LABORATORIO 2

ENTREGA FINAL

SISTEMAS OPERATIVOS

Docente: Danilo Da Rosa

Autores:

- Agustina Barcos
- Víctor Nicola
- Facundo Sevrini
- Thamara Yedig

Resumen de la información teórica utilizada.

Para la realización de este trabajo utilizamos diversas fuentes de información, como lo fue principalmente la teoría vista en clase. Como primer paso tuvimos que pensar cómo llevar el caso propuesto de la realidad para modelarlo. Originalmente, nuestra idea consistía en pensarlo como el problema de productor-consumidor, este ejemplo consiste en que el productor debe generar un producto, almacenarlo y comenzar nuevamente; mientras que el consumidor debe tomar (simultáneamente) productos uno a uno, el problema está en que el productor no añada más productos que la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío. Entonces, en nuestro caso, el productor sería uno único, la planta productora de garrafas, y el consumidor serían todos los camiones para consumir dichas garrafas. Seguidamente, debimos decidir a qué consumidor se atenderá primero, por lo que debimos hacer pruebas con los diferentes algoritmos de planificación de procesos. Utilizamos los algoritmos FCFS, SJF, y MLQ con Round Robin.

El algoritmo FCFS es el más simple de todos, consiste en asignarle la CPU al primer proceso que la solicite. Cuando un proceso llega a la cola de procesos listos y hay otro proceso haciendo uso de la CPU, este deberá esperar hasta que termine y sea eliminado de la cola, luego podrá tener la CPU libre.

El algoritmo SJF consiste en asignarle la CPU al proceso más corto, dependiendo de su ráfaga. Cuando la CPU queda libre se le asigna otro proceso con la ráfaga más corta, y si sucede que dos procesos tienen el mismo tiempo de ráfaga, este algoritmo funciona como FCFS.

El algoritmo por prioridades, también llamado Event Driven se basa en darle una prioridad a cada proceso y la CPU se le asigna al proceso que tenga mayor prioridad. Cuanto más larga sea la ráfaga de CPU, menor será la prioridad y viceversa. Como en el algoritmo anterior, si dos procesos tienen misma prioridad, se comporta como un FCFS.

El algoritmo MLQ o mediante colas multinivel consiste en clasificar los procesos. Por ejemplo, se pueden clasificar en procesos interactivos y procesos por lotes. Entonces, la cola de procesos listos se divide en diferentes colas. Si a este algoritmo se le agrega Event Driven, entonces se obtiene un algoritmo que planifica los procesos de las colas mediante prioridades.

Optamos por el algoritmo más justo, siendo FCFS porque respeta los tiempos de llegada. Luego debimos tomar en cuenta la sincronización de los procesos, de modo que dos camiones no se cargaran al mismo tiempo, como solución base monitores y semáforos.

Un semáforo es una variable entera que restringe el acceso a recursos compartidos en un entorno donde haya varios procesos ejecutándose al mismo tiempo. Se diferencian en

semáforos contadores (pueden variar en un dominio no restringido) y semáforos binarios (los valores solo pueden ser 0 o 1)

Los monitores surgen como una solución al problema de los semáforos, ya que estos tienen la desventaja de estar distribuidos por todo el código, y son muy complejos de implementar. Con monitores un solo hilo puede acceder al monitor a la vez, si sucede que un hilo quiere usar un monitor que está en uso, este se bloquea hasta que quede libre.

Dicho todo esto, nuestra primera solución consistía en crear dos tipos de hilos diferentes, una clase productor que representa al hilo encargado de producir garrafas llenas y una clase vehículo donde cada hilo consume garrafas según su capacidad. Los hilos correspondientes a los vehículos se crearían leyendo un archivo que contiene las capacidades de los distintos camiones. Se les asigna un identificador, la capacidad y se los inicializa. Luego para garantizar que dos camiones no se cargaran al mismo tiempo (exclusión mutua) utilizamos monitores.

Análisis del problema, identificando y justificando qué procesos y recursos se han de utilizar.

