos y de horarios.

Como último recurso, nos gustaría mencionar el recurso de **los datos**, a medida que los usuarios realicen pedidos, se podrá almacenar esa información con el fin de reconocer ciertos patrones de compra, que luego podrían ser usados para por ejemplo, realizar sugerencias personalizadas. Si bien este recurso es totalmente prescindible, creemos que el almacenamiento y posterior procesamiento de los datos generados por los usuarios brinda valor al sistema y lo ayuda en su funcionamiento. (Este último recurso podría llegar a ser implementado en el futuro.).

En nuestro caso contemplaremos tres tipos distintos de comercios: restaurantes, almacenes y farmacias.

# Conceptos y problemas similares de sistemas operativos

Cuando se trata de conceptos de sistemas operativos, muchos fueron los conceptos que dimos y que nos sirvieron de base para nuestra solución.

Para empezar, cuando se trata de planificar los procesos de un sistema operativo, el mismo busca hacer el mejor uso de los recursos disponibles para llevar a cabo su función, nosotros incurrimos en el mismo problema cuando tratamos de planificar como asignaremos repartidores a los pedidos de manera que los pedidos se entreguen lo más rápido posible y que todos estén ocupados el mayor tiempo posible, tal y como ocurre con los recursos de los que dispone un sistema operativo, podríamos querer por ejemplo no tener nunca a un repartidor desocupado, así como no nos gustaría tener el CPU inactivo. Tampoco nos gustaría tener a un repartidor esperando mucho tiempo a que un pedido se realice fuera de un comercio, como tampoco nos gustaría tener un CPU esperando por E/S mientras podría estar haciendo otra cosa.

Pues bien, para esto hay que planificar, y en sistemas operativos existe el concepto de planificadores de corto, medio y largo plazo.

El **planificador de largo plazo** se encarga de balancear los tipos de procesos, aquellos que consumen CPU y aquellos que esperan por E/S por ejemplo, para facilitar la multiprogramación

(una técnica por la que dos o más procesos pueden alojarse en la memoria principal y ser ejecutados concurrentemente por el procesador o CPU.).

En nuestro caso, el planificador de largo plazo sería el que acepta pedidos nuevos, este, al igual que en el caso de los sistemas operativos, difícilmente rechazara a un pedido nuevo, ya que simplemente iría a una cola de pedidos en espera.

El planificador de mediano plazo es el que realiza lo que se conoce como swapping, es el que maneja los procesos que se suspenden por ejemplo por una solicitud de E/S.

El **planificador de corto plazo** es el que más se usa en un sistema operativo, este debería ser rápido y eficiente ya que se encarga de la planificación del CPU, su función principal es aumentar el rendimiento en base a ciertos criterios arbitrarios el sistema. Este planificador ayudará a elegir de un conjunto de procesos listos para ejecutar y asignarle CPU.

Este planificador se explicará más adelante, y es para nosotros el que se encargará de asignarle a un pedido un repartidor.

Nuestro sistema está pensado de la siguiente manera, una vez ingresa un pedido, se le notifica al comercio para que proceda a realizarlo, una vez listo el pedido, el local avisará y se le asignará un repartidor disponible.

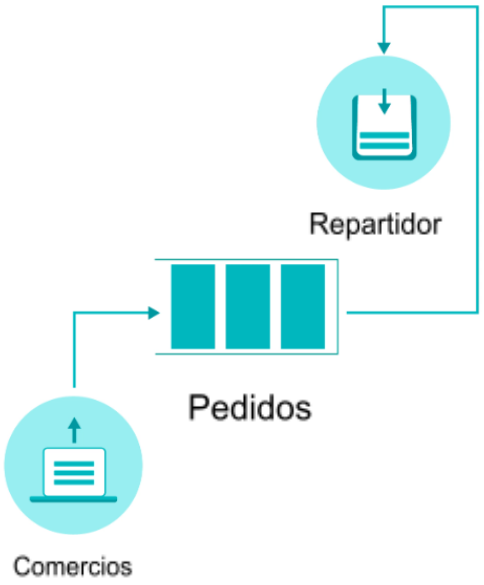
Los planificadores de los sistemas operativos suelen cumplir con ciertos criterios generales, sean los de **justicia**, es decir, no permitir que algunos no reciban tiempo de CPU en lo absoluto, además todos siguen **una política arbitraria**, el de nuestro sistema no es una excepción, y por último el criterio de **balance**, es decir, mantener a todas las partes ocupadas.

Además, ahora nos podemos servir de un criterio de planificación usado en los sistemas operativos y adaptarlo para nuestro sistema como un criterio de optimización, en cuestión el criterio de **tiempo de retorno**, que en nuestro caso sería el tiempo que transcurre desde que el usuario realizó el pedido hasta que se le entregó.

En cuanto a los diferentes planificadores, nos dimos cuenta durante el curso que **ninguno era realmente bueno en todos los contextos**, por lo que lo ideal sería poder tener una variedad de algoritmos diferentes en varias colas multinivel, para así poder adaptarse a todas las situaciones a las que se enfrentan los sistemas operativos.

También nos dimos cuenta, que algunos planificadores multinivel **pueden no ser eficientes porque no realizan una buena clasificación de los procesos**, poniendo a procesos muy diferentes en las mismas clases.

Otro concepto que nos puede ayudar mucho es un clásico problema de sistemas operativos, el de **productor-consumidor**, sirve para hacer una analogía con nuestro sistema, ya que tenemos muchos productores de pedidos (clientes) y muchos consumidores de pedidos (repartidores).




Otra forma de verlo es que, a medida que los pedidos son completados por los comercios, estos pueden ser identificados como los productores (de pedidos listos pendiente de entrega) y a los repartidores nuevamente como consumidores de estos pedidos para ser entregados.

En nuestra simulación, utilizaremos hilos de librerías de Java para lograr simular la concurrencia de clientes, repartidores y comercios.

Para finalizar, otro conjunto de herramientas que nos ayudará mucho, en concreto en la realización de nuestra simulación, serán las herramientas de sincronización, ya que, como ocurre en sistemas operativos, cuando varios procesos operan en un mismo entorno, necesitan sincronizarse, en nuestra simulación ocurre exactamente lo mismo, y deberemos usar las herramientas tales como semáforos y monitores para proteger las regiones críticas de nuestro código.

# Análisis de la planificación

Por un lado, como criterio de optimización podríamos desear reducir al mínimo posible el tiempo que transcurre desde que se inicia un pedido hasta que este llega al cliente. Por otro lado, podríamos querer optimizar la cantidad de pedidos que hace un repartidor por hora, para que nuestros repartidores sean eficientes.

