Boletín 3.1
Ejercicios sobre mutex y variables de condición

July 14, 2016

1. ¿Para qué sirve un mutex? ¿Y una variable de condición?
2. Responde a las siguientes preguntas (sin necesidad de justificarlas):
   (a) ¿Se bloquea un hilo cuando éste adquiere un mutex (con una operación `lock`)?
   (b) ¿Se bloquea un hilo cuando intenta (con una operación `lock`) bloquear un mutex que ya está bloqueado?
   (c) ¿Se bloquea siempre un hilo cuando ejecuta una operación `wait` sobre una variable de condición?
   (d) ¿Tiene algún efecto una operación `signal` si no hay hilos esperando en la cola variable de condición sobre la que se aplica?
3. La solución dada en las transparencias 39 y 40 del tema 3 resuelve el problema que se plantea en el ejemplo de las transparencias 11 y 12.
   (a) ¿Qué ocurre si ponemos en el código que ejecutan los hilos subordinados, `hilo1` e `hilo2`, la función `pthread_mutex_unlock` detrás de `sleep`?
   (b) ¿Qué ocurre si extraemos las funciones `pthread_mutex_lock` y `pthread_mutex_unlock` fuera del bucle `for`?

4. ¿Por qué la función `pthread_cond_wait` del estándar POSIX requiere un mutex como segundo argumento?

5. Sea el siguiente fragmento de código que ejecuta un hilo determinado:

```
1: pthread_mutex_lock(&mutex);
2: while (ocupado == true)
3:    pthread_cond_wait(&cond, &mutex);
4: ocupado = true;
5: pthread_mutex_unlock(&mutex);
```

   Y sea el siguiente fragmento de código que permite desbloquear al hilo que ejecute el código anterior:

```
6: pthread_mutex_lock(&mutex);
7: ocupado = false;
8: pthread_cond_signal(&cond);
9: pthread_mutex_unlock(&mutex);
```

   Supongamos que en un instante determinado el valor de la variable `ocupado` es `true` y que en el sistema se están ejecutando dos hilos que denotamos por A y B, respectivamente. Indicar de manera justificada si serían posibles y por qué las siguientes secuencias de ejecución, donde X_i denota que el hilo *X* ejecuta la instrucción *i*.

   (a) A1 A2 A3 B6 B7 B8 A2 A4 A5 B9

   (b) A1 B6 A2 A3 B7 B8 B9 A4 A5

   (c) A1 A2 B6 A3 B7 B8 B9 A2 A4 A5

   (d) B6 A1 B7 B8 B9 A2 A4 A5

   (e) B6 B7 A1 A2 A4 A5 B8 B9

6. En el ejercicio 2 de las transparencias del tema 3 se plantean dos posibles soluciones para dicho ejercicio. La primera es correcta y la segunda, no. ¿Por qué no es correcta la segunda solución?

7. Sea el siguiente programa escrito en C:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NTHREADS 4

int contador=0;
```

```
void *hilo(void *arg)
{
  int  i;

  i =  (int) arg;
  contador = contador + i;
  printf("Hilo %u, contador = %u\n",pthread_self(),contador);
  pthread_exit(0);
}
int main()
{
  int i;
  pthread_t thread[NTHREADS]; //vector para definir los hilos