Optamos por solucionar el problema utilizando el ejemplo del productor consumidor. Existen dos clases de hilos en el programa, uno siendo el productor (planta generadora de garrafas), y también están los hilos consumidores que son todos los camiones generados a partir de un archivo txt con sus respectivas capacidades. Los vehículos compiten por los recursos, que en este caso serían las garrafas, teniendo en cuenta que los vehículos pueden consumir garrafas uno a la vez.

Análisis de la planificación de todo el sistema.

Tratamos de simular la situación de la forma más real posible, pero nos vimos limitados por nuestros conocimientos de programación concurrente.

Una de las limitaciones que se nos presentó, fue simular la interrupción de la carga de un camión por la llegada de una camioneta, pues no estamos muy familiarizados con el manejo de hilos, así como de la zona crítica. Esto fue un impedimento para la implementación de Round Robin, nuestra idea original.

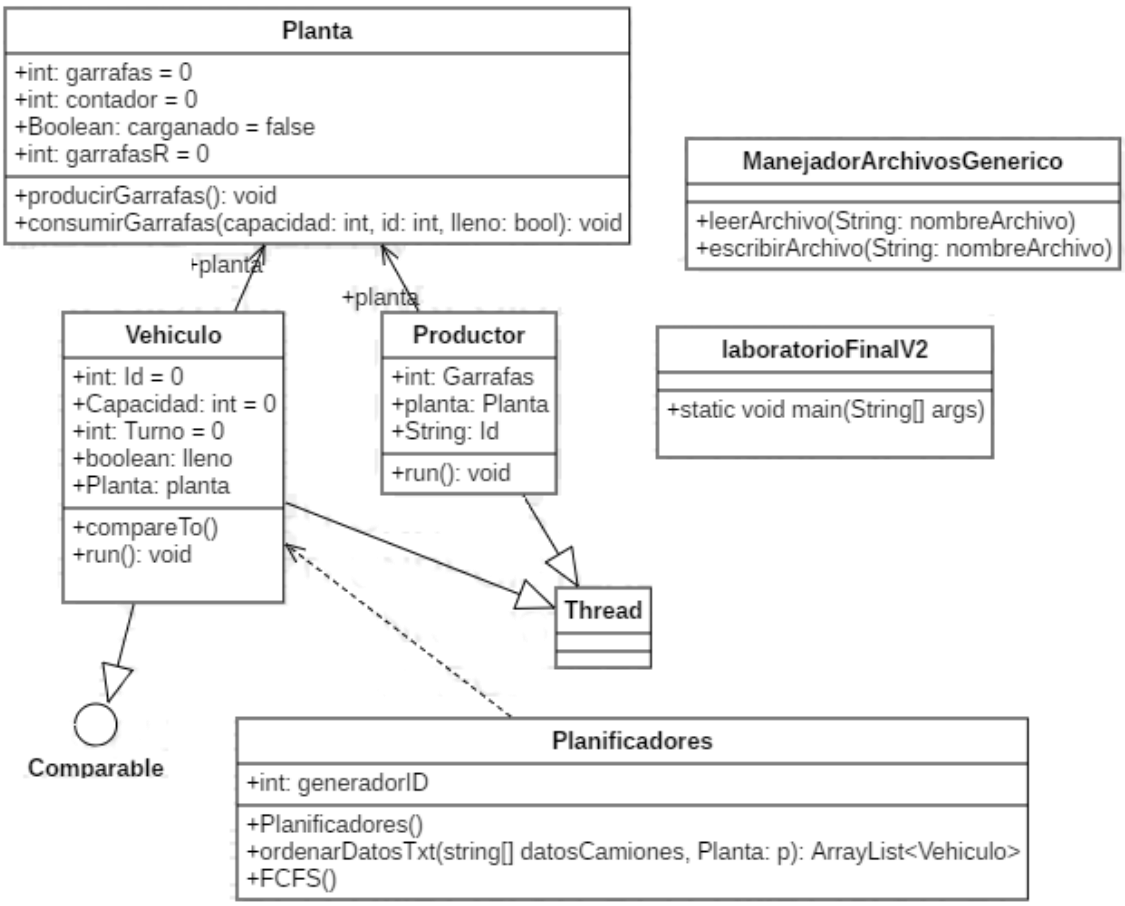
Tampoco tuvimos en cuenta el tiempo empleado para mover los camiones a la plataforma de carga, el tiempo de intercambio entre camiones y camionetas (cambio de contexto).