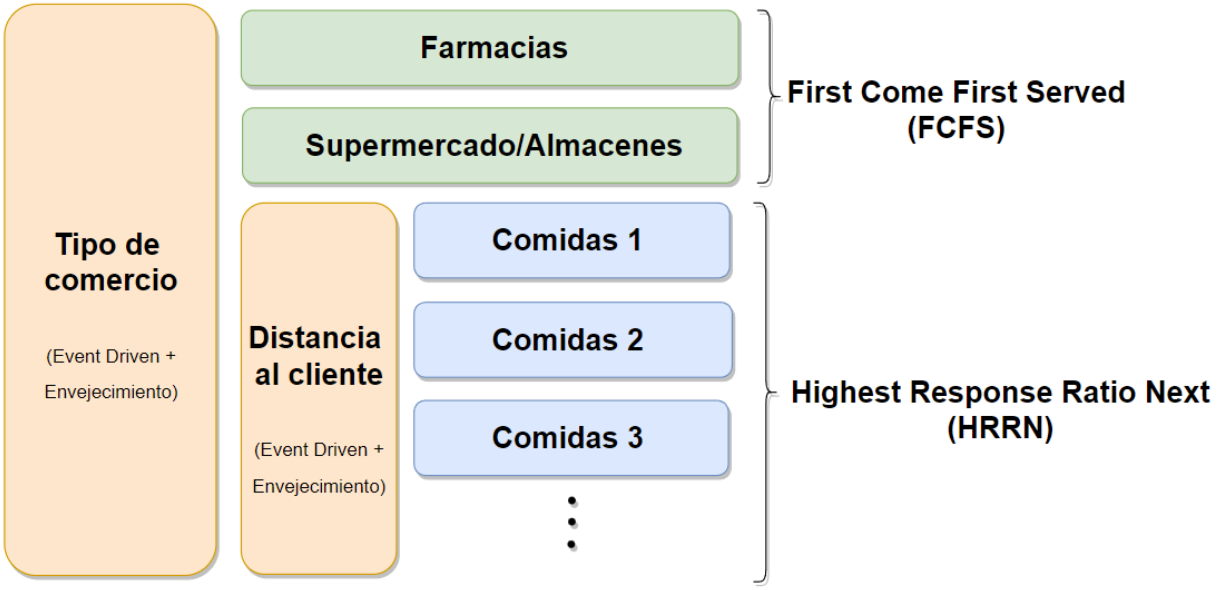
En nuestro caso particular, podríamos querer optimizar los pedidos cortos de restaurante por encima de los pedidos largos, y los pedidos de farmacias por sobre los de almacenes y supermercados.

En nuestro caso, decidimos optar por el siguiente criterio de optimización: darle mayor prioridad a los pedidos de farmacias y almacenes, ya que estos son menos comunes y su tiempo de elaboración es cero.

Entre ellos se le da mayor prioridad a las farmacias debido al posible carácter urgente del pedido.

Por otro lado, en un tercer nivel de prioridad, se encuentran los pedidos de restaurantes, a los cuales, a su vez, se les asignan diferentes prioridades en base a su tiempo de elaboración y a su distancia del cliente.

En consecuencia, el planificador que se nos ocurrió, utiliza una cola Event Driven con envejecimiento para diferenciar las colas First Come First Served de farmacias y almacenes, y los restaurantes los cuales se encuentran gestionados por otra cola Event Driven con envejecimiento la cual se diferencia a las distintas colas Highest Response Ratio Next de restaurantes, las cuales ordenan sus pedidos, en base al tiempo de elaboración y su antigüedad en ticks.




Los nuevos pedidos de farmacia y almacenes, ingresan directamente al final de la cola de farmacias y almacenes y serán atendidos en el orden de llegada.

Por otro lado, los nuevos pedidos de restaurantes, cuya distancia sea menor a 10, ingresarán a la cola Comidas 1, aquellos que sean menores a 15, ingresarán a la cola Comidas 2, y los demás ingresarán a la cola Comidas 3.

El envejecimiento en nuestro planificador opera de la siguiente manera: para ser transferido de la cola Comidas 3, a la cola Comidas 2, los pedidos deberán tener una antigüedad de cómo mínimo 20 ticks en la cola Comidas 3. Lo mismo sucede para que un pedido sea transferido de la cola Comidas 2 a la cola Comidas 1.

Luego, para que un pedido sea transferido de la cola Comidas 1 a la cola de Supermercados/Almacenes, el mismo deberá haber esperado en la cola Comidas 1 un mínimo de 25 ticks, mientras que, para ser transferidas desde Supermercados/Almacenes a Farmacias, deberá haber esperado un mínimo de 30 ticks.

Las colas Highest Response Ratio Next, se implementaron utilizando los siguientes cálculos:

Utilizando el tiempo de elaboración obtenemos un valor entre 0 y 100, el cual 0 corresponde a un tiempo de elaboración de 45 y 100 a 10. Luego, la prioridad de cada pedido es el máximo entre este valor y 5 multiplicado por la antigüedad en ticks.

Aclaración: cada comercio puede elaborar un pedido a la vez, y aquellos pedidos a la espera, serán ingresados a una cola Highest Response Ratio Next idéntica a las colas de Comidas del planificador.

# Diagramando la solución

Para realizar la simulación, es imperativo entregar como datos iniciales los pedidos que se realizarán a lo largo del escenario, los comercios afiliados en ese momento y el número de repartidores disponibles.

Esto deberá ser cargado en los dos archivos archivos de entrada “Comercios.csv” y “Pedidos.csv”.
El formato de “Comercios.csv”, es el siguiente, en la segunda columna de la primera fila, se especifica el número de repartidores que tendremos en nuestra simulación. En el resto de las filas, contendrán en la columna 1 el nombre de los comercios de nuestra simulación, no hay mayor restricciones para estos, pero aquellos nombres que se especifiquen aquí, deberán coincidir con los nombres que se ingresen en la siguiente entrada.

