Programación Concurrente
Práctico 1: Algunos Problemas de Concurrencia

J. Blanco, N. Wolovick

1 de abril de 2003

1. El siguiente programa sufre de deadlock. Intente mejorar su condición de progreso insertando sentencias $x := True$ en la componente A y $y := True$ en la componente B.

Pre: *true*	
$A : y := False;$ $\quad$ **if** $y \rightarrow$ **skip fi**	$B : x := False;$ $\quad$ **if** $x \rightarrow$ **skip fi**

   De argumentos convincentes que muestren que ambas componentes siempre terminan.

2. Una primitiva de sincronización importante es la *barrera*, también denominada *sincronización de inclusión mutua*.

   Operacionalmente una barrera entre n procesos se comporta bloqueando las componentes hasta que *todas* hayan entrado a ella, es decir que ningún proceso sale de la barrera hasta que todos hayan entrado.

   En el siguiente ejemplo nunca puede darse que la sentencia T_0 esté ejecutando mientras S_1 ó S_2 lo hacen.

P_0: S_0; "*barrera*"; T_0	P_1: S_1; "*barrera*"; T_1	P_2: S_2; "*barrera*"; T_2

   a) Implemente una barrera para $n = 2$ procesos utilizando **if** bloqueantes o **await**. Indicar la precondición necesaria y aclarar el grado de atomicidad utilizando la notación $< S_i >$.

   b) Generalice a cualquier n.

3. La siguiente es una solución utilizando semáforos del problema del Productor-Consumidor. Éste se dice correcto cuando el consumidor toma todos los datos que el productor genera[1]. Las únicas variables compartidas entre productores y consumidores son los semáforos y el arreglo b. La variable i se comparte entre los productores y j entre los consumidores.

Pre: $\langle \forall i : 0 \leq i < N : def(b.i) \rangle \wedge i = 0 \wedge j = 0 \wedge empty = N \wedge full = 0$	
$Prod$: $*[$ $x := produce();$ $P(empty);$ $b.i := x;$ $i := (i+1)\ mod\ N;$ $V(full)$ $]$	$Cons$: $*[$ $P(full);$ $x := b.j;$ $j := (j+1)\ mod\ N;$ $V(empty);$ $consume(x)$ $]$

[1]Pensar que $produce()$ genera la secuencia ordenada de todos los naturales y $consume(x)$ imprime su parámetro, luego la sincronización será correcta cuando el consumidor imprima de manera ordenada todos los naturales.

a) Para 1 productor y 1 consumidor, argumente si la solución es correcta. En caso de no serla muestre un contraejemplo y proponga una solución.

b) Idem al anterior solo que para n productores y m consumidores, con $n, m > 1$.

4. Se tienen n procesos del tipo A y m procesos del tipo B, con $0 < n, m$, que comparten los semáforos generales b y *done*. Analizar el tipo de sincronización que se realiza e indicar si existe riesgo de deadlock, en caso de que sea posible, encontrar una solución.

$A:$	$B:$
$\vdots$	$\vdots$
$P(b);$	$V(b);$
$P(b);$	$P(done);$
$V(done);$	$\vdots$
$V(done);$	
$\vdots$	

5. Implemente un semáforo general utilizando solamente semáforos binarios. Especifique el invariante de representación.

6. Se tiene un solo baño en la Facultad, el cual puede ser usado tanto por hombres como por mujeres, aunque no al mismo tiempo (unisex).

   Si representamos la cantidad de hombres y mujeres dentro del baño con nm, nw, la acción de ingresar un hombre se codifica con $nm := nm + 1$, idem para las otras 3 acciones.

   a) Especificar el problema a través de un invariante.

   b) Construir una solución usando monitores.

   c) Modificar la especificación y la solución si se agrega la restricción de que a lo sumo N personas puedan estar en el baño al mismo tiempo.

   d) Discuta que problemas de *fairness* tienen las soluciones obtenidas.