Actividad 7 (v. 240409) Pablo González Nalda
Depto. de *Lenguajes y Sistemas Informáticos* lsi.vc.ehu.eus/pablogn
Universidad del País Vasco Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

Hilos (hebras)

En esta actividad vamos a ver las características principales de los hilos. Los programas de ejemplo de este documento se han tomado de los siguientes tutoriales.

- IBM: POSIX threads explained https://www.ibm.com/developerworks/library/l-posix1/index.html
- Hilos Posix: pthreads https://computing.llnl.gov/tutorials/pthreads/
- Pthread Creation and Termination https://computing.llnl.gov/tutorials/pthreads/#CreatingThreads
- Mutexes https://computing.llnl.gov/tutorials/pthreads/#Mutexes

La forma de listar los hilos de un proceso es la siguiente:

```
ps aux| grep firefox | grep -v parent # -v elimina líneas con un patrón
ps -T -p 4509
```

Más información en http://ask.xmodulo.com/view-threads-process-linux.html

7.1. Hilos

Este programa crea 9 hilos que imprimirán un mensaje y terminarán.

Observa, analiza y prueba este programa y los siguientes. Ejecuta este programa varias veces y observa que no hay un orden determinado para ejecutarse. Compila con `gcc -o ej1 A.c -lpthread` y ejecuta con `./ej1`

Prueba a listar sus hilos con las formas anteriores. Para ello activa el ciclo infinito de las líneas 18 y 36, compila de nuevo y ejecuta con `./ej1 &` y después haz `ps -T -p <PID del Ppal.>`

```
/*
 * Example: Pthread Creation and Termination
 * gcc -o ej1 A.c -lpthread
 */
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define NUM_THREADS     9

 void *PrintHello(void *threadid) {
    long tid;
    tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
//    while(1);
    pthread_exit(NULL);
 }

 int main (int argc, char *argv[]) {
```

```
20      pthread_t threads[NUM_THREADS];
21      int rc;
22      long t;
23      printf("\nPID del Ppal %d\n",getpid());
24      for(t=0; t<NUM_THREADS; t++){
25         printf("In main: creating thread %ld\n", t);
26         rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
27         if (rc){
28            printf("ERROR; return code from pthread_create() is %d\n", rc);
29            exit(-1);
30         }
31      }
32
33  //    while(1);
34      /* Last thing that main() should do */
35      pthread_exit(NULL);
36   }
```

Fichero 7.1: Fichero A.c

7.2. Control de hilos

```
1  /*
2   * Crear un hilo y esperarlo
3   * gcc thread1.c -o thread1 -lpthread
4   */
5  #include <stdio.h>
6  #include <pthread.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9  void *thread_function(void *arg) { // función que se ejecuta en el hilo
10   int i;
11   for ( i=0; i<5; i++ ) {
12     printf("Thread says hi!\n");
13     sleep(1);
14   }
15   return NULL;
16 }
17
18 int main(void) { // hilo del programa principal
19   pthread_t mythread;
20   if ( pthread_create( &mythread, NULL, thread_function, NULL) ) {
21     printf("error creating thread.");
22     abort();
23   }
24   if ( pthread_join ( mythread, NULL ) ) {
25     printf("error joining thread.");
26     abort();
27   }
28   exit(0);
29 }
```

Fichero 7.2: Fichero thread1.c

7.3. Cálculo en paralelo con hilos

Este programa paraleliza el cálculo de un sumatorio como explica en Wikipedia: https://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80

La clave es que cada hilo hace una parte, lo almacena en una posición del vector `sump` y se suman al final de la función `calcula`.