En el archivo de entrada “Pedidos.csv”, el formato será el siguiente: en la primera columna, se ingresará el ID del pedido, de manera ***decreciente***, en la segunda columna, un nombre de comercio que coincida con alguno de los comercios especificados en la entrada anterior (“Comercios.csv”). En la tercera columna irá el tipo de comercio, el cual debe ser uno de los siguientes tres valores: “restaurante”, “farmacia”o “almacen”. En la cuarta columna irá el tiempo de elaboración, el cual será “***0***”, si el tipo de comercio es “farmacia” o “almacen”, o un valor ***entre 10 y 45,*** si el tipo de comercio es “restaurante. En la quinta columna irá la distancia al cliente, la cual tendrá un valor entre ***5 y 20.*** Por último, en la sexta columna irá el minuto de ingreso, cuyos valores pueden repetirse, pero ***deben*** estar ordenados ***decrecientemente***. Es por esto que, para crear pedidos manualmente, se recomienda empezar desde abajo hacia arriba en el archivo de entrada. Otra opción es generarlo automáticamente utilizando el script provisto de python el cual genera 1000 pedidos aleatorios, en los que 8 de cada 10 son restaurantes, la elaboración y la distancia son aleatorias y el minuto de ingreso es entre 0 a 5 minutos mayor al anterior de manera aleatoria. El archivo se llama “GeneradorDePedidosRandom.py”

A los comercios se les proporcionará una ID, un tipo de comercio y un nombre. Para los repartidores, nos basta con saber únicamente de cuántos disponemos.

Para esta simulación, creemos pertinente el asignarle los valores de 1000 pedidos de 30 comercios diferentes para 15 repartidores.

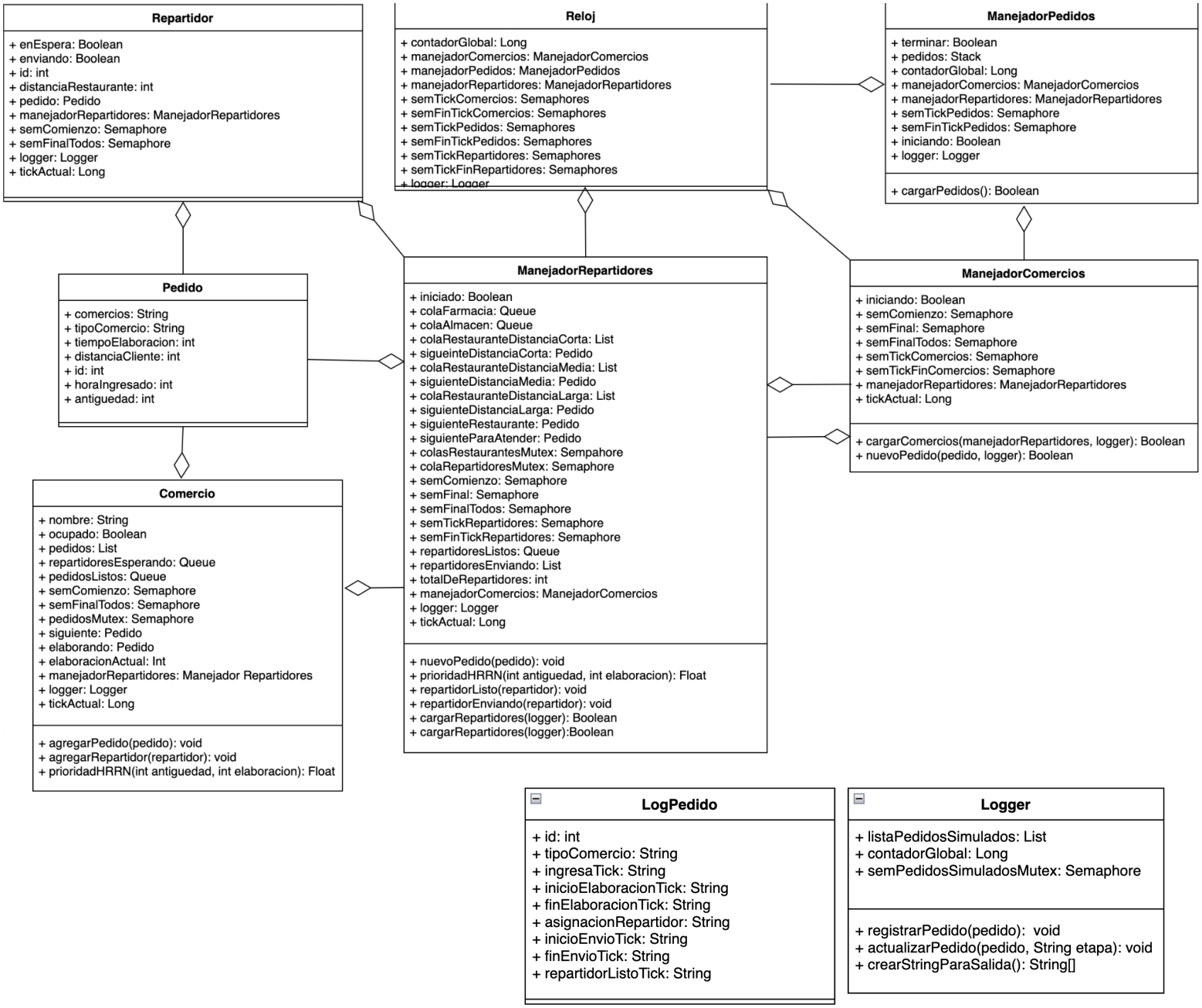
A medida que transcurre el tiempo, el programa comienza a guardar la información de todo lo que ocurre en la simulación, y calcula algunos datos a partir de ésta, todo esto dividido en cuatro archivos de salida. En la bitácora de pedidos se almacena la información de qué ocurre en cada tick con respecto a los pedidos. Esto es, por ejemplo, cuándo se ingresó al sistema, cuándo se comenzó a elaborar, cuándo finalizó de elaborar, cuándo se le asignó un repartidor y cuándo fue entregado.

De igual forma, la bitácora de repartidores guarda la información en el mismo formato, pero acerca de los repartidores. Es decir, cuándo un repartidor es asignado a un pedido, cuándo comienza a repartir, cuándo entrega un pedido y cuando vuelve a estar disponible.

En el archivo “ResumenConIndicadores” se encuentran los datos de los pedidos, ahora ordenados por pedido. Con todos los datos antes mencionados, más algunos indicadores generados a partir de estos. Este archivo es especialmente útil a la hora de corroborar que no hayan errores de eficiencia (como que un pedido espere demasiado tiempo), sin necesidad de recorrer todo los “ticks” en la bitácora de pedidos.

Finalmente, en el archivo “SaturaciónDeColas”, como sugiere su nombre, alberga la información de las colas de planificación de los repartidores.

