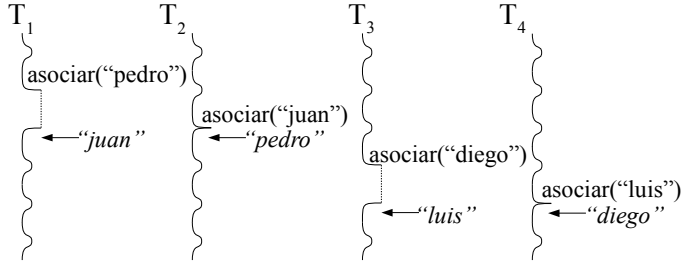
Pregunta 1

La función *asociar* debe obtener el nombre de un socio único. Recibe como argumento el nombre de la persona que busca socio y debe retornar el nombre de su socio. Al llamarla pueden ocurrir solo 2 casos: (i) no hay un socio pendiente, en cuyo caso el llamador queda como socio pendiente hasta que otro thread invoque *asociar*, o (ii) hay un socio pendiente, en cuyo caso el socio del llamador es el socio pendiente y viceversa, y no queda ningún socio pendiente. El siguiente diagrama muestra un ejemplo de ejecución:



La siguiente solución es incorrecta aunque funciona casi siempre:

```
nSem m; // = nMakeSem(1)
nSem w; // = nMakeSem(0)
```

```
char *socio;
int pend= FALSE;
```

```
char *asociar(char *nom) {
  char *mi_socio;
  nWaitSem(m); // secc. crítica
  if (!pend) {
    socio= nom;
    pend= TRUE;
    nSignalSem(m); // fin s.c.
    nWaitSem(w);
    mi_socio= socio;
  }
```

```
  else { // if (!pend)
    pend= FALSE;
    mi_socio= socio;
    socio= nom;
    nSignalSem(w);
    nSignalSem(m); // fin s.c.
  }
  return mi_socio;
}
```

Parte a.- Considere que llegan Pedro, Juan y Diego. Entran a la sección crítica en ese orden. Asociar le retorna a Juan que su socio es Pedro. Haga 2 diagramas de threads: en el primero muestre que podría ocurrir que asociar le retorna incorrectamente a Pedro que su socio es Diego, mientras que Diego queda en espera. En el segundo diagrama muestre que asociar podría hacer continuar incorrectamente a Diego y dejar esperando a Pedro.

Parte b.- Programe una solución de *asociar* que sea correcta y eficiente usando los semáforos de nSystem. *Ayuda:* (i) cada socio pendiente debe crear su propio semáforo para quedar en espera, y (ii) declare una variable global *pmi_socio* de tipo *char* **, un socio que queda pendiente almacena ahí la dirección de su variable local *mi_socio*, para que sea asignada en la próxima invocación de *asociar*.

Pregunta 2

Para evitar hambruna en el problema de los lectores/escritores se requiere programar la siguiente política para la aceptación de las solicitudes de ingreso de lectores y escritores.

I. Una solicitud de ingreso de un escritor (función *enterWrite*) se acepta de inmediato si no hay lectores o un escritor trabajando. De otro modo la solicitud queda pendiente.

II. Una solicitud de ingreso de un lector (función *enterRead)* se acepta de inmediato si no hay un escritor trabajando y *no hay solicitudes de escritores pendientes*. De lo contrario la solicitud queda pendiente.

III. Cuando termina de trabajar un escritor (función *exitWrite*) y hay solicitudes de lectores pendientes, se acepta el ingreso de todos los lectores pendientes. De lo contrario si hay solicitudes de escritores pendientes, se acepta el ingreso del escritor que lleva más tiempo esperando.

IV. Cuando termina de trabajar un lector (función *exitRead)* y ya no quedan otros lectores trabajando, si hay solicitudes de escritores pendientes, se acepta el ingreso del escritor que lleva más tiempo esperando.

Programe las funciones *enterWrite*, *enterRead*, *exitWrite* y *exitRead* de acuerdo a estos requerimientos. Para la sincronización Ud. debe usar los monitores de nSystem y múltiples condiciones. Para hacer esperar a los threads pendientes use colas FIFO de condiciones.

Observe que en I si nadie trabaja, no puede haber ninguna solicitud de un lector o escritor pendiente. En IV si no hay solicitudes de escritores pendientes no puede haber solicitudes de lectores pendientes.

Tenga cuidado en III y IV: al momento de aceptar una solicitud de ingreso registre en ese mismo instante cuantos lectores están trabajando o si hay un escritor trabajando.