PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
2da práctica (tipo a)
(Primer semestre de 2016)

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 (4 puntos – 20 min.) (*PCDPAM by Ben-Ari*) ***Race Conditions***. Sea *a*, *b*, *c* tres arreglos de enteros ordenados; se sabe que al menos un elemento aparece en cada uno de los tres arreglos. A continuación se presenta un esquema del algoritmo secuencial para encontrar el índice más pequeño *i*, *j*, *k*, para el cual *a*[*i*] = *b*[*j*] = *c*[*k*]:

```
                integer array[0..N] a, b, c ← . . . (as required)
                integer i ← 0, j ← 0, k ← 0
        loop
p1:       if condition-1
p2:         i ← i + 1
p3:       else if condition-2
p4:         j ← j + 1
p5:       else if condition-3
p6:         k ← k + 1
          else exit loop
```

a) (1.5 puntos – 7.5 minutos) Escriba la expresión condicional que hace el algoritmo correcto.

b) (2.5 puntos – 12.5 minutos) Desarrolle un algoritmo concurrente para este problema.

Pregunta 2 (4 puntos – 20 min.) (*Manna and Pnueli*) Considere en los siguientes algoritmos que la líneas de código (P_i, Q_i) son atómicas:

```
integer wantp=0, wantq=0;

Hilo P                                                  Hilo Q

    while (TRUE) {                                          while (TRUE) {
P1:     non-critical section;                           Q1:     non-critical section;
P2:     if (wantq == -1) wantp=-1 else wantp=1;         Q2:     if (wantp == -1) wantq=1 else wantq=-1;
P3:     while (wantq == wantp);                         Q3:     while (wantp == -wantq);
P4:     critical section;                               Q4:     critical section;
P5:     wantp=0;                                        Q5:     wantq=0;
    }                                                       }
```

a) (1 punto – 5 min.) En el escenario (“escenario” - “5. Posibilidades o perspectivas de un hecho o de una situación”, según RAE) cuando primero se ejecutan las líneas 1-4 de un hilo y después otro hilo intenta ejecutar las mismas líneas, ¿qué valores tendrán las variables `wantp` y `wantq`? Considere todos los casos.

b) (1 punto – 5 min.) En el escenario de la ejecución estrictamente intercalada, cuando a una línea de un hilo sigue una línea de otro hilo, ¿qué valores tendrán las variables `wantp` y `wantq` hasta el momento cuando un hilo entre a su sección crítica? Considere todos los casos.

c) (2 puntos – 10 min.) En caso de que la sentencia `if` no es atómica y se divide en 3 líneas (como está presentado en el siguiente código), existe un escenario cuando la exclusión mutua ya no se cumple. Encuentre este escenario y preséntelo como la secuencia de los números de las líneas ejecutadas.

```
integer wantp=0, wantq=0;

Hilo P                                          Hilo Q

    while (TRUE) {                                  while (TRUE) {
P1:     non-critical section;                   Q1:     non-critical section;
P2:     if (wantq == -1)                        Q2:     if (wantp == -1)
P3:         wantp=-1;                           Q3:         wantq=1;
P4:     else wantp=1;                           Q4:     else wantq=-1;
P5:     while (wantq == wantp);                 Q5:     while (wantp == -wantq);
P6:     critical section;                       Q6:     critical section;
P7:     wantp=0;                                Q7:     wantq=0;
    }                                               }
```

Pregunta 3 (6 puntos – 30 min.) (*PCDP2 by M. Ben-Ari*) Cuando un semáforo `S` está definido como la tupla de un entero no-negativo `S.V` y un conjunto de procesos bloqueados en este semáforo `S.L`, y cuando la operación `signal(S)` (en caso cuando hay procesos bloqueados en el semáforo) libera un proceso *arbitrario* del conjunto, el problema de la sección crítica (*the critical section problem*) para 2 procesos (**`p`** y **`q`**) se resuelve de la siguiente manera:

```
binary semaphore S ← (1,{})     (valor inicial 1, conjunto inicial vacío)

p                                               q
        loop forever                                    loop forever
p1:         non-critical section                q1:         non-critical section
p2:         wait(S)                             q2:         wait(S)
p3:         critical section                    q3:         critical section
p4:         signal(S)                           q4:         signal(S)
```

Esta solución garantiza la exclusión mutua (*mutual exclusion*), la ausencia de *deadlock*, y también garantiza la ausencia de inanición (*starvation*) porque si ambos procesos intentan entrar en sus secciones críticas, uno lo logrará y otro se bloqueará en el semáforo. Al salir de su sección crítica el proceso ejecutará `signal(S)` y, como en el conjunto `S.L` está un único proceso bloqueado, el proceso bloqueado se activará y entrará en su sección crítica.

