Sistemas Operativos II

Práctica 3 – Programación Multihilo
Sistema de venta online en Cines

Javier Alonso Albusac Jiménez
Escuela Superior de Informática :: Universidad de Castilla-La Mancha

En esta práctica se solicita al estudiante que desarrolle en el lenguaje de programación C++ un **sistema *online* de venta para cines**, tanto para la adquisición de *tickets,* como para las compra de bebida y palomitas. Para que un cliente pueda comprar bebida y palomitas, primero ha debido adquirir una serie de asientos (FASE 1).

Supongamos que el cine está constituido por una única sala y dispone de 72 asientos. La representación de la sala[1] se debe hacer mediante un *array* de 6 filas y 12 asientos por cada fila. Por otro lado, se modelará una aplicación cliente/servidor donde cada cliente solicita en primer lugar un determinado número de asientos. Por cada cliente se debe generar un hilo que represente su solicitud[2] (el control de turnos de los clientes se debe modelar mediante una cola FIFO). La generación de cada petición de cliente se realizará de forma aleatoria y constará de los siguientes elementos:

```
SOLICITUD ENTRADAS: <número_cliente> <número_asientos> <zona> <zona_horizontal>
```

Siendo *número_cliente* el identificador del cliente que representa el orden en el que ha llegado. Dicho identificador se utilizará para mostrar toda la información relativa al cliente. El *número_asientos* que demanda el cliente, *zona_vertical* la cual puede ser *lejos* (2 últimas filas), *medio* (2 filas intermedias), *cerca* (2 primeras filas). Por último, *zona_horizontal* que puede ser *izquierda* (4 primeras columnas), *medio* (las 4 columnas centrales) y *derecha* (4 columnas finales, situadas a la derecha). Si no se pudieran satisfacer las preferencias del cliente, el sistema debe asignar automáticamente unos asientos al cliente, cuya prioridad en la elección será definida por el propio estudiante.

Una vez seleccionados los asientos se debe proceder al pago. El **sistema de pago** por tarjeta es un recurso compartido por todos los clientes en el que se debe **garantizar exclusión mutua**. Es decir, solo un cliente puede hacer uso al mismo tiempo del sistema de pago. Cuando uno de ellos tenga asignado el recurso, el resto deberá esperar. Para evitar inconsistencias en la **asignación de asientos y pago de entradas**, **ambos recursos deberán ser adquiridos de forma simultánea**. Solo cuando un cliente tenga acceso exclusivo a los asientos y al sistema de pago, podrá hacer la selección y el pago de las entradas. En caso de no ser así, deberá esperar. Tenga en cuenta que si no se adquieren ambos recursos de forma simultánea, y se hace el bloqueo de forma individualizada con políticas no apropiativas, se podrían dar **situaciones de interbloqueo**.

Finalmente, cuando un cliente cuente con sus asientos y haya realizado el pago, debe liberar los recursos para que otro cliente pueda elegir asientos, o bien, otro cliente pueda efectuar el pago de la bebida y palomitas (FASE II).

El sistema debe mostrar por pantalla en todo momento cualquier evento relacionado con el cliente

1 La sala es un recurso compartido por todos los clientes.
2 Una simulación más apropiada consistiría en representar cada cliente mediante un proceso independiente y su petición como un hilo en el servidor. Con el fin de reducir la complejidad se hará una representación directa de las peticiones mediante hilos.

(siempre haciendo referencia a éste a través de su ID): cuándo llega un cliente, si se encuentra en estado de espera, cuándo adquiere el sistema de asientos y pago, la solicitud que realiza, si se puede atender a sus preferencias, qué asientos se le asignan (si es que se le puede asignar alguno) y cuándo efectúa el pago.

