PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
2da práctica (tipo a)
(Primer semestre de 2013)

Horario 0781: prof. V.Khlebnikov
Horario 0782: prof. A.Bello R.

Duración: 1 h. 50 min.
Nota: No se puede usar ningún material de consulta.
La presentación, la ortografía y la gramática influirán en la calificación.
Puntaje total: 20 puntos

Pregunta 1 (7 puntos) (*Algorithms for Mutual Exclusion by M. Raynal*) "Finding a software solution to the mutual exclusion problem became something of a challenge to computer scientists in the sixties, leading to the publication of many false solutions, including the following which was proposed by Hyman (1966).

The two processes P_0 and P_1, which compete for access to their critical sections, share the following variables:

var $flag$: ***array*** 0 .. 1 ***of*** *boolean* ;
$turn$: 0 .. 1 ;

initialized respectively to false and to some arbitrary value.

The protocol proposed for P_i is then (where $i = 0$ or 1, $j = i + 1$ ***mod*** 2):

$flag\ [i] \leftarrow true$;
while $turn \neq i$ ***do***
while $flag\ [j]$ ***do*** *nothing* ***enddo*** ;
$turn \leftarrow i$;
enddo ;
$<$ *critical section* $>$;
$flag\ [i] \leftarrow false$;

a) (2 puntos) Explique el caso cuando solamente uno de 2 procesos pretende entrar en su sección crítica mientras que el otro no lo pretende. La variable *turn* puede tener el valor arbitrario.

b) (2 puntos) Explique el caso cuando ambos procesos, simultáneamente, pretenden entrar en sus secciones criticas.

c) (3 puntos) Encuentre el contraejemplo cuando la exclusión mutua no se cumple.

Pregunta 2 (6 puntos) Se desea implementar una solución al problema del productor/consumidor con *buffer* ilimitado, empleando únicamente semáforos binarios. La solución se presenta a continuación:

```
/* program productor_consumidor */

int n;
semaforo_binario s = 1;
semaforo_binario retraso = 0;
```

```
void productor() {
  while(cierto) {
     producir();
     waitB(s);
     añadir();
     n++;
     if(n == 1) signalB(retraso);
     signalB(s);
  }
}

void consumidor() {
  waitB(retraso);
  while(cierto) {
     waitB(s);
     tomar();
     n--;
     signalB(s);
     consumir();
     if(n == 0) waitB(retraso);
  }
}

void main()
{ n = 0;
  parbegin(productor, consumidor);
}
```

a) (3 puntos) La solución presentada es incorrecta. Presente una secuencia de ejecución donde se muestre claramente el error. Puede ayudarse de una tabla semejante a la siguiente:

	Productor	**Consumidor**	***s***	***n***	**retraso**
1			1	0	0
2	waitB(s)		0	0	0
3	...				

b) (3 puntos) Una vez ubicado el error corriga el programa anterior de forma que sea una solución correcta al problema del productor/consumidor con buffer ilimitado, empleando semáforos binarios.

Pregunta 3 (4 puntos) A continuación se tiene el problema del productor/consumidor con *buffer* limitado, empleando monitores:

```
/* program productor_consumidor */

monitor buffer_acotado;
char buffer[N];
int sigent, sigsal;
int contador;
condition no_lleno, no_vacio;
```

```
void añadir(char x) {
   if(contador == N) cwait(no_lleno);
   buffer[sigent] = x;
   sigent = (sigent + 1) % N;
   contador++;
   csignal(no_vacio);
}

void tomar(char x) {
   if(contador == 0) cwait(no_vacio);
   x = buffer[sigsal];
   sigsal = (sigsal + 1) % N;
   contador--;
   csignal(no_lleno);
}

{   sigent = 0;
    sigsal = 0;
    contador = 0;
}

void productor()
{ char x;
    while(cierto) {
       producir(x);
       añadir(x);
    }
}

void consumidor()
{ char x;
  while(cierto) {
    tomar(x);
    consumir(x);
  }
}
```

La definición que hace Hoare de los monitores exige que, si hay al mennos un proceso en una cola de condición, un proceso de dicha cola deberá ejecutarse en cuanto otro porceso ejecuta un *csignal* para la condición. Así pues, el proceso que ejecuta el *csignal* debe salir inmediatamente del monitor o suspenderse en el monitor.

a) (2 puntos) Mencione los inconvenientes de esta solución.

b) (2 puntos) Lampson y Redell desarrollaron una definición diferente de monitores para el lenguaje Mesa. En Mesa la primitiva *csignal* se reemplaza por *cnotify* con la siguiente interpretación: cuando un proceso que está en un monitor ejecuta *cnotify(x)*, origina una notificación hacia la cola de condición x, pero el proceso que da la señal puede continuar ejecutándiose. El resultado de la notificación es que el proceso que encabeza la cola de condición será reanudado en un futuro cercano, cuando el monitor esté disponible. Sin embargo, puesto que esto no garantiza que ningún otro proceso entre en el monitor antes que el proceso que espera debe volver a comprobar la condición. Bajo esta descrición modifique el programa mostrado antes para emplear la primitiva *cnotify*.

Pregunta 4 (3 puntos) Un sistema de comunicación entre procesos está formado por *canales*, por eso el primer argumento en las primitivas *send()* y *receive()* es el nombre del canal. Hay 5 canales *forks[5]*: *forks[0]*, ..., *forks[4]*. Se lanzan 5 procesos de filósofos: *Phil(0, forks[0], forks[1])*, *Phil(1, forks[1], forks[2])*, *Phil(2, forks[2], forks[3])*, *Phil(3, forks[3], forks[4])*, *Phil(4, forks[4], forks[0])*. También se lanzan 5 procesos de recursos (tenedores): *Fork(forks[0])*, *Fork(forks[1])*, *Fork(forks[2])*, *Fork(forks[3])*, *Fork(forks[4])*. Presente los códigos de ambos procesos correspondientes a la solución del problema de los filósofos comensales usando la comunicación por canales.

La práctica fue preparada por AB(2,3) y VK(1,4)

Profesores del curso: V.Khlebnikov
A.Bello R.

Pando, 23 de abril de 2013