En lo que a nuestra implementación refiere, mostramos a continuación el siguiente diagrama de clases, y luego procederemos a explicar cómo opera el programa:



Empezaremos a explicar el flujo del programa desde el main:
En el main leemos las entradas, creamos instancias de ManejadorComercios, ManejadorPedidos y ManejadorRepartidores, luego creamos y cargamos los objetos pedido, comercios y repartidores, a cada manejador, y para cada instancia de comercios y repartidores, se crea un hilo y se ejecuta el método run, el cual en todos los hilos hace esperar por el signal de un semáforo, de forma que todos quedan bloqueados.

A continuación, se crea un hilo por cada manejador y se inician, por último, en el hilo main se ejecuta el método run del hilo reloj.

A partir de esto, el reloj se mantendrá en un bucle hasta que se hayan simulado todos los pedidos.
La responsabilidad de la clase reloj es contar los ticks de la simulación, liberando semáforos, de modo que cada uno de los tres manejadores avancen un tick en sus respectivas tareas:

El manejador de pedidos se encarga de simular la llegada de uno o varios pedidos, en caso de que la hora de ingreso de estos pedidos coincida con el tick actual.

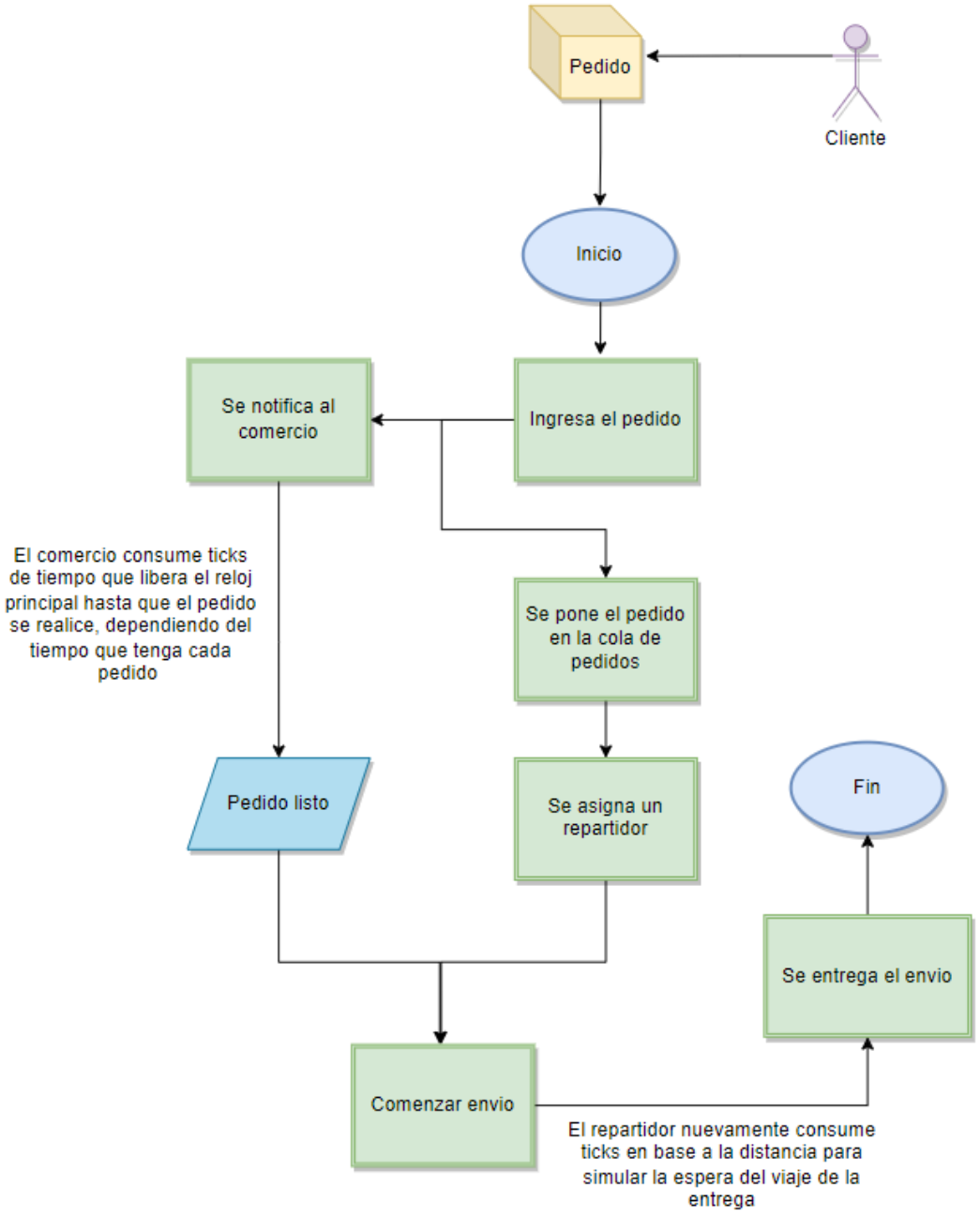
El manejador de comercios se encarga, utilizando tres semáforos, de que cada uno de los comercios avancen su elaboración, uno y solamente un tick.

El manejador de repartidores, por un lado se encarga de, utilizando tres semáforos, que cada uno de los repartidores avancen su envío, uno y solamente un tick, y por el otro lado, es también el encargado de, utilizando nuestro planificador previamente mencionado, asignarle repartidores, a los comercios, tratando de hacerlo de forma que los repartidores lleguen a los comercios aproximadamente cuando estos están terminando de elaborar el pedido.

Cada uno de estos manejadores, una vez terminadas sus tareas, libera un semáforo indicando al reloj que puede continuar al siguiente tick.

