CC4302 Sistemas Operativos

Control 1 – Semestre Primavera 2013

Prof.: Luis Mateu

Pregunta 1

Un pub posee un único baño que debe ser compartido por damas y varones. El baño es amplio y admite un número ilimitado de personas. El problema consiste en evitar que las damas se encuentren con los varones dentro del baño. Las damas solicitan entrar al baño invocando *entrarDama* y notifican su salida con *salirDama*, mientras que los varones invocan *entrarVaron* y *salirVaron*. Se plantea la siguiente solución *incorrecta* para este problema:

```
int mutex= FALSE;
int damas= 0, varones= 0;
```

```
void entraDama() {
  if (damas==0) {
    while (mutex)
      ;
    mutex= TRUE;
  }
  damas++;
}

void saleDama() {
  damas--;
  if (damas==0)
    mutex= FALSE;
}
```

```
void entraVaron() {
  if (varones==0) {
    while (mutex)
      ;
    mutex= TRUE;
  }
  varones++;
}

void saleVaron() {
  varones--;
  if (varones==0)
    mutex= FALSE;
}
```

(a) Muestre mediante un *diagrama de threads* que una dama puede entrar al baño cuando hay varones presentes. Detalle bien las invocaciones de procedimientos y los instantes en que se hacen los if, se incrementan variables, etc.

(b) Escriba una *solución correcta y eficiente* para este problema utilizando 3 semáforos de nSystem (hint: utilice la estructura de la solución incorrecta). No importa que en su solución algunos procesos sufran "hambruna".

(c) Suponga que se agrega una nueva restricción: el baño tiene capacidad solo para 4 personas. Reescriba su solución para evitar que en algún momento hayan más de 4 personas en el baño. Use un semáforo adicional.

Pregunta 2

La federación de tenis de mesa de Chile organiza un torneo en donde todos los jugadores se enfrentan con todos. En el torneo participan 8 tenimesistas, numerados de 0 a 7, que se enfrentan en 7 rondas de 4 partidos cada una. La siguiente es una implementación secuencial del torneo, es decir usa una sola mesa:

```
struct { int x, y; } partidos[4*7]= {
  /* 7 rondas, 4 partidos por ronda */
  {0,4}, {1,5}, {2,6}, {3,7}, /* 1era ronda, */
  {0,1}, {2,4}, {3,5}, {7,6}, /* 2da ronda, */
  {0,2}, {3,1}, {7,4}, {6,5}, /* 3era ronda, */
  ... };  /* y así hasta la 7ma ronda */
```

```
void torneo(int puntajes[]) {
  int i, k;
  for (k= 0; k<8; k++)
    puntajes[k]= 0;
  for (i= 0; i<4*7; i++) {
    int x= partidos[i].x;
    int y= partidos[i].y;
    int ganador= enfrentar(x, y, 1); /* A */
    puntajes[ganador]++;
} }
```

El arreglo *partidos* es dado y almacena la programación de enfrentamientos. Su primer elemento {0, 4} indica que el primer partido es entre los tenimesistas 0 y 4. El segundo partido es entre 1 y 5, etc. En A, la función *enfrentar* inicia el partido entre los jugadores x e y en la mesa 1, y espera hasta que termine, retornando el ganador: x o y. Esta función puede tomar mucho o poco tiempo dependiendo de la duración del partido.

Parte a.- (4 puntos) Se le pide a Ud. reprogramar en nSystem la función *torneo* considerando que hay 3 mesas disponibles, numeradas como 1, 2 y 3.

Restricciones:

i. Un jugador puede participar en un solo partido a la vez.
ii. En una mesa *no pueden* haber 2 o más partidos simultáneos.
iii. Los partidos se deben iniciar en el mismo orden que en la versión secuencial.
iv. Si en un instante dado hay una mesa disponible y un partido se puede iniciar sin violar las restricciones anteriores, el partido debe comenzar de inmediato.
v. Cree solo 3 tareas adicionales (una por cada mesa) y un solo monitor. No puede usar semáforos o mensajes.
vi. La función *torneo* solo retorna una vez que terminaron los 28 partidos y se invocó *nWaitTask* para cada una de las 3 tareas creadas.

Al invocar enfrentar, no olvide verificar que ambos jugadores estén disponibles. Podría ocurrir que un tenimesista esté jugando todavía un partido de la ronda previa.

Parte b.- (2 puntos) Reprograme su solución considerando que la restricción iii. se cambiar por: *Ningún partido de la ronda R+1 podrá comenzar mientras quede algún partido de la ronda R sin comenzar.*

Esto significa que es posible que pueda iniciar un partido j aún cuando no ha iniciado un partido previo i, siempre y cuando i y j estén en la misma ronda.