SISTEMAS CONCURRENTES Y DISTRIBUIDOS
Examen práctica 3 ()

APELLIDOS y NOMBRE:

1.- Construir un programa MPI cuyo nombre deberá ser filo_**ex.cpp**, con operaciones de paso de mensaje síncronas, modificando el problema de la cena de los filósofos con camarero de la siguiente manera:

- El filósofo con identificador 6 está un poco "rellenito" y necesita ocupar dos sillas de la mesa (suponemos que hay cinco sillas alrededor de la mesa, y los filósofos son libres de coger cualquiera que esté libre y llevarla a su puesto). Por tanto, si el filósofo 6 está sentado, podrían estar todas las sillas ocupadas (pero lógicamente, no pueden estar los cinco filósofos sentados)

- Cada vez que el camarero acepta la petición de sentarse y de levantarse de cualquier filósofo, debe indicar con un mensaje el número de sillas ocupadas que hay en ese momento.

- Cada vez que haya cuatro filósofos sentados a la mesa, el camarero debe indicar cuál es el primer filósofo que se levanta con el siguiente mensaje: *Estaban cuatro filosofos sentados y se ha levantado el filosofo X*

2.- En un supermercado hay tres cajas y 10 clientes (procesos 0 al 9). Los procesos clientes repiten un bucle de compra y resto de código. Para gestionar el pago, se utiliza un sistema de cola única que gestiona un proceso intermedio (proceso 10) que recibe las peticiones de pago de los clientes y controla la ocupación de las cajas para dar paso a dichos clientes (las cajas no son procesos, solo es necesario representar el número de cajas ocupadas/libres). Cuando el cliente finaliza la compra manda un mensaje al proceso intermedio para avisar que se ha quedado libre una caja. Construir un programa MPI, que se deberá llamar **examenp3.cpp**, con operaciones de paso de mensaje síncronas y que cumpla las características descritas de acuerdo al siguiente esquema (todos los procesos son bucles infinitos). **Incluir mensajes que permitan seguir la traza del programa**.

```
Proceso_cliente
   {retardo aleatorio} (compra en la tienda)
   ssend(intermedio,var) {solicitar pago}
   receive (intermedio,var)  {puede ir a una caja}
   {retardo aleatorio} (pago en la caja)
   ssend(intermedio, var)  {compra completa)
   {retardo aleatorio}  (resto de código del cliente)

Proceso_intermedio
        Si hay alguna caja libre
                esperar un mensaje de los clientes (de solicitar pago o de compra completa)
        En caso contrario (no hay cajas libres)
                esperar un mensaje de los clientes (sólo de compra completa)
        recibir_mensaje
        Si se ha recibido un mensaje de solicitar pago
                actualizar estado cajas (una caja libre menos)
                enviar mensaje a cliente  {se indica al cliente que puede ir a una caja}
        Si se ha recibido un mensaje de compra completa
                actualizar estado cajas (una caja libre más)
```