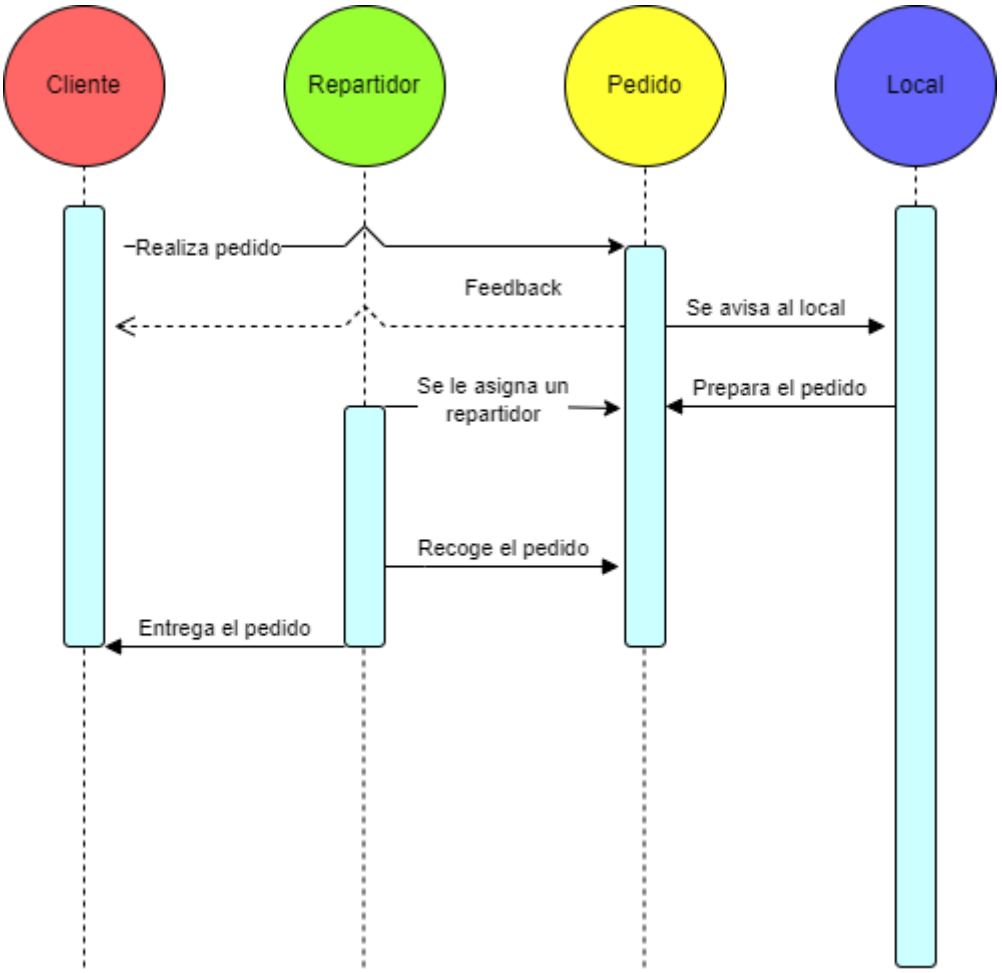
A su vez, los comercios en cada tick se encargan de avanzar en la elaboración, de gestionar prioridades de una cola HRRN de pedidos en espera para ser elaborados, y de asignarle a los repartidores que estén en espera, pedidos que ya estén elaborados.

Por otro lado, los repartidores se encargan de, en cada tick, disminuir la distancia restante la cual representa la cantidad de ticks que faltan para finalizar el envío, y también, se encarga de simular, utilizando nuevamente la distancia restante, el regreso del repartidor hasta volver a estar disponible para un nuevo pedido.




Durante la ejecución de cada tick, se registran todos los eventos relevantes en dos bitácoras y en la clase Logger, y también el estado de las colas del planificador del manejador de repartidores en el archivo “SaturacionDeColas.csv”.

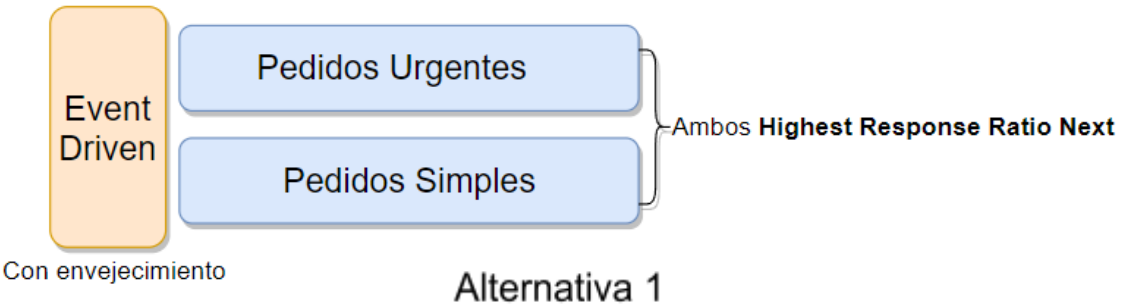
Una vez simulados todos los pedidos, el bucle del reloj finaliza, volviendo al main, el cual utilizando la clase Logger, genera la salida “ResumenConIndicadores.csv”, y a continuación, finaliza la simulación.




# Alternativas

Lo que sigue son tres alternativas posibles para un planificador de repartos para nuestro sistema, el objetivo del mismo es manejar el flujo de trabajo y la asignación de los repartidores a nuevos pedidos.

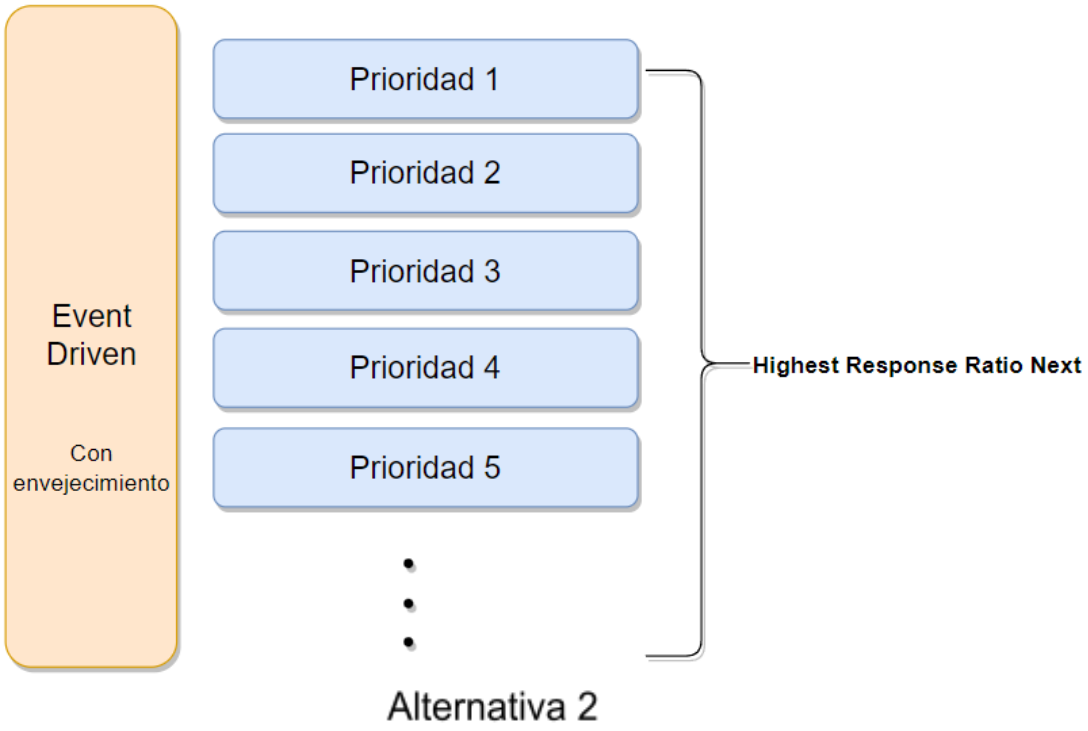
Como primera solución se nos ocurrió el siguiente planificador:




