

Sistemas Operativos – GII & GMI

Cuarto Parcial. Sincronización y Comunicación. 23 de mayo de 2016

Problema 1

Dado el siguiente código:

```
1 #include <pthread.h>
2 pthread_mutex_t mutex;
3 pthread_cond_t cond_1_2, cond_1_3, cond_2_3;
4 int control1=1, control2=1, control3=1, datos=0;
5
6 void Thread_codA(void){
7     pthread_mutex_lock(&mutex);
8     while ((control1<=0)||control2<=0)
9         pthread_cond_wait(&cond_1_2, &mutex);
10    control1--;
11    control2--;
12    pthread_mutex_unlock(&mutex);
13    datos++;
14    pthread_mutex_lock(&mutex);
15    control1++;
16    control2++;
17    pthread_cond_signal(&cond_1_2);
18    pthread_cond_signal(&cond_1_3);
19    pthread_cond_signal(&cond_2_3);
20    pthread_mutex_unlock(&mutex);
21 }
22
23 void Thread_codB(void){
24    pthread_mutex_lock(&mutex);
25    while ((control1<=0)||control3<=0)
26        pthread_cond_wait(&cond_1_3, &mutex);
27    control1--;
28    control3--;
29    pthread_mutex_unlock(&mutex);
30    datos--;
31    pthread_mutex_lock(&mutex);
32    control1++;
33    control3++;
34    pthread_cond_signal(&cond_1_3);
35    pthread_cond_signal(&cond_1_2);
36    pthread_cond_signal(&cond_2_3);
37    pthread_mutex_unlock(&mutex);
38 }
39
40 void Thread_codC(void){
41    pthread_mutex_lock(&mutex);
42    while ((control2<=0)||control3<=0)
43        pthread_cond_wait(&cond_2_3, &mutex);
44    control2--;
45    control3--;
46    pthread_mutex_unlock(&mutex);
47    datos=datos*2;
48    pthread_mutex_lock(&mutex);
49    control2++;
50    control3++;
51    pthread_cond_signal(&cond_2_3);
52    pthread_cond_signal(&cond_1_2);
53    pthread_cond_signal(&cond_1_3);
54    pthread_mutex_unlock(&mutex);
55 }
```

```

56
57 int main(void){
58     pthread_t thA1, thA2, thB1, thB2, thC1, thC2;
59     pthread_mutex_init(&mutex, NULL);
60     pthread_cond_init(&cond_1_2, NULL);
61     pthread_cond_init(&cond_1_3, NULL);
62     pthread_cond_init(&cond_2_3, NULL);
63     pthread_create(&thA1, NULL, Thread_codA, NULL);
64     pthread_create(&thA2, NULL, Thread_codA, NULL);
65     pthread_create(&thB1, NULL, Thread_codB, NULL);
66     pthread_create(&thB2, NULL, Thread_codB, NULL);
67     pthread_create(&thC1, NULL, Thread_codC, NULL);
68     pthread_create(&thC2, NULL, Thread_codC, NULL);
69     pthread_join(thA1, NULL); pthread_join(thA2, NULL);
70     pthread_join(thB1, NULL); pthread_join(thB2, NULL);
71     pthread_join(thC1, NULL); pthread_join(thC2, NULL);
72     pthread_mutex_destroy(&mutex);
73     pthread_cond_destroy(&cond_1_2);
74     pthread_cond_destroy(&cond_1_3);
75     pthread_cond_destroy(&cond_2_3);
76     return 0;
77 }

```

Responder de manera **justificada** las siguientes preguntas:

a) [1 pto] ¿Cuántos threads pueden ejecutar simultáneamente las líneas **13, 30 y 47**?

Sólo podría ejecutar un thread, dado que se trata de la sección crítica del código, cuyo acceso es controlado a través del mecanismo de m \acute{u} tex m \acute{a} s variables condicionales. El m \acute{u} tex se utiliza para modificar y consultar las variables de control control1, control2 y control3 y dichas variables establecen las condiciones para evitar que m \acute{a} s de un thread entre concurrentemente a ejecutar dichas l \acute{i} neas.

b) [1 pto] ¿En qu \acute{e} orden ejecutar \acute{i} an dichas l \acute{i} neas los threads existentes? ¿Cu \acute{a} l es el rango (valor m \acute{i} nimo, valor m \acute{a} ximo) que puede tomar la variable datos una vez ejecutado el programa, dependiendo del orden de ejecuci \acute{o} n de los threads?