  for (i=0;i<NTHREADS;i++)
  {   pthread_create(&thread[i], NULL, hilo, (void *)i);
      pthread_join(thread[i],NULL);
  }
  printf("Hilo principal, contador = %d\n",contador);
  pthread_exit (0);
}
```

(a) Indica, de manera justificada, lo que muestra por la pantalla tanto la hebra principal como cada uno de los hilos subordinados.

(b) Modifica este código para que todos los hilos puedan ejecutarse en paralelo (concurrentemente) y acceder a la variable `contador` de manera correcta.

8. Realiza un programa en C que calcule la suma de los elementos de un vector. El hilo principal deberá crear un número determinado hilos y repartir la suma del vector entre todos ellos. La función que ejecuta cada hilo debe recibir como parámetros el índice inicial y final del subvector a sumar. Y acumulará la suma de los elementos del subvector que le corresponde sobre una variable accesible por todos los hilos.

   Tanto el tamaño del vector como el número de hilos a crear serán constantes. Y los elementos del vector se calcularán de manera aleatoria.

   Para realizar el ejercicio puedes tomar como referencia el ejercicio 13 de las transparencias del tema 2.

9. ¿Qué ocurriría en la solución planteada para el ejercicio 3 de las transparencias del tema 3 si se eliminan en las funciones `f_hiloA` y `f_hiloB` la función `pthread_mutex_unlock` que precede inmediatamente al primer `printf` y la función `pthread_mutex_lock` que sigue inmediatamente al segundo `printf`?

10. En las transparencias del tema 3 se plantea una posible solución al problema del productor-consumidor con buffer limitado mediante mutex y variables de condición. A partir de ella, responde a las preguntas que aparecen a continuación:

    (a) ¿Se puede sustituir en la función `Productor` la sentencia `while` por un `if`? ¿Y en la función `Consumidor`?

    (b) ¿Sería necesario sustituir en la función `Productor` la función

    ```
    pthread_cond_signal(&no_vacio);
    ```

    por lo siguiente:

    ```
    if (n_elementos==1) pthread_cond_signal(&no_vacio);
    ```

    (c) ¿Qué cambios habría que hacer en el código de las transparencias para que hubiese dos hilos productores y un hilo consumidor?

    (d) ¿Se podría sustituir en la función `Productor` del código del apartado anterior la sentencia `while` por un `if`? ¿Y en la función `Consumidor`?

11. Realiza el ejercicio 4 de las trasparencias del tema 3 de la asignatura.

12. Se pretende que los hilos del código que aparece a continuación se sincronicen de tal forma que se imprima en primer lugar la primera fila de la matriz, a continuación la segunda, después la tercera y así sucesivamente hasta la última fila de la matriz. ¿Se realizan dichas operaciones con la ejecución concurrente de los hilos que se crean en el programa? En caso negativo, ¿cómo modificarías el código para conseguirlo?
    Justifica la respuesta o respuestas.

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define MAX_DIM 10
int n=4;
typedef int TMat [MAX_DIM][MAX_DIM];
TMat A={{1,2,5,1},{2,3,8,0},{3,8,5,1},{4,0,4,1}};
int turno=0;
pthread_mutex_t mutex;
pthread_cond_t  TocaTurnoFila;

void *Impr (void *arg)
{ int i;
  int f;

  f=*(int *)arg;
  pthread_mutex_lock(&mutex);
  while ( turno != f )
     pthread_cond_wait(&TocaTurnoFila,&mutex);
```

```
  printf("Fila %d (Hilo %u):  ", f, pthread_self());
  for (i=0; i<n; i++) printf("%d  ", A[f][i]);
  printf("\n");
  turno++;
  pthread_cond_signal (&TocaTurnoFila);
  pthread_mutex_unlock(&mutex);
  pthread_exit(0);
}
main()
{
  int i,j;
  int v[MAX_DIM];
  pthread_t hilo[MAX_DIM];

  pthread_mutex_init(&mutex, NULL);
  pthread_cond_init(&TocaTurnoFila,NULL);

  for (i=0; i<n; i++)
  { v[i]=i;
    pthread_create(&hilo[i], NULL,  Impr, (void *)&v[i]);
  }
  for (i=0; i<n; i++) pthread_join(hilo[i], NULL);

  pthread_cond_destroy(&TocaTurnoFila);
  pthread_mutex_destroy(&mutex);

  exit(0);
}
```

13. En el programa que aparece más abajo se crean cinco hilos simulando a cinco comensales que están sentados en una mesa circular en el mismo orden en el que fueron creados. En el centro de la mesa hay un plato de arroz y un cucharón para servirlo. Los comensales se sirven el arroz de uno en uno y en el orden en que están sentados a la mesa; esto es, comensal 0, 1, 2, 3, 4, 0, 1, ... Los hilos sincronizan su ejecución de la forma arriba descrita utilizando mutex y variables de condición,

    Responder a las siguientes preguntas sobre este programa:

    (a) ¿Por qué no se incluye la función `sleep`, que simula el periodo en el que come un comensal, dentro del mutex?

