CC41B : Sistemas Operativos
Control 1–Semestre Primavera'2002

Prof.: Luis Mateu.

Sin apuntes, 1 hora 30 minutos

Pregunta 1

Varios threads pueden esperar hasta que ocurra un cierto evento invocando el procedimiento `esperar()`. Un único thread notifica la ocurrencia del evento invocando el procedimiento `notificar()`. Si un thread invoca `esperar()` y el evento ya ocurrió, el thread continúa de inmediato (sin esperar). Un programador escribió la siguiente implementación basada en semáforos:

```
int ocurrio= FALSE;
int contEsperan= 0;
nSem sem; /* = nMakeSem(0); en el nMain */

void esperar() {       |    void notificar() {
  if (!ocurrio) {      |      ocurrio= TRUE;
    contEsperan++;     |      while (contEsperan>0) {
    nWaitSem(sem);     |        contEsperan--;
  }                    |        nSignalSem(sem);
}                      |    } }
```

- Parte a.- Utilizando un diagrama de threads, muestre que esta solución es incorrecta.
- Parte b.- Escriba la solución correcta para este problema. Emplee sólo semáforos en su solución (no puede recurrir a mensajes o monitores).

Pregunta 2

La municipalidad de Geek Island posee una barcaza para el transbordo de vehículos desde y hacia el continente. La barcaza tiene capacidad para llevar un solo vehículo. Los siguientes procedimientos describen la rutina de los isleños que viven en la isla y los turistas que la visitan:

```
int isleno() {              |    int turista() {
  for(;;) {                 |      for (;;) {
    cruzarAContinente();    |        cruzarAIsla();
    trabajar();             |        turistear();
    cruzarAIsla();          |        cruzarAContinente();
    dormir();               |        dormir();
  }                         |      }
}                           |    }
```

En donde `trabajar`, `turistear` y `dormir` son procedimientos dados. Para coordinar el uso adecuado de la barcaza, actualmente se usa la siguiente implementación de cruzarAContinente y cruzarAIsla:

```
nSem semBarcaza; /* = nMakeSem(1); en el nMain */
int enContinente= TRUE;

void cruzarAIsla() {          |   void cruzarAContinente();
  nWaitSem(semBarcaza);       |     nWaitSem(semBarcaza);
  if (!enContinente)          |     if (enContinente)
    navegarAContinente();     |       navegarAIsla();
  navegarAIsla();             |     navegarAContinente();
  enContinente= FALSE;        |     enContinente= TRUE;
  nSignalSem(semBarcaza);     |     nSignalSem(semBarcaza);
}                             |   }
```

En donde `navegarAContinente` y `navegarAIsla` son procedimientos dados y se demoran un tiempo considerable. Esta implementación no es eficiente porque puede ocurrir que habiendo un vehículo en el continente, la barcaza navegue vacía a la isla (y viceversa). No es de extrañar entonces que se formen filas de vehículos en espera de la barcaza en ambas orillas.

Utilizando el esquema cliente/servidor y el sistema de mensajes de nSystem, escriba una nueva implementación de `cruzarAIsla` y `cruzarAContinente` que permita aprovechar eficientemente la barcaza.

Observe que (i) la barcaza *no puede navegar* dos veces seguidas en el mismo sentido, (ii) un usuario que llama a `cruzarA...` *no puede continuar* mientras que su vehículo no haya sido transportado y la barcaza no haya llegado a la otra orilla, (iii) la barcaza *no puede transportar* un vehículo si éste no llegó a la orilla de partida (momento en que se invoca `cruzarA...`) antes que la barcaza parta, (iv) en cada viaje, si existen vehículos en espera en la orilla de partida, Ud. *debe transportar* un y sólo un vehículo en la barcaza y ese vehículo tiene que ser el que llegó primero a esa orilla, (v) Ud. *no debe detener* la barcaza si hay vehículos esperando en alguna de las orillas, (vi) Ud. *sí debe detener* la barcaza si no hay ningún vehículo que transportar en ninguna de las dos orillas.

```
        nSystem:                  |       FifoQueue:
                                  |
nTask nEmitTask(nProc, ... );     |  FifoQueue MakeFifoQueue();
int nSend(nTask task, void *msg); |  void PutObj(FifoQueue q, void* o);
void *nReceive(nTask *ptask,      |  void* GetObj(FifoQueue q);
               int max_delay);    |  int  EmptyFifoQueue(FifoQueue q);
void nReply(nTask task, int rc);  |
void nSleep(int delay);           |
int nGetTime();                   |
nTask nCurrentTask();             |
```