El programa se arranca con las instrucciones de las líneas 9 y 11.

```
1  /**
2   * Calculamos pi con hilos
3   *
4   * Se usa la fórmula de Leibniz: https://en.wikipedia.org/wiki/
       Leibniz_formula_for_%CF%80
5   *
6   * Según Octave, PI=3.14159265358979311599796346854 con 30 decimales.
7   *
8   * Compilar con:
9   * g++ -o pi piconhilos.cpp -std=c++11 -lpthread
10  *
11  * ./pi 100 1000
12  *
13  * A partir de https://twitter.com/Gaspar_FM
14  * https://poesiabinaria.net/2017/10/distribuir-calculos-varios-nucleos-acelerar-
       procesos-computacion-ejemplos-c/
15  */
16
17 #define _USE_MATH_DEFINES
18 #include <cmath>
19 #include <chrono>
20 #include <iostream>
21 #include <thread>
22 #include <list>
23 #include <iomanip>
24
25 using namespace std;
26
27 // Variables globales a todo el programa y todos los hilos
28 int Nthreads;
29 double* sump; //Array donde se guardarán las sumas parciales
30
31 /*
32  * Función que se ejecuta en un hilo.
33  *
34  * Hace las iteraciones del sumatorio que le corresponden a este hilo.
35  * Se calculan a partir de los parámetros:
36  * hilo: número de hilo
37  * n: cantidad base para calcular los términos que le corresponden al hilo
38  * h: valor común a todos los hilos, calculado fuera
39  */
40 void hilo_calcula_pi (int hilo, long unsigned n, double h)
41 {
42
43     double sp=0.0;  // La suma parcial la hacemos en un
44                     // espacio de memoria cercano, que será más rápido
```

```
45
46      for (long unsigned i=(n*hilo/Nthreads); i <(n*(hilo+1))/Nthreads; i++)
47      {
48          // Calculamos Sum[i=0, i=n] 4 / (1 + (1/n * (i-0.5)^2)
49          // siendo h=1/n mediante sumas parciales en cada hilo
50
51          double x = h * ((double)i - 0.5);
52
53          sp += 4.0 / (1.0 + x*x);
54      }
55      // Por último rellenamos el espacio correspondiente del array de
56      // sumas parciales.
57      sump[hilo]=sp;
58  }
59
60  double calcula (long unsigned n)
61  {
62      double suma=0;
63
64      // Preparación del entorno: reserva de memoria.
65      // Tabla para acumular los resultados parciales de los hilos.
66      sump=new double [Nthreads];
67
68      list<thread> threads;
69
70      // Evitamos que todos los threads calculen un dato común.
71      // h = 1/n, además, lo sacamos del sumatorio.
72      double h = 1.0 / (double) n;
73
74      // Cálculo de PI con 1/n * Sum[i=0, i=n] 4 / (1 + (1/n * (i-0.5)^2)
75      // Lanzamiento de los hilos
76      for (int i=0; i<Nthreads; i++)
77      {
78          // Lanzamos threads que ejecutan la función calcula_pi, les pasamos su
                número
79          // de thread, n y h
80          threads.push_back(thread(hilo_calcula_pi,i,n,h));
81      }
82
83      // Cada thread lo unimos al proceso principal, por lo que esperamos que
            terminen
84      // todos los threads lanzados.
85      for (auto &t: threads)
86      {
87          t.join();
88      }
89
90      // Fin del procesamiento y ahora iniciamos la reorganización, en este caso
91      // sumamos las sumas parciales en una sola variable y liberamos memoria
92      for (int i=0;i<Nthreads;i++)
93      {
94          suma=suma+sump[i];
95      }
96
```

```
97      delete [] sump;
98      return h * suma;
99  }
100
101 int main(int argc, char *argv[])
102 {
103     // Parseamos argumentos. El primer argumento será n iteraciones
104     // y el segundo el número de hilos
105
106     unsigned int n;
107     if (argc > 2)
108     {
109         n = atoi(argv[2]);
110         Nthreads = atoi(argv[1]);
111     }
112     else
113     {
114         printf("\nUso: %s Hilos Millones-de-Iteraciones\n",argv[0]);
115         exit(1);
116     }
117
118     auto time_inicio=chrono::high_resolution_clock::now();
119
120     double mypi=calcula(1E6*n);
121
122     auto time_fin=chrono::high_resolution_clock::now();
123     cout
124         << "Tiempo invertido: "
125         << chrono::duration<double,milli>(time_fin-time_inicio).count() << " ms,
            con: "
126         << Nthreads << " hilos y "
127         << n << " millones de iteraciones."
128         << endl;
129     cout
130         << "Error = " << setprecision(4)<< M_PI - mypi
131         << endl;
132     cout
133         << "PI = " << setprecision(16)<< mypi
134          << endl;
135     return 0;
136 }
```

7.4. Condiciones de carrera

Este programa `thread2.c` crea un hilo que escribe puntos e incrementa la variable con cada punto pero copiando el valor a una variable local `j`. El programa principal incrementa la variable global en la línea 38.