    (b) El hilo principal crea los hilos en un bucle y espera a que finalicen en otro bucle. ¿Por qué no se realizan ambas operaciones en un único bucle?

    (c) ¿Qué ocurre si se reemplaza la función `pthread_cond_signal` por `pthread_cond_broadcast`?

    (d) Modifica la solución proporcionada para que se utilice una única variable de condición para la sincronización.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

#define NHILOS  5

void *f_hilos(void *ind)
{ int indice;
  indice = (int)ind;

  while(1) {
      printf("Comensal %d: Sirve su comida\n",indice);
      sleep(random() % 3); /* Comensal ind comiendo */
      printf("Comensal %d: Plato vacio\n",indice);
  }
  pthread_exit(0);
}
main()
{ int i;
  pthread_t *hilos;
  time_t t;
  srandom(time(&t));

  hilos = malloc(NHILOS * sizeof(pthread_t));
  for (i = 0; i < NHILOS; i++) pthread_create(&hilos[i], NULL, f_hilos, (voi
  for (i = 0; i < NHILOS; i++) pthread_join(hilos[i], NULL);
  pthread_exit(0);
}
```

14. El programa que aparece más abajo crea tantos hilos como núcleos ("cores") tiene la máquina. Cada hilo ejecuta un determinado código en un núcleo, mediante la función `ejecuta`, y almacena el resultado en un fichero accesible por todos los hilos. Modifica este programa para que varios hilos puedan ejecutar la función `ejecuta` simultáneamente. Además, han de sincronizar su acceso al fichero que comparten de manera que los hilos guarden en él los resultados proporcionados por la función `ejecuta` en el siguiente orden: *hilo 0, hilo 1, ..., hilo num_hilos -1, hilo 0, hilo 1, ...*.

    Utiliza mutex y variables de condición como herramienta de sincronización. Fíjate cómo se crean los hilos en el código de partida. La implementación ha de ser lo más eficiente posible.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define N_CORES 8

int num_hilos, fd;

int ejecuta (int hilo, int n_core)
{ int resultado;
  ...   // Ejecuta en n_core un código y devuelve resultado
  return resultado;
}

void escribe_en_fichero (int iteracion, int n_core, int resultado)
{  ...  // Escribe en  fd
}

void *f_hilo(void *indice)
{ int i, n_core, resultado;
  n_core = (int)indice;

  for (i=0; i<10; i++)
  {   resultado = ejecuta (i, n_core);
      escribe_en_fichero  (i, n_core, resultado);
  }
  pthread_exit(0);
}

main()
{ int i, tamanyo, dato;
  pthread_t *mis_hilos;

  fd=open("fichero",O_WRONLY);
  num_hilos = N_CORES;
  mis_hilos = malloc(num_hilos * sizeof(pthread_t));
  for (i = 0; i < num_hilos; i++)
  {   pthread_create(&mis_hilos[i], NULL, f_hilo, (void *)i);
      pthread_join(mis_hilos[i], NULL);
  }
  close(fd);
  pthread_exit(0);
}
```

15. Se desea imprimir las tablas de multiplicar del 1 al 10 de tal manera que éstas se muestren respectivamente en 10 columnas consecutivas. Por lo tanto, en la primer línea se imprimirá el primer elemento de cada una de las sucesivas tablas (esto es,

`1x1, 2x1, ..., 10x1`). En la segunda línea se imprimirá el segundo elemento de cada una de las sucesivas tablas (es decir, `1x2, 2x2, ..., 10x2`). Y así hasta llegar a imprimir (en la décima línea) el último elemento de cada tabla (esto es, `1x10, 2x10, ..., 10x10`).