Usamos muchos “sleeps” para simular los tiempos empleados durante el trabajo, así como los descansos, aunque estos tiempos no se verán reflejados a la hora de los cálculos realizados por el planificador.

En cuanto a la medición de los tiempos, preferimos realizar una equivalencia de garrafas a minutos, que, si bien dista de la realidad de la situación, nos facilitó mucho los cálculos a realizar con nuestro planificador, aunque los resultados no reflejen la realidad.

Actualmente el planificador se realizó asimilando al ejemplo productor consumidor con un hilo productor (producir garrafas) y un hilo consumidor (consumir garrafas) respetando sus sincronizaciones en las secciones críticas cuando los camiones hacen su solicitud de carga. Para saber qué consumidor se va a atender se usó unos de los algoritmos de planificación visto en clase, FCFS (Primero en llegar, primero en ser atendido).

Diagramas que muestren los objetos involucrados en la solución, que variables y operaciones definen cada uno y qué interacción se da entre los mismos.



Alternativas de solución del problema, con su discusión; la elección de la alternativa a ser implementada; la implementación de dicha alternativa, restringida al medio ambiente de la solución.

En principio, la alternativa pensada fue hacer dos colas para camiones y camionetas respectivamente. El planificador entre colas usaría el algoritmo Round Robin y dentro de cada cola FCFS para respetar los turnos de llegada. Al ver que la implementación de esa alternativa

era bastante compleja decidimos realizar cálculos para tener una idea más clara de la diferencia de tiempo que ofrecía esa alternativa comparada a otros algoritmos cuya implementación sería más sencilla. Luego de realizados los cálculos vimos que FCFS no arrojaba tiempos tan malos en comparación a los demás, y sumado a eso, creíamos que era lo más justo.

El programa cuenta con el mismo hilo productor que la versión anterior, y un hilo por cada vehículo almacenados en un ArrayList. Estos se crean leyendo turno y capacidad de un archivo txt, se ordenan según ese turno para cargarlos usando FCFS.
Al final, el programa calcula tiempos de espera y espera promedio, expresados en garrafas y minutos respectivamente.

Los resultados de las simulaciones realizadas con su correspondiente análisis y conclusiones.

Los resultados de simular la llegada de los mismos camiones en distinto orden mostraron que lo que puede mejorar el tiempo de espera promedio es que las camionetas de capacidad más pequeña lleguen más temprano que los camiones de gran capacidad.

Conclusiones

Creemos que, si bien entendemos los conceptos tanto de sincronización como de planificación, no fuimos capaces de reflejarlos de la manera que hubiéramos querido en esta tarea. Nuestra versión final no implementa el algoritmo de planificación óptimo, ni varios algoritmos que permitan realizar una comparación partiendo del mismo juego de datos.

El problema constaba de varias partes, fue posible crear una analogía entre el clásico problema del productor-consumidor y el problema de la planta de garrafas. Se resolvió el problema del manejo de secciones críticas como lo son las cargas de los camiones, pero la planificación de las cargas no es responsabilidad de un hilo consumidor. A pesar de que el programa es funcional y hace en parte lo solicitado, no es exactamente lo que nos planteamos hacer en un principio.

Pecamos en intentar simular demasiado la realidad del problema, dejando de lado los puntos clave de la tarea sobre todo en lo que respecta a los planificadores.

Destacamos como positivos los conocimientos adquiridos durante el desarrollo y búsqueda de soluciones para esta tarea, sobre todo en lo que respecta a programación concurrente, un concepto nuevo para nosotros y que resultó de lo más interesante. Así como también la reafirmación de conceptos dados en clase gracias a la investigación hecha en internet durante el proceso de trabajo.

Sentimos también que debemos mejorar nuestra base de programación, que, si bien no es algo que se relacione directamente con el curso de sistemas operativos, influyó bastante en el desarrollo de este laboratorio.