¿No debería dar siempre `myglobal equals 40`? Ejecútalo varias veces.

Si se descomentan las líneas de los `sleep(1);` el efecto es que siempre da 21. Piensa por qué.

Éste es un ejemplo de una condición de carrera o *race condition* (https://en.wikipedia.org/wiki/Race_condition#Example). El código que incrementa la variable es la Sección Crítica.

```c
/*
 * Crear un hilo que escribe puntos e incrementa myglobal
 *  El otro hilo escribe oes y también incrementa myglobal.
 * gcc thread2.c -o thread2 -lpthread
 */
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
int myglobal=0;

 void *thread_function(void *arg) {
  int i,j;
  for ( i=0; i<20; i++ ) {
    j=myglobal;
    j=j+1;
    printf(".");
    fflush(stdout);
//   sleep(1);
    myglobal=j;
  }
  return NULL;
}

int main(void) {
  pthread_t mythread;
  int i;

  if ( pthread_create( &mythread, NULL, thread_function, NULL) ) {
    printf("error creating thread.");
    abort();
  }

  for ( i=0; i<20; i++) {
    myglobal=myglobal+1;
    printf("o");
    fflush(stdout);
//    sleep(1);
  }

  if ( pthread_join ( mythread, NULL ) ) {
    printf("error joining thread.");
    abort();
  }

  printf("\nmyglobal equals %d\n",myglobal);
  exit(0);

}
```

Fichero 7.3: Fichero thread2.c

7.5. Semáforos de exclusión mutua, o *Mútex*

Los *mútex* o *semáforos de exclusión mutua* son semáforos binarios o de dos estados.

Para mostrar la necesidad de los semáforos, se ha modificado el programa del cálculo de π anterior, de forma que añadimos las partes calculadas a una única variable `suma`, operación que es una *Sección Crítica*, en la línea 64, protegida por un mútex.

