SCD (GIM+GIADE) – Curso 23-24 – Prácticas.

Ejercicios adicionales de la práctica 3.

Ejercicio P3.1. (gasolinera con un único surtidor)

Implementa con paso de mensajes en MPI el ejemplo de la gasolinera que hay descrito en los ejercicios adicionales de la práctica 2. Ahora todos los surtidores serán de gasolina y todos los coches funcionan con gasolina. Ten en cuenta que:

1. Por cada hebra que representa un coche en ese ejercicio de la práctica 2, ahora habrá un proceso con el mismo propósito.
2. Habrá un proceso adicional (lo llamamos *proceso gasolinera*), que se encarga de gestionar las variables que representan el estado de la gasolinera (el número de surtidores disponibles en cada momento).
3. Los procesos coche envían (de forma síncrona) un mensaje al proceso gasolinera para comenzar a repostar. Mediante este envío síncrono, el proceso coche debe quedar bloqueado hasta que no haya disponible al menos un surtidor.
4. Los procesos coche envían un mensaje (también síncrono) al proceso gasolinera para indicar que han terminado de repostar.

Todos los procesos ejecutan un bucle infinito.

Ejercicio P3.2. (gasolinera con tres tipos de surtidores).

Extiende el problema anterior para tener en cuenta ahora que la gasolinera sirve tres tipos de combustible en tres tipos de surtidores, y que cada proceso coche necesita un tipo de combustible distinto. Ten en cuenta esto:

1. Numeramos los tipos de combustible como 0,1 y 2.
2. La gasolinera tiene un numero fijo surtidores para cada tipo de combustible. Se usa un array de 3 enteros que no se modifica durante la ejecución (un array constante, incializado en su declaración). El total de surtidores debe ser menor que el número de coches.
3. Cada coche, al inicio del programa, decide aleatoriamente que tipo de combustible usa de entre los tres posibles.
4. La gasolinera no puede, en cada iteración, quedarse bloqueada a la espera de mensajes, como se hace en el ejemplo anterior. En lugar de esto, al inicio de su bucle, la gasolinera comprueba si hay mensajes de cada uno de los 4 tipos de mensajes que puede recibir, empezando con mensajes de fin de repostar, y luego mensajes de comienzo de repostar del combustible 0, luego del 1 y luego del 2 (lo comprueba por este orden). Para cada tipo de mensaje, si hay al menos un mensaje aceptable ya enviado de ese tipo, la gasolinera recibe y procesa el primero de dichos mensajes enviados. Si al final no hay ningún mensaje enviado aceptable de ningún tipo, espera bloqueada durante 20 milisegundos, y repite el bucle.
5. Ten en cuenta que la gasolinera no puede aceptar mensajes para inicio de repostar para un tipo de combustible si no quedan surtidores de ese tipo disponibles. Por supuesto, siempre puede aceptar mensajes de fin de repostar.

Ejercicio P3.3. (gasolinera de tres surtidores + proceso 'impresor')

Extiende el ejemplo de la gasolinera del ejercicio 2, de forma que haya un proceso nuevo que llamamos proceso *impresor*. Este proceso ejecuta un bucle infinito, y en cada iteración recibe un mensaje de cualquiera de los otros procesos (el envío es síncrono seguro). Ese mensaje lleva una cadena de texto, que será impresa por el impresor en el terminal inmediatamente tras ser recibida. Asegúrate de que:

- La memoria empleada para recibir cada cadena es estrictamente la necesaria para dicha cadena.

- Los procesos (distintos del proceso impresor), no escriben nada usando **cout**, en lugar de eso invocan la función **imprimir** (acepta un parámetro de tipo **const std::string &**, con una referencia a la cadena, de tipo string). Esa función es la encargada de enviarle la cadena al proceso impresor

La función **imprimir** debe usar el método **c_str** de la clase **string** para obtener un puntero (de tipo **const char ***) al array con los caracteres de la cadena.

Nota: para construir las cadenas que se pasan como parámetro a la función, se puede usar el tipo **string** y la función **to_string,** a modo de ejemplo, donde antes hacíamos:

```
cout « "El consumidor número " « num_cons « " ha consumido el valor " « valor « endl ;
```

ahora podemos hacer en su lugar:

```
imprimir( "El consumidor número " + to_string(num_cons) + " ha consumido el valor " + to_string(valor) );
```

donde suponemos que **imprimir** es la función que envía la cadena a la impresora. Esto requiere hacer *include* de **<string>,** además de **using namespace std;**