Pregunta 1

Múltiples threads productores de oxígeno e hidrógeno deben formar moléculas de agua (H_2O):

```
void productorOxi() {
  for (;;) {
    Oxigen *o= ...;
    H2O *agua= combinarOxi(o);
    energia(agua);
} }
```

```
void productorHidro() {
  for (;;) {
    Hidrogeno *h= ...;
    H2O *agua= combinarHidro(h);
    condensar(agua);
} }
```

Por supuesto se deben combinar 2 átomos de hidrógeno con 1 átomo de oxígeno para formar H_2O. La siguiente es una implementación incorrecta de *combinarOxi* y *combinarHidro*, pero funciona cuando no hay invocaciones simultáneas (note que *barrera* parte con 1 ticket y los otros 2 semáforos con 0 tickets):

```
Hidrogeno *h1= NULL, *h2= NULL;
H2O *h2o;
nSem barrera; /* 1 ticket */
nSem sem_h2o; /* 0 tickets */
nSem sem_h;   /* 0 tickets */
H2O *combinarOxi(Oxigeno *o) {
  nWaitSem(barrera);
  nWaitSem(sem_h);
  nWaitSem(sem_h);
  h2o= makeH2O(o, h1, h2);
  h1= h2= NULL;
  nSignalSem(sem_h2o);
  nSignalSem(sem_h2o);
  nSignalSem(barrera);
  return h2o;
}
```

```
H2O *combinarHidro(Hidrogeno *h)
{
  while (h1!=NULL && h2!=NULL)
    ;
  if (h1==NULL)
    h1= h;
  else
    h2= h;
  nSignalSem(sem_h);
  nWaitSem(sem_h2o);
  H2O *res= h2o;
  return res;
}
```

Parte a.- Haga un diagrama de threads que muestre que bastan 2 invocaciones simultáneas de *combinarHidro* para que se pierda un átomo de hidrógeno.

Parte b.- Corrija esta solución manteniendo su espíritu. Es decir haga pequeñas modificaciones. Para ello use 2 semáforos adicionales. Asegúrese de que su solución no sufre del mismo problema de la parte a.- y que sea eficiente (elimine ciclos de busy-waiting).

Pregunta 2

En una calle hay 10 estacionamientos contiguos al lado derecho (y ninguno al lado izquierdo). Se identifican como 0, 1, 2, 3, …, 9. Las tareas de nSystem representan a los automovilistas que solicitan estacionar sus vehículos llamando a la función *reservar*. Esta función recibe como parámetro el número de estacionamientos contiguos requeridos (*k*). Un automóvil requiere solo 1 estacionamiento, una camioneta requiere 2 que estén contiguos, un camión 3 contiguos pero si tiene remolque podría requerir 5 estacionamientos. Si hay *k* estacionamientos contiguos disponibles, esta función los reserva y retorna de inmediato la identificación del primer estacionamiento en la serie otorgada (0, 1, 2, etc.). De lo contrario, *reservar* espera hasta que hayan *k* estacionamientos contiguos disponibles. Cuando un automovilista se va invoca la función *liberar* indicando el primer estacionamiento que ocupaba (*e*) y cuantos se le habían otorgado (*k*). Esta función libera todos los estacionamientos de la serie otorgada previamente a ese automovilista con *reservar*, y por lo tanto pueden ser otorgados a otros automovilistas.

Programe las funciones *reservar* y *liberar*. Sus encabezados son:

```
int reservar(int k);
void liberar(int e, k);
```

Restricciones: Debe atender a los automovilistas por orden de llegada. Debe evitar cambios de contexto innecesarios. Para la sincronización use un solo monitor de nSystem con múltiples condiciones.

Ayuda

Defina una estructura para representar una solicitud de estacionamiento. Incluya 3 campos: (i) una condición para esperar hasta que estén los estacionamientos disponibles, (ii) la cantidad *k* de estacionamientos solicitados, y (iii) la posición otorgada. Declare las siguientes variables globales y otras que necesite: (a) una *FifoQueue* para las solicitudes de estacionamiento, y (b) un arreglo de 10 valores booleanos que indican el estado de ocupación de cada estacionamiento.

Le será de utilidad programar una función *solicitar(k)* que busca en el arreglo (b) *k* estacionamientos libres consecutivos. Si encuentra los estacionamientos, esta función los marca como ocupados y entrega la posición del primero. De lo contrario entrega -1.

Suponga que los parámetros *e* y *k* suministrados a *liberar* son correctos.

Recuerde que *GetObj* extrae el primer elemento de una *FifoQueue*, *PutObj* agrega un elemento al final de la cola y *PushObj* agrega al principio.