Alternativa 1

Este planificador recibe pedidos y los separa en dos clases, aquellos que requieren una entrega rápida (por ejemplo productos de farmacia) son asignados a la cola de pedidos urgentes, y aquellos que no son urgentes (por ejemplo un pedido de una pizza) son asignados a la cola de pedidos simples.
Ambas son parte de otra cola Event Driven con envejecimiento por lo que aquellos pedidos simples que no sean atendidos durante mucho tiempo pueden terminar siendo ingresados a la cola de pedidos urgentes.

Tanto la cola de Pedidos Urgentes como la de Pedidos Simples son Highest Response Ratio Next (HRRN) ya que de esta forma aquellos pedidos cuyos envíos sean más rápidos serán atendidos primero pero tras unos minutos (para que por ejemplo, la comida caliente no se enfríe) los pedidos más largos también serán atendidos.
A continuación dejamos una segunda alternativa:



Alternativa 2

Este planificador recibe pedidos, y basándose en sus características les asigna una prioridad que determina el orden en el que se le notificará del mismo al local siendo enviado tras su elaboración. Siendo prioridad 1 la mayor prioridad y prioridad 5 la menor prioridad.

La prioridad se asigna de la siguiente manera:

- **Pedido normal** = distancia * 0.3 + complejidad * 0.7
- **Pedido cliente premium** = el mínimo entre su cálculo normal y 2
- **Pedido cliente nuevo** = el mínimo entre el su cálculo normal y 2
- **Pedido de farmacia** = 1

**Distancia** = Valor del 1 al 5 siendo 1 muy cerca y 5 muy lejos.
**Complejidad** = Valor del 1 al 5 siendo 1 pocos platos o simples y 5 muchos o complejos.

Tras haber sido calculada la prioridad, está aún puede ser ajustada mediante envejecimiento permitiendo que un pedido que demore en ser atendido sea asignado a una cola de mayor prioridad.
Las colas de todas las prioridades son Highest Response Ratio Next (HRN) ya que de esta forma aquellos pedidos cuyos envíos sean más rápidos serán atendidos primero pero tras unos minutos (para que por ejemplo, la comida caliente no se enfríe) los pedidos más largos también serán atendidos.

# Justificación de nuestro planificador

De las 3 colas planteadas (las 2 alternativas y nuestra cola implementada) llegamos a la conclusión de que nuestra opción implementada es la más indicada por los que se comentará a continuación.
Entendemos que la primera opción es demasiado simple y no toma en cuenta las diversas clasificaciones que debemos considerar en nuestros pedidos, no hay suficientes colas para clasificar los distintos tipos de los pedidos ni los diferentes tiempos de elaboración que los pedidos requieren a la hora de elaborar un plato resultando en que no tengamos un control real de las prioridades de nuestro sistema.

Por otra parte, la opción 2 nos permite controlar nuestro sistema basado en prioridades de pedidos, las cuales son escalables no solo en cantidad de colas sino en cantidad de clasificaciones (cliente vip, nuevo, etc). Aun así, esta solución por sí sola no es aplicable a nuestro escenario ya que calcular un valor a partir de tanto la distancia de envío como la complejidad de elaboración para luego empezar a elaborar y enviar el pedido al cliente no es correcto. Si bien la idea es que los pedidos más simples y rápidos se atiendan primero, no existe ningún mecanismo para intentar optimizar la disponibilidad de repartidores de forma de que estos lleguen al local a una hora próxima aproximada a la que el local esté listo para entregar el pedido, en cambio en esta opción los pedidos comienzan a elaborarse y llaman al repartidor más cercano al mismo tiempo, generando tiempo ocioso entre los repartidores.

El planificador elegido y explicado en la sección “Análisis de la planificación” intenta que los repartidores lleguen al local con la menor diferencia de tiempo posible a cuando el local termina la elaboración del producto mediante la clasificación en colas según el tipo de local. A diferencia de la opción 2, en este sistema se notifica al local ni bien se recibe el pedido y se comienza a clasificar desde ese punto, a aquellos pedidos que no requieren de tiempo de elaboración (los cuales se espera que sean la minoría) se les intentará dar mayor prioridad mientras que aquellos que sí la requieren se les asignará una prioridad menor cuanto más tiempo de elaboración se espere.

Otra cosa que se tiene en cuenta la opción elegida es que los repartidores son limitados, es por esto que aquellos pedidos con una distancia corta que resulte en un envío rápido serán atendidos antes que aquellos con una mayor distancia de esta forma reduciendo la presencia del efecto convoy, aún así, pasados unos pocos minutos, aquellos pedidos de distancias largas serán atendidos igualmente antes de que la comida se enfríe o pierda calidad.

Otra característica de la cola de nuestra elección es que al igual que la segunda es escalable, se pueden agregar nuevas colas basadas en distintos tipos de pedidos o tiempos de elaboración.

En cuanto al cálculo de la prioridad en las colas de comidas, esta se basa en el tiempo de elaboración estimado de cada comida, esto nos resultó lógico, ya que cuando llega el pedido el mismo tambien se envia al comercio, y si se estima (quizás por pedidos anteriores) que el tiempo de elaboración de la comida es alto, de nada servirá asignarle rápido un repartidor, ya que este quedara esperando por el comercio mientras podría estar realizando otras entregas más rápidas, algo similar a que el CPU se quede esperando por E/S mientras podría estar realizando otras tareas.

# Resultados

Tal y como fué mencionado previamente la salida de nuestro programa consta de 4 archivos .csv cada uno reflejando diferente información, estadísticas e indicadores de la simulación realizada.

Si bien las conclusiones son las mismas independientemente de las entradas, los resultados específicos varían, es por esto que todas las salidas y entradas utilizadas para generar las siguientes conclusiones serán adjuntadas junto con este documento. Se utilizaron para esta simulación como entrada 24 restaurantes, 3 farmacias, 3 almacenes, 15 repartidores, y se generaron 1000 pedidos los cuales cada uno ingresa aleatoriamente entre 0 y 5 ticks después del anterior (generando aleatoriamente momentos de mayor y menor saturación).