Una vez que un cliente dispone de sus entradas, la siguiente fase consiste en la **compra de bebida y palomitas**. Para la compra de alimentos existen **tres puntos de venta y una sola cola de espera**. Un cliente hará uso de un punto de venta cuando quede libre. Cada punto de venta tiene un número limitado de bebidas y palomitas. Cuando éstas se acaban será necesario despertar a un hilo **reponedor** (cuyo acceso debe ser exclusivo, al existir solo uno), encargado de restaurar el número de bebidas y palomitas en el punto de venta. Mientras el restaurador hace su trabajo, el cliente deberá esperar en el punto de venta. Cuando el *reponedor* finalice, éste debe notificar al cliente que puede continuar con su compra. La petición de un cliente en esta segunda fase consta de los siguientes campos:

```
SOLICITUD BEBIDAS/PALOMITAS: <número_cliente><numero_bebidas><numero_palomitas>
```

Por último, el cliente deberá esperar a que el sistema de pago con tarjeta esté libre para adquirirlo y efectuar el pago. Note que mientas un cliente efectúa el pago de sus bebidas y palomitas, ningún otro cliente podrá hacer ningún pago. Esto implica que el sistema de asignación de asientos deberá estar bloqueado y ningún otro cliente podrá pagar sus alimentos. Dada está situación el sistema debe gestionar un **mecanismo de prioridades** que otorgue con mayor frecuencia el sistema de pago a los clientes que quieren adquirir asientos. Es decir, se da mayor prioridad al acceso al cine que a la compra de bebidas y palomitas. Que se otorgue con mayor frecuencia el sistema de pago a los clientes de la entrada, no quiere decir que los clientes que se encuentran en la segunda fase puedan efectuar pagos mientras existan clientes en la primera, ya que en tal caso podría producirse un **problema de inanición**[3].

Tras el pago, el cliente entra a la sala y el hilo que representa su solicitud puede finalizar. De nuevo, en esta segunda fase se debe mostrar por pantalla en todo momento toda la información relativa al cliente y su petición.

3 Valore la posibilidad de utilizar la función std::thread::hardware_concurrency() para controlar el número de hilos, según las características de su máquina, y encontrar un buen equilibrio entre eficiencia y sobrecarga del sistema.

Evaluación de la práctica – Puntos de control

A continuación se listan una serie de puntos de control que se tendrán en cuenta en el proceso de evaluación para determinar qué objetivos se han alcanzado y en qué grado.

Nº	Puntos de control	Satisfecho/No Satisfecho
	FASE I	
1	Creación aleatoria de las solicitudes de los clientes.	☐
2	Control de hilos mediante *hardware_concurrency()*.	☐
3	Control de turnos para la entrada al cine mediante cola FIFO.	☐
4	Adquisición simultánea del sistema de asignación de asientos y el sistema de pago.	☐
5	Uso exclusivo de los dos recursos anteriores.	☐
6	Algoritmo para la asignación automática de asientos. ¿Todos los clientes consiguen sus entradas a menos que no haya asientos libres?. ¿Se atienden correctamente las preferencias del cliente?.	☐
7	Sin situaciones de interbloqueo en la adquisición de entradas.	☐
8	Nivel de detalle de la información mostrada por pantalla en Fase I.	☐
9	**FASE II**	
10	Gestión de única cola de espera para acceso a los puntos de venta.	☐
11	Acceso correcto a cada punto de venta libre y atención al cliente. ¿Se satisfacen las peticiones de todos los clientes?.	☐
12	Control de existencias en los puntos de venta (bebidas y palomitas).	☐
13	Hilo *Reponedor*, acceso exclusivo. ¿despierta correctamente? ¿Espera el cliente de forma adecuada a que se repongan las existencias?, ¿se retoma la compra para el mismo cliente?.	☐
14	Adquisición de sistema de pago de forma exclusiva.	☐
15	Mecanismo de prioridades para adquisición del sistema de pago, con mayor prioridad a la venta de entradas.	☐
16	¿Existe inanición?	☐
	OTROS	
17	Número de herramientas de sincronización empleadas (qué herramientas de sincronización de las vistas en teoría han sido empleadas para resolver la práctica).	☐
18	Calidad y adecuación de las estructuras de datos diseñadas para resolver el problema.	☐
19	Estructuración adecuada del código.	☐
20	Código limpio, claro, con alto grado de interpretabilidad.	☐
21	Uso de patrones de sincronización.	☐
22	Rendimiento global del sistema en términos de eficiencia temporal.	☐
23	Se hace una gestión adecuada de los errores.	☐
24	Se liberan los recursos correctamente al finalizar	☐
25	Creación de Makefile para facilitar la compilación de código.	☐
26	Incluye instrucciones README.txt para la compilación y ejecución del código.	☐
27	Organización adecuada en directorios de los archivos del proyecto: exec, obj, src e include.	☐