PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

2da práctica (tipo a)
(Primer semestre de 2012)

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 (3 puntos) (*Steve Carr, Jean Mayo and Ching-Kuang Shene - Race Conditions: A Case Study*) Se tienen dos grupos de hilos, A y B, que intercambian mensajes. Cada hilo en A ejecuta la función `T_A()`, y cada hilo en B ejecuta la función `T_B()`. Tanto `T_A()` como `T_B()` tiene un lazo infinito y nunca se detienen. Cuando una instancia de A crea un mensaje disponible, este puede continuar solo sí este recibe un mensaje de la instancia de B quien ha recuperado satisfactoriamente un mensaje de A. De forma similar, cuando una instancia de B crea un mensaje disponible, este puede continuar solo sí este recibe un mensaje de una instancia de A quien ha recuperado satisfactoriamente un mensaje de B. Cuando se hable de intercambio de mensajes no se hace referencia a las primitivas *send/receive* sino al hecho de que un valor (mensaje) creado por A sea visto por B y viceversa.

a) (1 punto) Escriba un programa que implemente el intercambio de mensajes entre A y B tal como se ha descrito arriba.

b) (2 puntos) Explique con un ejemplo cuáles son los problemas de *race conditions* que se presentan.

Pregunta 2 (3 puntos) (*M.Raynal, Algorithms for Mutual Exclusion, 2.3*) Dijkstra generalizó la solución de Dekker para el caso de n procesos. Las variables compartidas entre los n procesos $P_0, ..., P_i, ..., P_{n-1}$ son:

var $flag$: **array**$[0..n-1]$ **of** $(passive, requesting, in\text{-}cs)$;
$turn : 0..n-1$;

Los elementos de $flag$ están inicializadas a *passive*, y $turn$ toma un valor arbitrario. Cada proceso P_i tiene una variable entera j.

repeat
$flag[i] \leftarrow requesting$;
while $turn \neq i$ **do if** $flag[turn] = passive$ **then** $turn \leftarrow i$ **endif**; **enddo**;
$flag[i] \leftarrow in\text{-}cs$;
$j \leftarrow 0$;
while $(j < n) \wedge (j = i \vee flag[j] \neq in\text{-}cs)$ **do** $j \leftarrow j + 1$ **enddo**;
until $j \geq n$;
$<$ *critical section* $>$;
$flag[i] \leftarrow passive$;

Si se ejecutan los procesos P_0, P_1, P_2 (en este orden) y las interrupciones suceden después de cada operación (la asignación, la comparación, la conjunción, etc.), explique el comportamiento de cada proceso hasta que todos ellos entren en su secciones críticas.

Pregunta 3 (7 puntos) (*A. Casillas R., L. Iglesias V. - Ejer. 3.4.3*) Hay tres procesos: A, B y C con tres partes secuenciales de su código: $A1, A2, A3$ en A, $B1, B2, B3$ en B, y $C1, C2, C3$ en C. Se necesita sincronizarlos de tal manera que
$A2$ se ejecute después de $B1$, $A3$ se ejecute después de $C2$;
$B2$ se ejecute después de $A1$, $B3$ se ejecute después de $C1$; y
$C2$ se ejecute después de $B2$, $C3$ se ejecute después de $A2$.

Para evitar la espera ocupada de cambios de las variables, la solución debe ser con semáforos.

a) (1 punto) ¿Cuántos semáforos se necesitan?

b) (2 puntos $*$ 3**)** Presente la solución.

Pregunta 4 (3 puntos) Monitores

a) (3 puntos) Se desea simular semáforos usando monitores. En el siguiente código complete las líneas que faltan de modo que `monP` y `monV` se comporten como semáforos.

```
/* Monitor implementation of semaphore. */
 monitor monSemaphore {
     int semvalue;
     condition notbusy;

     void monP() {
        . . .
     }

     void monV() {
        . . .
     }

     init{ semvalue = 1; }
 } // end of monSemaphore monitor
```

```
int n;
void inc(int i) {
   monP();
   n = n + 1;
   monV();
}

void main() {
   cobegin {
     inc(1); inc(2);
   }
}
```

b) (4 puntos) Se tiene una fábrica de gaseosas. En el interior del taller se encuentran tres robots trabajando todo el día. Un robot está continuamente fabricando tapas de botella, que va depositando en un recipiente. El recipiente tiene una capacidad limitada: cuando se llena, el robot deja de producir tapas hasta que haya espacio libre en el recipiente. Otro robot está continuamente fabricando botellas, que también deposita en su correspondiente recipiente de capacidad limitada. Un tercer robot se encarga de tomar una botella envasar la gaseosa y colocarle la tapa de forma hermética, tomando en cada caso una botella y una tapa de los correspondientes recipientes.

Se pide escribir el código haciendo uso de monitores para sincronizar a estos tres robots, de modo que los dos primeros robots no avancen si su recipiente está llena, y que el tercer robot no avance si le faltan piezas para envasar una gaseosa. Las capacidades de los dos recipientes son constantes y distintas, por ejemplo constantes enteras `NumMaxTapas` y `NumMaxBotellas`.

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

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

Pando, 18 de abril de 2012