Boletín de ejercicios 2.2
Ejercicios sobre gestión básica de hilos

July 13, 2016

1. Implementa un programa en C que calcule el factorial de los números que se le pasan como argumentos. El programa creará tantos hilos como argumentos se introduzcan y cada uno de los hilos recibirá uno de esos argumentos y calculará su factorial. No importa que se mezcle la salida que imprimen los hilos.

   Para pasar el argumento al hilo puedes tomar como referencia el ejemplo 4 de las tansparencias. Pero recuerda que cada hilo ha de recibir como argumento el **número** cuyo factorial debe calcular.

   Un ejemplo de la ejecución del programa podría ser la siguiente:

```
$ factorial 3 6 4 1 0
Factorial de 1: 1
Factorial de 0: 1
Factorial de 4: 24
Factorial de 6: 720
Factorial de 3: 6
$
```

2. Modifica el programa anterior para que no se mezclen las impresiones de los diferentes hilos. Una posible solución consistiría en que cada hilo devolviese al hilo principal, mediante la función `pthread_exit`, el resultado que ha calculado y que el hilo principal lo imprima a continuación.

   Puedes tomar como referencia el ejemplo 6 de las tansparencias.

3. Implementa un programa que calcule una secuencia de potencias de 2. El programa creará tantos hilos como se especifique en su único argumento. Y cada hilo calculará la potencia de 2 de un número que se le pasa como argumento. El primer hilo recibirá 0 como argumento y calculará 2^0. El segundo hilo, 2^1. El tercero, 2^2 y así sucesivamente. El hilo principal mostrará por pantalla el resultado calculado por cada hilo subordinado.

   Puedes tomar como referencia los ejemplos 4 y 6 de las tansparencias.

   Un ejemplo de la ejecución del programa podría ser la siguiente:

```
$ pot2 5
2 ^ 0: 1
2 ^ 1: 2
2 ^ 2: 4
2 ^ 3: 8
2 ^ 4: 16
$
```

4. Implementa un programa que genere un vector de MAX números enteros aleatorios comprendidos entre 0 y 9. A continuación buscará secuencialmente en dicho vector el número que se le pasa como primer argumento. El programa debe crear tantos hilos como se le indique en el segundo argumento. Y cada hilo ha de realizar la búsqueda en un fragmento consecutivo del vector de un tamaño aproximadamente igual para todos los hilos. Así, si denotamos por `N` al número total de hilos a crear, el primero de ellos buscará en las posiciones `0,1,...,(MAX div N)-1`, el segundo en `(MAX div N),(MAX div N)+1,...,2*(MAX div N)-1` y así sucesivamente. Una vez que un hilo encuentre el número, todos los hilos finalizarán la búsqueda. El hilo principal esperará a que acaben todos los hilos y mostrará por pantalla el índice de una posición en la que esté el número ó -1 si el número no está en el vector.

   Puedes tomar como referencia los ejercicios 13 y 14 de las tansparencias así como el siguiente programa, que genera un vector de 20 números aleatorios:

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

#define MAX 20

int main()
```

```
{ time_t t;
  int i, v[MAX];

  srandom(time(&t));    // Genera semilla para no crear siempre
                        // los mismos números aleatorios
  for (i=0;i<MAX;i++)
  { v[i]=random()%10;  // Crea un número aleatorio entre 0 y 9
    printf("V[%d]=%d\n",i,v[i]);
  }
  exit(0);
}
```

Un ejemplo de la ejecución del programa podría ser la siguiente:

