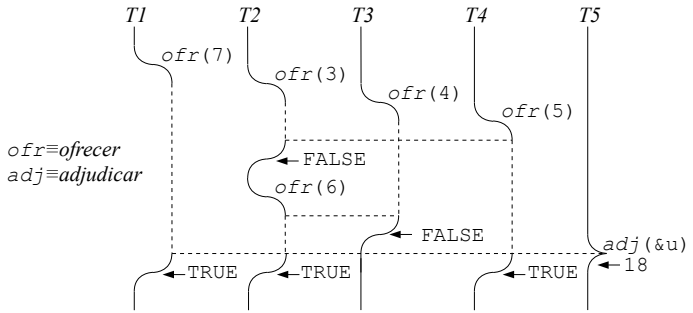
En esta tarea Ud. deberá programar un sistema de subastas en el que los vendedores y compradores son representados mediante threads.

En cada nueva subasta un vendedor ofrece n productos y espera un tiempo t, luego de lo cual adjudica los productos a los compradores que realizaron una oferta. Por otro lado, los compradores realizan una oferta por una unidad del producto a un valor *precio* y esperan a que se les adjudique la unidad del producto o que los saquen de la subasta.

La API que Ud. debe implementar utilizando spin-locks son las siguientes:

- *Subasta nuevaSubasta(int n)*: crea un nuevo sistema para subastar *n* unidades idénticas.
- *void destruirSubasta(Subasta s)*: libera los recursos de la subasta.
- *double adjudicar(Subasta s, int *prestantes)*: Cierra la subasta retornando el monto total recaudado y entregando en **prestantes* la cantidad de unidades sobrantes de la subasta, es decir las que no se vendieron porque llegaron menos que *n* oferentes.
- *int ofrecer(nSubasta s, double precio)*: múltiples tareas invocan esta función para hacer una oferta por comprar una unidad del producto al valor *precio*. Esta función espera hasta que (i) la subasta se cierre, retornando TRUE, o (ii) *n* otras tareas ofrecieron un *precio* mayor por el producto, y en tal caso se retorna FALSE.

El siguiente diagrama de threads muestra un ejemplo de ejecución considerando $N \equiv 3$:



Observe que cuando *T4* ofrece 5, hay 4 oferentes siendo *T2* la peor oferta con *precio*=3 y por lo tanto *T2* pierde retornando FALSE. Más tarde *T2* hace una contraoferta lo que transforma a *T3* en la peor oferta (*precio*=4) y retorna FALSE. Finalmente *T5* llama a *adjudicar* terminando la subasta. Los ganadores son *T1* que ofreció 7, *T2* con 6 y *T4* con 5, por lo que la recaudación asciende a 18. El valor final de *u* es 0 ya que se vendieron todas las unidades.

Programe las funciones *adjudicar* y *ofrecer* usando obligatoriamente esta metodología:

- Para programar la sincronización requerida Ud. debe usar spin-locks. No puede usar otras herramientas de sincronización como mutex/condiciones o semáforos. Tampoco puede crear nuevos threads con *pthread_create*.
- Puede usar cola de prioridades que está programada en pss.h y pss.c con las operaciones makePriQueue, priBest, priPeek, priGet, priPut, emptyPriQueue, priLength y destroyPriQueue.
- Use una estructura por cada subasta que contenga al menos el spin-lock que garantiza la exclusión mutua, el número de productos ofrecidos y el estado de la subasta.
- Use una por cada oferta que contenta al menos el spin-lock que garantiza la exclusión mutua, el precio de la oferta y el resultado de la oferta.

Prueba de la tarea: La verificación del funcionamiento correcto de su tarea se realizará primero implementando los spin-locks con mutex y condiciones, sin busy-waiting, y luego usando verdaderos spin-locks que sí esperan con busy-waiting y por lo tanto, se ocupa el 100% de la CPU. El primer método es útil para detectar dataraces.

Instrucciones

Descargue *t6.zip* de U-cursos y descomprímalo. Ejecute el comando *make* sin parámetros en el directorio *T6* para recibir instrucciones acerca del archivo en donde debe programar su solución (*subasta.c*), cómo compilar, probar y depurar su solución, los requisitos que debe cumplir para aprobar la tarea y cómo entregar su tarea por U-cursos.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *subasta.zip* generado por *make zip*. Este incluye *subasta.c* y *resultados.txt*. Recuerde descargar el archivo que subió, descargar nuevamente los archivos adjuntos y volver a probar la tarea tal cual como la subió a U-cursos. Solo así estará seguro de no haber entregado archivos incorrectos. Se descuenta medio punto por día de atraso. No se consideran los días de receso, sábado, domingo o festivos.