# CC4302 Sistemas Operativos – Control 1 – Semestre Primavera 2015 – Prof.: Luis Mateu

## Pregunta 1

***Parte a.-*** La siguiente es una implementación incorrecta de los lectores/escritores usando los semáforos de nSystem.

```
nSem wmutex; /* = nMakeSem(1); */
nSem rmutex; /* = nMakeSem(1); */
int readers= 0;
```

```
void enterWrite() {
  nWaitSem(wmutex);
}
```

```
void exitWrite() {
  nSignalSem(wmutex);
}
```

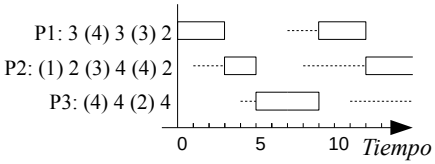
```
void enterRead() {
  if (readers==0)
    nWaitSem(wmutex);
  nWaitSem(rmutex);
  readers++;
  nSignalSem(rmutex);
}
```

```
void exitRead() {
  nWaitSem(rmutex);
  readers--;
  nSignalSem(rmutex);
  if (readers==0)
    nSignalSem(wmutex);
}
```

Haga un diagrama de threads que muestre que un lector puede entrar junto con un escritor.

***Parte b.-*** Haga el mínimo número de cambios posibles a la solución de la parte a de modo que funcione correctamente.

***Parte c.-*** El diagrama parcial muestra las decisiones de scheduling para 3 procesos. Para cada proceso se indica la duración de la ráfaga y la duración de su estado de espera entre paréntesis. En el diagrama la línea punteada indica que el estado del proceso es READY. Si el proceso está en estado de espera el espacio aparece en blanco. ¿De qué estrategia de scheduling se trata? Rehaga el diagrama completo considerando ahora la estrategia *shortest job first* non preemptive. Considere que el predictor de duración para la próxima ráfaga es la duración de la última ráfaga.



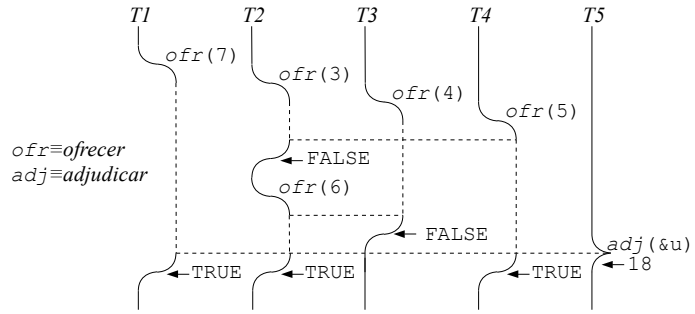

## Pregunta 2

Se requiere un sistema para subastar *N* unidades de un mismo producto. Para ello se le pide programar las siguientes funciones:

```
int ofrecer(double precio);
double adjudicar(int *punidades);
```

Múltiples threads invocan concurrentemente la función *ofrecer* para hacer una oferta por comprar una unidad del producto al valor *precio*. Esta función espera hasta que (i) se llame a la función *adjudicar*, en cuyo caso se retorna TRUE y el thread llamador es ganador, o (ii) *N* otros threads ofrecieron un precio mayor por el producto, y en tal caso se retorna FALSE indicando la calidad de perdedor.

Un solo thread invoca una sola vez la función *adjudicar* que determina los ganadores de la subasta haciendo que todas las llamadas pendientes de *ofrecer* retornen TRUE. La función *adjudicar* retorna el monto recaudado en la subasta equivalente a la suma de los precios ofrecidos por los ganadores. Además entrega en **punidades* el número de unidades vendidas. Si solo *p* threads invocaron *ofrecer* y *p<N,* entonces todos son ganadores y en tal caso **punidades*≡*p*. Nunca se excederá la cantidad de unidades disponibles *N*. El siguiente diagrama de threads muestra un ejemplo de ejecución considerando *N*≡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 3 ya que se vendieron todas las unidades.

Programe este sistema de subastas usando los monitores de nSystem.

Para resolver este problema Ud. puede usar las siguientes colas de prioridades:

- `PriQueue *MakePriQueue()`: entrega una nueva cola.
- `void PriPut(PriQueue *q, void *ptr, double pri)`: agrega el objeto *ptr* con prioridad *pri*.
- `void *PriGet(PriQueue *q)`: entrega y extrae el objeto cuya prioridad tiene el valor numérico menor.
- `int PriLength(PriQueue *q)`: entrega el largo de la cola.