Programación Concurrente en Java
Práctico 0: Algunos Problemas de Concurrencia

J. Blanco, N. Wolovick

1. Indique que valores puede tomar la variable x en el siguiente multiprograma.

Pre: $x = 0$	
P_0: $*[\ x := x + 1$ $\quad ; x := x - 1$ $\quad]$	P_1: $*[\ x := x + 1$ $\quad ; x := x - 1$ $\quad]$

2. 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 : $y := False$; **if** $y \rightarrow$ **skip fi**	B : $x := False$; **if** $x \rightarrow$ **skip fi**

   De argumentos convincentes que muestren que ambas componentes siempre terminan.

3. 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. Indicar la precondición necesaria y aclarar el *grado de atomicidad* de todas las sentencias utilizando la notación $\langle S_i \rangle$.

   b) Generalice a cualquier n.

4. La siguiente es una solución al problema del *Productor-Consumidor* utilizando *semáforos generales*. 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 *empty* y *full* 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*: $*[$ $P(full)$; $x := b.j$; $j := (j + 1)\ mod\ N$; $V(empty)$; $consume(x)$ $]$

   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 $n, m > 1$.

5. Un *semáforo binario* es simplemente un semáforo general que solo puede tomar valores 0 y 1. Esta restricción está dada por la *topología del multiprograma*, como por ejemplo cuando utilizamos un semáforo para establecer *exclusión mutua*.

Pre: $lock = 1$
P_i: $*[$ NCS_i ; $P(lock)$; CS_i ; $V(lock)$ $]$

   Construya la primitiva de sincronización semáforo general utilizando semáforos binarios.

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 varones y mujeres dentro del baño con v, m, la acción de ingresar un hombre se codifica con $v := v + 1$, y de manera similar para el resto de las acciones.

   a) Especificar el problema a través de un *invariante*, es decir por una expresión lógica que involucra v, m y es verdadera *exactamente* cuando se respeta la condición del problema.

   b) Deducir la *precondición* que debe valer antes de cada sentencia de incremento y decremento de v, m para que luego de cada una el invariante se mantenga.

   c) Construir una solución usando semáforos generales.

   d) Discuta que problemas de *fairness* puede tener la solución obtenida en el punto 6*c*.

[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.