No se conoce ni se puede presuponer ning \acute{u} n orden de ejecuci \acute{o} n.

El valor m \acute{a} ximo que podr \acute{i} a tomar la variable datos es: 6, si se ejecutan en este orden las l \acute{i} neas: 13,13,47,47,30,30.

El valor m \acute{i} nimo que podr \acute{i} a tomar la variable datos es: -6, si se ejecutan en este orden las l \acute{i} neas: 30,30,47,47,13,13.

c) [1 pto] ¿Cu \acute{a} l es el rango (valor m \acute{i} nimo, valor m \acute{a} ximo) que pueden tomar las variables control1, control2 y control3 durante la ejecuci \acute{o} n de los threads?

Estas variables de control s \acute{o} lo pueden tomar valores 0 y 1, dado que el thread que decrementa su valor una vez obtenido el m \acute{u} tex es el \acute{u} nico que puede continuar su c \acute{o} digo sin quedarse bloqueado, y por tanto volver a poner las variables a 1, increment \acute{a} ndolas.

d) [1 pto] ¿Cu \acute{a} l ser \acute{i} a la diferencia en el comportamiento del programa si sustituy \acute{e} ramos todas las invocaciones a pthread_cond_signal() por pthread_cond_broadcast()?

La diferencia es que la invocaci \acute{o} n a pthread_cond_broadcast() despertaría a todos los threads que estuvieran esperando en dicha condici \acute{o} n, frente a pthread_cond_signal(), que s \acute{o} lo despierta a uno de los threads bloqueados. Sin embargo, el comportamiento del programa no se modificar \acute{i} a, dado que todos los threads volver \acute{i} an a competir por el m \acute{u} tex, siendo s \acute{o} lo uno el que lo obtendr \acute{i} a, modificando las variables de control y dejando al otro thread de nuevo bloqueado.

e) [1 pto] Si las variables de control (control1, control2, control3) pudieran inicializarse a los valores 0 ó 1, ¿en qué combinaciones de estos valores para dichas variables el programa siempre finalizaría y en cuáles podría no finalizar?

Sólo en el caso de que todas las variables estén inicializadas a 1, el programa finalizaría, ya que no se quedaría ningún thread eternamente bloqueado en una condición.

Si alguna de las variables de control estuviera a 0, al menos 2 tipos de threads se quedarían bloqueados eternamente (ningún thread incrementaría esa variable de control, por lo que la condición de bloqueo siempre se cumpliría).

Cuestiones

Responder brevemente a las siguientes preguntas:

- 1) [1 pto]** ¿Cuál es la diferencia entre coordinación de acceso a recursos implícita y explícita? Indicar además algún ejemplo de cada caso.
- 2) [1 pto]** Considérese un proceso servidor que se comunica con uno o más clientes usando sockets. El servidor registra los mensajes (cadenas de caracteres) que recibe de cada cliente y los escribe en un fichero local, anotando además la fecha y hora de recepción. ¿Se podría implementar este servidor usando sockets datagrama? ¿Qué ventajas e inconvenientes tendría? ¿Y si usamos sockets stream?
- 3) [1 pto]** ¿Cuál es la diferencia entre un cerrojo compartido y uno exclusivo? ¿Se puede establecer un cerrojo compartido sobre una región de un fichero en la que ya hay un cerrojo exclusivo? ¿Y viceversa?
- 4) [1 pto]** Considérese un grupo de tres procesos pesados (A, B y C) coordinados mediante un semáforo. El proceso A realiza un `sem_wait` sobre el semáforo cuando éste vale 0, quedando bloqueado. Dos segundos después, el proceso B realiza `sem_wait` sobre el mismo semáforo, quedando igualmente bloqueado. Al cabo de unos instantes, el proceso C realiza un `sem_post`, incrementando el contador del semáforo. En ese momento ¿Se desbloqueará algún proceso? ¿Cuál? Razonar la respuesta.
- 5) [1 pto]** Explicar las diferencias entre un PIPE y un FIFO, e indicar en qué contexto es preferible usar cada uno.