Para llevar esto a cabo se crearán 10 hilos, uno por tabla, que se ejecutarán concurrentemente. Y la sincronización entre hilos se realizará mediante mutex y variables de condición. Todos los hilos ejecutarán una misma función a la que se le pasará como argumento el número de su correspondiente tabla. El hilo principal finalizará cuando se hayan impreso todas las tablas.

La implementación y la ejecución han de ser lo más eficientemente posibles.

Una posible salida del programa podría ser la siguiente:

```
$ tablas_multiplicar
1x1=1     2x1=2      ...  10x1=10
1x2=2     2x2=4      ...  10x2=20
1x3=3     2x3=6      ...  10x3=30
...
1x10=10   2x10=20    ...  10x10=100
$
```

16. Resuelve el ejercicio 2 de las transparencias del tema 3 utilizando una única variable de condición.

17. El siguiente programa no funciona correctamente. Analiza lo que debe hacer y soluciónalo usando mutex y variables de condición. Se deberá colocar en la sección crítica (esto es, dentro del mutex) sólo las instrucciones estrictamente imprescindibles. Se valorará la eficiencia de la solución propuesta.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXBUF 10
int BUF [MAXBUF];
int valor=0;

void *A (void *arg){
  int i, j;
  for (i=0; i<MAXBUF; i++)
  { scanf("%d",&j);
    BUF[i]=j;
    valor++;
  }
  pthread_exit(0);
}
void *B (void *arg){
```

```
  int i, j;
  for (i=0; i<MAXBUF; i++)
  { j=BUF[i];
    valor--;
    printf("Leido %d\n",j);
  }
  pthread_exit(0);
}
int main(){
    int v=0;
    pthread_t thread_A, thread_B;
    pthread_create (&thread_A, NULL, A, NULL);
    pthread_create (&thread_B, NULL, B, NULL);
    pthread_join (thread_A,NULL);
    pthread_join (thread_B,NULL);
    pthread_exit(0);
}
```

18. Indica de forma razonada la veracidad o falsedad de las siguientes afirmaciones:

    (a) El hilo `thread_A` del código del ejercicio anterior tiene acceso tanto a la variable `v` definida en la función `main` como a la variable `valor` definida justo antes de la función `A`.

    (b) Cuando se crea un hilo el contador de programa del nuevo hilo apunta a la instrucción siguiente a la función `pthread_create` con la que fue creado.

    (c) Cuando se crea un hilo se le asigna una pila de ejecución diferente a la del hilo que lo creó.

19. Explica qué cambios de estado se pueden producir en un hilo cuando este ejecuta cada una de las siguientes funciones:

    (a) `pthread_create`

    (b) `pthread_self`

    (c) `pthread_cond_wait`

    (d) `pthread_join`.

20. Una serie de hilos utilizan simultaneamente un fichero. Unos acceden al fichero sin modificarlo y otros, modificándolo. A los primeros hilos les denominaremos *lectores* y a los segundos, *escritores*. Para acceder al fichero se establecen las siguientes restricciones:

    - Sólo un escritor puede tener acceso al fichero al mismo tiempo. Mientras el escritor esté accediendo al fichero, ningún otro hilo lector o escritor podrá acceder a él.
    - Se permite, sin embargo, que múltiples lectores tengan acceso al fichero, ya que ellos nunca van a modificar el contenido del mismo.

Modifica el programa que aparece a continuación para que los hilos lectores y escritores sincronicen su ejecución de la forma arriba descrita utilizando mutex y variables de condición.

No es necesario implementar la lectura y la escritura en el fichero. Estas operaciones se simulan ya en el código de partida con la función `sleep(random()%3)`.

Asumir que siempre hay datos para leer en el fichero y espacio suficiente para escribir en él.

Se valorará la eficiencia de la solución propuesta.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define N_HILOS  10

int n_lectores, n_escritores;

void *f_lector (void *indice)
{ int i, n_lector, resultado;
  n_lector = (int)indice;
  sleep(random()%5);

  printf("Lector %d va a leer\n",n_lector);
  sleep(random()%3); /* Simula acceso lectura fichero */
  printf("Lector %d deja de leer\n",n_lector);

  pthread_exit(0);
}

void *f_escritor (void *indice)
{ int i, n_escritor, resultado;
  n_escritor= (int)indice;
  sleep(random()%3);

  printf("Escritor %d va a escribir\n",n_escritor);
  sleep(random()%3); /* Simula acceso escritura fichero */
  printf("Escritor %d deja de escribir\n",n_escritor);

  pthread_exit(0);
}

main()
{ int i;
```

```
    pthread_t *hilos_lectores, *hilos_escritores;

    time_t t;
    srandom(time(&t));

    hilos_lectores =   malloc(N_HILOS * sizeof(pthread_t));
    hilos_escritores = malloc(N_HILOS * sizeof(pthread_t));

    for (i = 0; i < N_HILOS; i++)
        pthread_create(&hilos_lectores[i], NULL, f_lector, (void *)i);
    for (i = 0; i < N_HILOS; i++)
        pthread_create(&hilos_escritores[i], NULL, f_escritor, (void *)i);
    for (i = 0; i < N_HILOS; i++) pthread_join(hilos_lectores[i], NULL);
    for (i = 0; i < N_HILOS; i++) pthread_join(hilos_escritores[i], NULL);

    pthread_exit(0);
}
```

