PRÁCTICA 4
PROGRAMACIÓN
CON MONITORES

Autor: Rubén Rodríguez Esteban
NIP: 737215

Ejercicio 1:

Para poder resolver este ejercicio he creado un monitor gasolinera que cuenta con una variable condición **repostaje** para controlar la entrada y salida de los coches en los surtidores, es decir, dicha variable gestiona el posible bloqueo que los procesos pueden sufrir al intentar entrar en la gasolinera cuando los surtidores están ocupados. También me he creado un conjunto de variables permanentes propias del monitor (en el código del programa está comentado el papel de cada una de ellas), una cola asociada al monitor, y tres operaciones internas del monitor:

- **iniciarRepostaje**
- **finalizarRepostaje**
- **iniciarEstadistico**

En cuánto al control del final del programa, he creado una variable global de tipo entero llamada **vecesRepostados** que lleva la cuenta de las veces que han repostado en total todos los coches. Al ser un recurso compartido por los 20 procesos vehículo, podría ocurrir que dicha variable se viese modificada a la vez por 2 o más procesos a la vez, es decir, podrían surgir problemas de sincronización. No obstante, esta variable sólo se modifica en las operaciones internas del monitor, y cómo éstas se ejecutan en exclusión mutua es obvio que sólo va a poder modificarla un proceso cada vez.

De las dos operaciones del monitor tan sólo una de ellas es bloqueante, concretamente **iniciarRepostaje**. Para que un proceso cualquiera pueda entrar a repostar, deben cumplirse dos condiciones: ser el primero de la cola FIFO asociada al monitor, y haber al menos un surtidor libre. En caso contrario, se bloquea el proceso ejecutando un waitC sobre la variable condición **repostaje.** Así dicho proceso pasará a ser almacenado en la cola.

Si el coche entra a repostar, se procede a buscar uno de los surtidores libres de la gasolinera, de tal forma que se para de buscar cuando se halla encontrado el primero. Posteriormente se duerme hasta que termine de repostar.

En cuánto a la operación **finalizarRepostaje**, el primer punto importante es que en ningún momento es bloqueante, puesto que no hay ninguna restricción a que los coches salgan del surtidor en el que han repostado. Cuando uno de los procesos sale de la gasolinera, se avisa a todos los que están en la cola bloqueados de que un surtidor ha quedado libre haciendo un signalC_all sobre la variable condición **repostaje.**

Como consecuencia del uso de monitores, la implementación es mucho más sencilla dado que sus operaciones se ejecutan en exclusión mutua por definición, haciendo más fácil la gestión de los recursos que comparten, en este caso, los surtidores de la gasolinera.

En cuanto al proceso estadístico, no es bloqueante en ningún momento, no se necesitan variables condición para controlar su operatividad al igual que tampoco se requiere el uso de colas FIFO asociadas ya que el proceso estadístico entra y sale de la CPU cada cierto tiempo sin ninguna condición previa.

El esquema que sigue el proceso vehículo de acuerdo a mi diseño es el siguiente:

- entra a repostar si puede, sino se queda bloqueado → operación **iniciarRepostaje**
- Se duerme mientras está repostando
- Termina de repostar y avisa a los vehículos bloqueados → operación **finalizarRepostaje**

Ejercicio 2:

Para este ejercicio he reutilizado el monitor del anterior pero con ligeras modificaciones. En primer lugar he creado otra variable de condición llamada r**evision** para gestionar el control de la revisión de la gasolinera. También he creado nuevas variables permanentes del monitor, pero no una cola asociada a dicha variable porque en ningún momento pide que los procesos que se bloquean cuando se está revisando tengan más prioridad que los que se bloquean cuando los surtidores están llenos, es decir, no hay prioridad de unos sobre otros, por lo que ambas variables se pueden gestionar con una única cola FIFO. Además, a parte de las operaciones anteriores, el monitor integra dos operaciones nuevas: **iniciarRevision** y **finalizarRevision**, ambas para controlar las revisiones de la gasolinera.

Al igual que ocurría con la operación **iniciarRepostaje** vista en el primer ejercicio, la operación del monitor **iniciarRevision** también es bloqueante, por ello, debe ser gestionada de forma adecuada. Una revisión sólo puede comenzar cuando los surtidores están vacíos. Así las cosas, mientras al menos uno no lo esté, se ejecutará un waitC en la variable **revision** bloqueando así la revisión. Si se puede realizar la revisión, los coches no pueden acceder a repostar por lo que también deberá ser recogida en la condición de bloqueo en la variable de bloqueo **repostaje.** Cuando una revisión se termina se ejecuta un 4 instrucciones signalC_all para avisar a 4 coches distintos, puesto que ahora están todos los surtidores están libres. Sino se hiciese de esa forma, habría surtidores que estarían vacíos.

Obviamente también es necesario cambiar el código de la operación **finalizarRepostaje** dado que hay que llevar control de cuántos surtidores quedan libres después de que un coche sale de un surtidor concreto. Así las cosas, si sólo hay un surtidor ocupado y los otros restantes están libres, cuando salga el coche, se ejecuta un signalC sobre la variable condición revisión para indicar que se puede realizar una revisión de la gasolinera. En caso contrario, unicamente se avisa al resto de vehículos de que pueden repostar siguiendo el mecanismo del ejercicio anterior.

Los aspectos referidos al proceso estadístico son exactamente iguales a los que tenía en el primer ejercicio por lo que no es necesario hacer de nuevo énfasis en él.

Problemas encontrados a la hora de realizar la práctica

Para realizar esta práctica no he tenido muchos problemas ya que con las indicaciones vistas en clase y contando con gran parte del código procedente de la práctica 3 de semáforos ha sido bastante más rápido. No obstante, mi principal problema ha sido adaptar el concepto de monitor visto en clase y como se implementa en lenguaje C++ porque yo pensaba que la cola FIFO asociada ya se creaba cuando creabas la variable condición, pero no se tiene que gestionar a parte. Además, tenía otro error de concepto porque pensaba que la instrucción signalC de un monitor sólo llamaba a un proceso, el primero de la cola, pero en realidad llama a uno de la cola aleatorio. Aunque realmente no supone mucha diferencia hacerlo con una signalC o signalC_all. Por otro lado, en el enunciado tampoco pone muy detallado si es necesario que los coches sigan algún tipo de prioridad para entrar en la gasolinera así que he supuesto que ambos tenían la misma. Sería muy sencillo, simplemente bastaría con implementar otra cola distinta que recoja los coches bloqueados mientras se revisa la gasolinera. Entonces a la hora de repostar, se comprueba si hay primero coches en esa cola, se llaman a todos y si no pues se hace normal.