Para *N* procesos esta solución sigue garantizando la exclusión mutua y la ausencia de *deadlock*, pero ya no garantiza la ausencia de inanición porque, seleccionando para desbloquear un proceso *arbitrario* en el conjunto `S.L`, puede suceder que un proceso siempre se ignora y él sufrirá la inanición.

Aquí está la solución libre de inanición desarrollada por Jan T. Udding:

```
semaphore gate1 ← 1, gate2 ← 0
integer numGate1 ← 0, numGate2 ← 0

        loop forever
            non-critical section
 1:         wait(gate1)
 2:         numGate1 ← numGate1 + 1
 3:         signal(gate1)

 4:         wait(gate1)                 // First gate
 5:         numGate2 ← numGate2 + 1
 6:         numGate1 ← numGate1 - 1
 7:         if numGate1 > 0
 8:             signal(gate1)
 9:         else signal(gate2)

 10:        wait(gate2)                 // Second gate
 11:        numGate2 ← numGate2 - 1

            critical section
 12:        if numGate2 > 0
 13:            signal(gate2)
 14:        else signal(gate1)
```

a) (3 puntos – 15 min.) Presente el escenario de ejecución para *un proceso* de la siguiente manera: nos interesan solo las operaciones sobre semáforos, estados bloqueados de los procesos y estados en CS (*critical section*), por eso solamente estos se indican en las filas (las filas son *estados*); en la 2da columna se indica la sentencia todavía *no ejecutada* sino que *será ejecutada*; el nuevo valor asignado a la variable es parte del *siguiente estado* que se encuentra en la siguiente fila:

n	Process p	gate1	gate2	nGate1	nGate2
1	p1: wait(gate1)	1	0	0	0
2	p3: signal(gate1)	0	0	1	0
3	p4: wait(gate1)	1	0	1	0
4	p9: signal(gate2)	0	0	0	1
5	...	...	...	...	...

b) (3 puntos – 15 min.) Presente un escenario para *2 procesos*. Marque con círculo la operación que será ejecutada (aquí, en la tabla, estará con **negrita**).

n	Process p	Process q	gate1	gate2	nGate1	nGate2
1	**p1: wait(gate1)**	q1: wait(gate1)	1	0	0	0
2	**p3: signal(gate1)**	q1: blocked	0	0	1	0
3	p4: blocked	**q3: signal(gate1)**	0	0	2	0
4	**p8: signal(gate1)**	q4: blocked	0	0	1	1
5	...	...	...	...	...	...

<u>Pregunta 4</u> (6 puntos – 30 min.) (*Concurrent Programming by Tom Aderson*) You have been hired by Pro Transporte to synchronize traffic over a narrow light-duty bridge on a public highway. Traffic may only cross the bridge in one direction at a time, and if there are ever more than 3 vehicles on the bridge at one time, it will collapse under their weight. In this system, each car is represented by one thread, which executes the procedure `OneVehicle` when it arrive at the bridge:

```
OneVehicle(int direc) {
   ArriveBridge(direc);
   CrossBridge(direc);
   ExitBridge(direc);
}
```

In the code above, `direc` is either 0 or 1; it gives the direction in which the vehicle will cross the bridge.

a) (4 puntos – 20 minutos) Write the procedures `ArriveBridge` and `ExitBridge` (the `CrossBridge` procedure should just print out a debug message), using monitors. `ArriveBridge` must no return until it safe for the car to cross the bridge in the given direction (it must guarantee that there will be no head-on collisions or bridge collapses). `ExitBridge` is called to indicate that the caller has finished crossing the bridge; `ExitBridge` should take steps to let additional cars cross the bridge. This a lightly-travelled rural bridge, so you do not need to guarantee fairness or freedom starvation.

b) (2 puntos – 10 minutos) In your solution, if a car arrives while traffic is currently moving in its direction of travel across the bridge, but there is another car already waiting to cross in the opposite direction, will the new arrival cross *before* the car waiting on the other side, *after* the car on the other side, or is it impossible to say? Explain briefly.

La práctica ha sido preparada por AB(1,4) y VK(2,3)
con LibreOffice Writer en Linux Mint 17.3 Rosa.

Profesor del curso: (0781) V. Khlebnikov
(0782) A. Bello R.

Pando, 22 de abril de 2016