21. Realiza un programa en C que cree un hilo productor y 2 hilos consumidores y resuelva el problema del productor-consumidor con buffer circular limitado mediante mutex y variables de condición. Se valorará la eficiencia de la solución propuesta.

22. Sea la solución propuesta en las transparencias del tema 3 para el ejercicio 3. Modifícala para que incluya, además, la restricción de que a lo sumo haya 5 hilos del mismo tipo accediendo a los datos.

23. Implementa un programa en lenguaje C que cree 10 hilos de tipo "sumador" y otros 10 hilos de tipo "restador". Los *hilos sumadores* sumarán uno a una variable global `v`, realizando cada uno esta operación 1000 veces. Los *hilos restadores* restarán uno a la variable global `v` y cada uno realizará esta operación 1000 veces. Se incluye la restricción de que el valor de la variable `v` no puede ser nunca negativo. Utiliza mutex y variables de condición para sincronizar la ejecución de los hilos.

24. Implementa un programa en lenguaje C que cree 10 hilos de tipo "sumador" y otros 10 hilos de tipo "restador". Los *hilos sumadores* sumarán uno a una variable global `v`, realizando cada uno esta operación 1000 veces. Los *hilos restadores* restarán uno a la variable global `v` y cada uno realizará esta operación 1000 veces. Todos los hilos se ejecutarán concurrentemente. Se incluye la restricción de que no se pueden realizar dos operaciones iguales de forma consecutiva. O sea, si hace una suma, la siguiente operación debe ser una resta y viceversa. Utiliza mutex y variables de condición para sincronizar la ejecución de los hilos.

25. Implementa un programa en lenguaje C que cree 10 hilos de tipo "h1", otros 10 hilos de tipo "h2" y otros 10 hilos de tipo "h3". Todos los hilos se ejecutarán concurrentemente. Por otro lado, en el sistema hay dos recursos compartidos, A y B. Los *hilos h1* utilizarán el recurso A, los *hilos h2* utilizarán el recurso B y los *hilos h3* utilizarán simultaneamente los recursos A y B. Tanto el recurso A como el B deben ser usados en exclusión mutua, esto es, solo un hilo puede acceder de

forma simultánea al recurso. Por tanto, un hilo "h1" puede acceder al recurso A mientras un hilo "h2" está accediendo al recurso B, sin embargo, cuando un hilo "h3" esté accediendo a los recursos A y B los hilos "h1" y "h2" no podrán acceder respectivamente a sus recursos.
Notas:
(1) El uso del recurso puede simularse poninendo: sleep(random() % 3);
(2) Utiliza mutex y variables de condición para sincronizar la ejecución de los hilos.