Empecemos centrándonos en la salida "ResumenConIndicadores.csv"; este archivo tal y como su nombre lo indica nos brinda información sobre los eventos significativos de cada pedido (hora de ingreso, elaboración, envío, etc) y nos genera algunos indicadores con los cuales podemos identificar que tan correcto es el comportamiento de nuestro sistema:

**Demora Total del pedido:** Este indicador nos brinda una idea de la viabilidad de nuestro sistema ya que nos calcula el tiempo transcurrido desde que un usuario hace un pedido hasta que llega a su casa.
Resultados:

| | |
|---|---|
| MÁXIMO | 138 |
| MINIMO | 6 |
| PROMEDIO | 47.327 |

Teniendo en cuenta que para restaurantes el tiempo de elaboración es, dependiendo del pedido, un numero de ticks entre 10 y 45 y que la distancia (equivalente al tiempo de envío en ticks) es un valor del 5 al 20, creemos que un promedio de 47,3 ticks es una cantidad razonable mientras que el máximo de 138 ticks podría, en primera instancia, parecer elevado, pero debido a la escasa frecuencia de estos casos creemos que es un aspecto a mejorar pero no un problema grave.

**Retraso de comienzo de elaboración:** Este indicador representa el tiempo transcurrido desde que un pedido ingresa al sistema hasta que se empieza a elaborar indicando cuando un restaurante recibe múltiples pedidos en poco tiempo y debido a que en nuestra simulación cada local elabora un pedido a la vez, los que lleguen a un restaurante ocupado esperaran en una cola HRRN a ser elaborados.
Resultados:

| | |
|---|---|
| MÁXIMO | 99 |
| MINIMO | 0 |
| PROMEDIO | 7.809 |

Creemos que estos valores son adecuados ya que la relación de comercios, repartidores y pedidos que utilizamos como entradas fueron calculados para que existan momentos de mayor y menor saturación dentro de unos márgenes de tiempo razonables.

**Espera con pedido elaborado:** Este indicador es muy importante para medir la precisión de la asignación de repartidores de nuestro sistema. Nuestro planificador asigna prioridades entre otras cosas en base a la distancia (para una mayor rotatividad de repartidores) y en base al tiempo de elaboración (para que el pedido esté el menor tiempo posible parado antes de ser enviado), este indicador nos permite comprobar la precisión del segundo objetivo.
Resultados:

| | |
|---|---|
| MÁXIMO | 61 |
| MINIMO | 0 |
| PROMEDIO | 6.035 |

En cuanto al promedio, estamos satisfechos con el valor de 6 ticks, creemos que teniendo en cuenta que planteamos la simulación para que haya momentos en los que los repartidores se ven saturados debido al número de pedidos, este valor es más que aceptable. Por otro lado, al ver el máximo siendo un número tan diferente del promedio, entendemos que existe una minoría de pedidos que experimentan un retraso para recibir un repartidor.
Lo primero es entender la precisión del sistema, para esto diferenciaremos aquellos pedidos que esperen más o menos de 20 ticks en recibir un repartidor tras ser elaborados:

| Restaurantes | Almacenes | Farmacias | Veces que se demoró mas o menos de 20 ticks en comenzar un envio | Numero de repartidores simulados |
|---|---|---|---|---|
| 24 | 3 | 3 | 89<br>911 | 15 |

A partir de estos valores calcularemos la precisión de asignación:


Basándonos en estos resultados entendemos que si bien nuestro sistema tiene una precisión muy elevada, sería ideal tener como objetivo que el retraso tienda a cero, para de esta forma prevenir los casos aislados en los que un usuario realiza un pedido y debe esperar un tiempo elevado para recibirlo.
Las principales razones que causan este problema en la minoría de los envíos las podemos identificar gracias a estos resultados y a los del siguiente indicador, es por esto que las discutiremos tras el análisis de dicho siguiente indicador.

**Tiempo para asignar repartidor:** Este último indicador nos permite saber el tiempo transcurrido desde que ingresa el pedido al sistema hasta que se le asigna un repartidor, esto es muy importante ya que refleja el comportamiento de nuestro planificador, podemos observar si en efecto se le da prioridad a aquellos pedidos a los que queremos asignarles un repartidor antes.
Resultados:

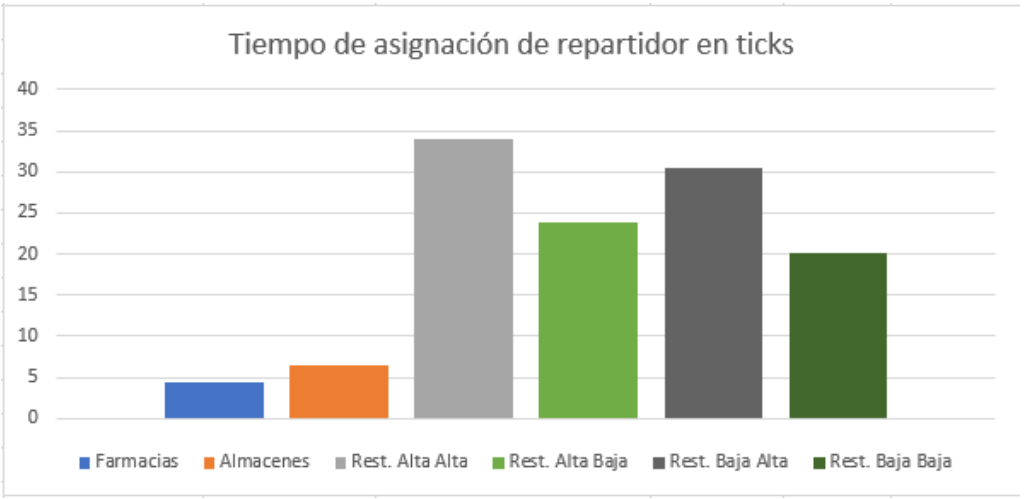
| | |
|---|---|
| MÁXIMO | 87 |
| MINIMO | 1 |
| PROMEDIO | 23.39 |