```
$ $1 7 5
VECTOR
V[0]=3
V[1]=8
  ...
V[15]=8
V[16]=7
V[17]=8
V[18]=6
V[19]=2

Hilo 0: Buscará entre las componenetes 0 y 3
Hilo 1: Buscará entre las componenetes 4 y 7
Hilo 2: Buscará entre las componenetes 8 y 11
Hilo 3: Buscará entre las componenetes 12 y 15
Hilo 4: Buscará entre las componenetes 16 y 19
****Hilo 3035712368: Encuentra el número en V[16]
El número 7 está en la posición 16 (V[16]=7)
$
```

5. Sea el siguiente programa:

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

#define NUMTHREADS 4

typedef struct interval{
   int iter;
   int ini;
   int fin;
} t_interval;
```

```
int suma_parcial[NUMTHREADS];
int suma = 0;

void *hilo(void *arg)
{ int  i;
  int ini1,fin1, itera;
  t_interval *inter;

  inter = ((t_interval *) (arg));
  itera = inter->iter;
  ini1 = inter->ini;
  fin1 = inter->fin;

  suma_parcial[itera] = 0;
  for(i=ini1;i<=fin1;i++){
   suma_parcial[itera] = suma_parcial[itera] + i;
  }
  printf("HIlo i=%d, suma = %d\n",itera,suma_parcial[itera]);
  pthread_exit(0);
}

int main()
{
   pthread_t thread[NUMTHREADS];
   int ini,fin;
   int i,elem,inicio;
   t_interval param[NUMTHREADS];

   printf("Introduce inicio intervalo: ");
   scanf("%d",&ini);
   printf("Introduce fin intervalo: ");
   scanf("%d",&fin);

   elem = ((fin-ini)+1)/NUMTHREADS;

   inicio=ini;
   for (i=0;i<NUMTHREADS;i++)
   {
     param[i].iter = i;
     param[i].ini = inicio;
     if (i==(NUMTHREADS-1)) param[i].fin = fin;
     else param[i].fin = param[i].ini + elem;

     pthread_create(&thread[i], NULL, hilo, (void *) &param[i]);
     printf("hilo 1 iter = %d, ini = %d, fin = %d\n",param[i].iter,
             param[i].ini,param[i].fin);
```

```
      inicio = param[i].fin+1;
   }
    for (i=0; i<NUMTHREADS; i++) pthread_join(thread[i],NULL);

    for (i=0; i<NUMTHREADS; i++) suma = suma + suma_parcial[i];

    printf("Suma = %d\n ",suma);
    pthread_exit (0);
}
```

A partir de este programa responde a las siguientes preguntas:

a) Explica la tarea que realiza cada uno de los hilos subordinados. ¿Dónde se almacena el resultado que calcula cada hilo?

b) Cada hilo modifica un elemento del vector `suma_parcial`. ¿Ven estas modificaciones el resto de hilos y la hebra principal?

c) ¿Qué resultado muestra por pantalla el programa?

6. Any required number of parameters can be sent to a thread upon its creation using the `*arg` pointer specified in the `pthread_create()` prototype. Read carefully the code of the `max_vector.c` example. Notice what parameter is passed to each thread, its meaning and its use. Modify the `max_vector` program so that each thread returns the maximum value it computes via the `pthread_exit` function.

   Create a new version of your modified program in which each thread receives as parameters the starting and ending points of the input vector to be processed. To send many parameters to a thread, group them into a C structure and send a pointer to this structure.

   Could we use in the `max_vector.c` code a global integer variable which was updated by each thread if the maximum value it has computed is bigger than the actual value of the global variable? Justify your answer.

```
$ cat max_vector.c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

#define MAX_NUMTHREADS 20   /* We create MAX_NUMTHREADS at most */
#define VSIZE 2000000       /* Problem size */

int num_threads, subvector_dim;
int v       [VSIZE];
int vmax    [MAX_NUMTHREADS];
int index_v [MAX_NUMTHREADS];
```

```
void *f_max(void *arg)
{
  int i, begin_v, end_v, nthread, max_for_thread;

  nthread = *(int *)arg;

  begin_v = nthread * subvector_dim;
  if ( nthread  == (num_threads-1) )  end_v = VSIZE - 1;
  else                                end_v = begin_v + subvector_dim -1;

  max_for_thread=0;
  for (i=begin_v; i<=end_v; i++)
    if (v[i]>max_for_thread) max_for_thread=v[i];
  printf("Maximum computed by thread %02d (%u):  %07d\n", nthread,
          pthread_self(), max_for_thread);
  vmax[nthread] = max_for_thread;
  pthread_exit(0);
}

int main(int argc, char * argv[])
{
  int i, max;
  time_t t;

  pthread_t  thread[MAX_NUMTHREADS];

  pthread_attr_t attr;

  pthread_attr_init(&attr); /* They are to be joinable */
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

  srandom(time(&t));

  if (argc==1)
  {   printf("Error: The number of threads is required as an argument\n");
      exit(-1);
  }
  num_threads=atoi(argv[1]);

  for (i=0; i<VSIZE; i++)  /* We create the vector */
    v[i] = random() % VSIZE;
  puts("Data vector created");

  subvector_dim = VSIZE / num_threads;
  for (i = 0; i < num_threads; i++) /* We create threads, it can fail */
    { index_v[i]=i;
```

```
      if (pthread_create(&thread[i], &attr, f_max, (void *)&index_v[i]))
      {  printf("On iteration %d: ", i);
 perror("error creating thread.");
 exit(-1);
      }
    }
  printf("%d Threads successfully created.\n\n", i);

  for (i = 0; i < num_threads; i++)  /* We wait for their completion */
       if  ( pthread_join(thread[i],NULL ) )
      {  printf("On iteration %d: ", i);  /* It can fail */
 perror("error joining thread.");
 exit(-1);
      }
  printf("\n%d Threads successfully joined.\n\n", i);

  max=0;
  for (i = 0; i < num_threads; i++)
  /* We get the maximum element returned by threads */
    if ( vmax[i] > max ) max = vmax[i];
  printf("Final maximum:  %d\n", max);

  pthread_attr_destroy(&attr);    /* To release memory */
  exit(0);
}
```