Si bien estos valores en general parecen ideales teniendo en cuenta las entradas, debido a que los valores esperados son diferentes dependiendo del pedido, para poder sacar conclusiones debemos diferenciar los resultados:

| Promedio de demora de asignación de repartidores para **FARMACIAS** | Promedio de demora de asignación de repartidores para **RESTAURANTES** | Promedio de demora de asignación de repartidores para **ALMACENES** |
|---|---|---|
| 4.469387755 | 27.98490566 | 6.579439252 |

| | |
|---|---|
| Promedio RESTAURANTES con **ELABORACION ALTA** y **DISTANCIA ALTA** | 34.02898551 |
| Promedio RESTAURANTES con **ELABORACION ALTA** y **DISTANCIA BAJA** | 23.85377358 |
| Promedio RESTAURANTES con **ELABORACION BAJA** y **DISTANCIA ALTA** | 30.53191489 |
| Promedio RESTAURANTES con **ELABORACION BAJA** y **DISTANCIA BAJA** | 20.05084746 |

*Elaboracion: 10 a 45 ticks | Distancia: 5 a 20 ticks




Una vez hecha la distinción, teniendo en consideración los valores de elaboración y distancia de los pedidos observamos que el planificador funciona muy bien, dando prioridad a las farmacias y almacenes los cuales no tienen elaboración y repartiendo la prioridad entre los restaurantes en base a su tiempo de elaboración y distancia al cliente.
Aún así, otra cosa que detectamos en estos resultados es que entre la distancia y la elaboración, se le da más peso del que anticipamos a la distancia lo cual tiene como positivo una alta rotatividad de repartidores ya que priorizan aquellos pedidos que les van a permitir volver a estar listos lo antes

posible. Mientras que el no darle el suficiente peso a la elaboración tiene como efecto negativo el problema que detectamos en el indicador anterior, una menor precisión en la asignación de repartidores a restaurantes. Si bien es verdad que esto solo afecta a la minoria de los casos, es nuestro objetivo crear un sistema lo más eficiente posible y es por esto que detectamos que lo primero que en lo que deberíamos centrarnos para mejorar nuestro sistema es en modificar el planificador para brindarle un mayor peso al tiempo de elaboración a la hora de calcular la prioridad de los pedidos de los restaurantes.

Las otras 3 salidas de nuestro programa constan de 2 bitácoras las cuales registran todos los eventos de nuestra simulación por cada tick, siendo muy útiles para observar qué pasa en la simulación en cada momento y para corroborar la información obtenida a partir de las demás salidas.

Por último, el archivo “Saturación Colas.csv” nos permite tal y como su nombre lo indica, comprobar que tan saturadas están las colas del planificador en cada tick dándonos los valores de el número de pedidos en cada cola además del ID del pedido de mayor prioridad de los restaurantes y el ID del de mayor prioridad de todos (el siguiente a ser atendido).
Esta salida es de gran utilidad para observar nuestro planificador ya que nos permite observar ejemplos de pedidos siendo atendidos antes que otros que hayan llegado antes:

| ID Siguiente Para Atender |
|---|
| Ninguno |
| 0 |
| 1 |
| 2 |
| Ninguno |
| Ninguno |
| Ninguno |
| Ninguno |
| 3 |
| Ninguno |
| 4 |
| 6 |
| 5 |
| Ninguno |

*En este ejemplo el pedido 6 ( de una farmacia) se adelanta al 5 (de un restaurante)

| |
|---|
| 20 |
| 20 |
| 23 |
| 25 |
| 26 |
| 27 |
| 22 |
| 22 |

*En este ejemplo un pedido de menor prioridad para un restaurante (22) es atendido después del 23, 25 y 26, todos pedidos para restaurantes de mayor prioridad.

# Conclusiones

Luego analizar nuestras salidas y nuestros indicadores, nos dimos cuenta que nuestra evaluación general con respecto al resultado obtenido es buena, y cumple con nuestros criterios de optimización, ya que como vimos antes, las farmacias adelantan efectivamente a los restaurantes, y entre los restaurantes, se respeta la prioridad planteada.

Si bien la solución nos resultó, de manera general, correcta, también nos pusimos a la tarea de pensar y encontrar áreas en las que nuestro programa podría ser mejorado. Tomando por ejemplo el máximo tiempo que se tardó en asignarle un repartidor a un pedido, éste es un error que si bien no termina siendo algo grave, debido a la relativa baja frecuencia con la que ocurre, es algo que podría llegar a perfeccionarse si se dispusiera de más tiempo.
O quizás, por mencionar otro ejemplo, también podría llegar a mejorarse la relación de importancia entre la distancia del destino y el tiempo de elaboración de un pedido, a la hora de decidir a qué pedido se le asignará un repartidor primero. Ya que no es lo mismo un pedido que tarde mucho en hacerse y poco en entregarse, que uno que tarde poco en hacerse y mucho en entregarse.

Se nos ocurrió, por ejemplo, que para mejorar la eficiencia podríamos modificar el planificador para que las colas HRRN en vez de determinar la prioridad de los pedidos en base a la elaboración lo hagan en base a la distancia al destino y que en cambio sea la cola Event Driven que se encargue de diferenciar las colas en base a la elaboración en vez de la distancia, de esta forma invertimos el peso de dichas variables en la prioridad de nuestros pedidos para de esta forma hacer aún más pequeño el porcentaje de pedidos que demoran en recibir un repartidor

En lo que a operar con múltiples hilos refiere, concluimos que es una labor minuciosa y hay que prestar especial atención a los recursos compartidos. También entendimos que el uso de semáforos puede conllevar errores y que esto se puede solucionar utilizando monitores.

En cuanto a la planificación de procesos, nos quedó claro que no existe un planificador general que resuelva todos nuestros problemas de manera óptima, el planificador depende en grán medida de la naturaleza del problema en particular y de los datos que se tengan (o no) de antemano. También nos quedó claro que siempre es mejor, cuando se trata de planificadores MLQ, no escatimar en las clasificaciones, ya que son éstas las que realmente nos dan control sobre la forma en la que deseamos planificar nuestros procesos.

Éste programa, como cualquier creación humana, está sujeta a innumerables mejoras si se lo estudia con la suficiente rigurosidad. Sin embargo, como mencionamos al comienzo de este apartado, estamos satisfechos con el resultado. Creemos que fuimos capaces de tomar los conceptos de sistemas operativos aprendidos en clase (particularmente conceptos cómo hilos, semáforos y planificadores), y convertirlos en un programa coherente, eficiente y, sobre todo, funcional.

# Bibliografía


- Remzi Arpaci-Dusseau, Operating Systems: Three Easy Pieces, Kindle Edition
- Andrew S. Tanenbaum, Sistemas Operativos Modernos, 3era Edición
- https://www.fing.edu.uy/tecnoinf/mvd/cursos/so/material/teo/so05-hilos.pdf
- https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html
- http://www.infocobuild.com/education/audio-video-courses/computer-science/cs162-spring2015-berkeley.html