Comprueba que el programa da mucho error a veces si comentas las líneas que manejan el *mútex* y pones muchos hilos, por ejemplo 1000.

```
1  /**
2   * Calculamos pi con hilos y UN MÚTEX para proteger la variable global
3   *
4   * Se usa la fórmula de Leibniz: https://en.wikipedia.org/wiki/
       Leibniz_formula_for_%CF%80
5   *
6   * Según Octave, PI=3.14159265358979311599796346854 4 con 30 decimales.
7   *
8   * Compilar con:
9   * g++ -o pim piconhilosymutex.cpp -std=c++11 -lpthread
10  *
11  * ./pim 100 1000
12  *
13  * A partir de https://twitter.com/Gaspar_FM
14  * https://poesiabinaria.net/2017/10/distribuir-calculos-varios-nucleos-acelerar-
       procesos-computacion-ejemplos-c/
15  */
16
17 #define _USE_MATH_DEFINES
18 #include <cmath>
19 #include <chrono>
20 #include <iostream>
21 #include <thread>
22 #include <list>
23 #include <iomanip>
24
25 using namespace std;
26
27 // Variables globales a todo el programa y todos los hilos
28 int Nthreads;
29 double suma=0; //Variable donde se acumularán las sumas parciales
30 pthread_mutex_t mutexsum;
31
32 /*
33  * Función que se ejecuta en un hilo.
34  *
35  * Hace las iteraciones del sumatorio que le corresponden a este hilo.
36  * Se calculan a partir de los parámetros:
37  * hilo: número de hilo
38  * n: cantidad base para calcular los términos que le corresponden al hilo
39  * h: valor común a todos los hilos, calculado fuera
40  */
41 void hilo_calcula_pi (int hilo, long unsigned n, double h)
42 {
```

```
43
44      double sp=0.0;  // La suma parcial la hacemos en un
45                      // espacio de memoria cercano, que será más rápido
46
47      for (long unsigned i=(n*hilo/Nthreads); i <(n*(hilo+1))/Nthreads; i++)
48      {
49          // Calculamos Sum[i=0, i=n] 4 / (1 + (1/n * (i-0.5)^2)
50          // siendo h=1/n mediante sumas parciales en cada hilo
51
52          double x = h * ((double)i - 0.5);
53
54          sp += 4.0 / (1.0 + x*x);
55      }
56      // Por último rellenamos el espacio correspondiente del array de
57      // sumas parciales.
58      pthread_mutex_lock (&mutexsum);
59          double t = suma+sp; // sección crítica
60          cout
61              << "Hilo "<< hilo
62          << endl;
63          cout
64              << " \t sp="<< setprecision(16) << sp*h
65              << " \t suma="<< setprecision(16) << suma*h
66              << " \t t="<< setprecision(16) << t*h
67           << endl;
68          suma=t;
69      pthread_mutex_unlock (&mutexsum);
70  }
71
72  double calcula (long unsigned n)
73  {
74      list<thread> threads;
75
76      // Evitamos que todos los threads calculen un dato común.
77      // h = 1/n, además, lo sacamos del sumatorio.
78      double h = 1.0 / (double) n;
79
80      // Cálculo de PI con 1/n * Sum[i=0, i=n] 4 / (1 + (1/n * (i-0.5)^2)
81      // Lanzamiento de los hilos
82      for (int i=0; i<Nthreads; i++)
83      {
84          // Lanzamos threads que ejecutan la función calcula_pi, les pasamos su
                número
85          // de thread, n y h
86          threads.push_back(thread(hilo_calcula_pi,i,n,h));
87      }
88
89      // Cada thread lo unimos al proceso principal, por lo que esperamos que
            terminen
90      // todos los threads lanzados.
91      for (auto &t: threads)
92      {
93          t.join();
94      }
```

```
95
96      return h * suma;
97  }
98
99  int main(int argc, char *argv[])
100 {
101     // Parseamos argumentos. El primer argumento será n iteraciones
102     // y el segundo el número de hilos
103
104     unsigned int n;
105     if (argc > 2)
106     {
107         n = atoi(argv[2]);
108         Nthreads = atoi(argv[1]);
109     }
110     else
111     {
112         printf("\nUso: %s Hilos Millones-de-Iteraciones\n",argv[0]);
113         exit(1);
114     }
115     pthread_mutex_init(&mutexsum, NULL);
116     auto time_inicio=chrono::high_resolution_clock::now();
117
118     double mypi=calcula(1E6*n);
119
120     auto time_fin=chrono::high_resolution_clock::now();
121     cout
122         << "Tiempo invertido: "
123         << chrono::duration<double,milli>(time_fin-time_inicio).count() << " ms,
               con: "
124         << Nthreads << " hilos y "
125         << n << " millones de iteraciones."
126         << endl;
127     cout
128         << "Error = " << setprecision(4)<< M_PI - mypi
129         << endl;
130     cout
131         << "PI = " << setprecision(16)<< mypi
132          << endl;
133     pthread_mutex_destroy(&mutexsum);
134     return 0;
135 }
```

Fichero 7.4: Fichero piconhilosymutex.cpp

Las diferencias entre las dos versiones se ven en la salida de este comando:

```
1 diff piconhilos.cpp piconhilosymutex.cpp > salida.diff
```

El resultado de la instrucción `diff` es el siguiente. `2c2` indica que cambia la línea 2 de ambos ficheros, y con los < y > muestra las filas del fichero de la izquierda y el de la derecha.

```
1 2c2
2 <   * Calculamos pi con hilos
3 ---
4 >   * Calculamos pi con hilos y UN MÚTEX para proteger la variable global
```

```
9c9
<  * g++ -o pi piconhilos.cpp -std=c++11 -lpthread
---
>  * g++ -o pim piconhilosymutex.cpp -std=c++11 -lpthread
11c11
<  * ./pi 100 1000
---
>  * ./pim 100 1000
29c29,30
< double* sump; //Array donde se guardarán las sumas parciales
---
> double suma=0; //Variable donde se acumularán las sumas parciales
> pthread_mutex_t mutexsum;
57c58,60
<     sump[hilo]=sp;
---
>     pthread_mutex_lock (&mutexsum);
>     suma+=sp; // sección crítica
>     pthread_mutex_unlock (&mutexsum);
62,67d64
<     double suma=0;
<
<     // Preparación del entorno: reserva de memoria.
<     // Tabla para acumular los resultados parciales de los hilos.
<     sump=new double [Nthreads];
<
90,97d86
<     // Fin del procesamiento y ahora iniciamos la reorganización, en este caso
<     // sumamos las sumas parciales en una sola variable y liberamos memoria
<     for (int i=0;i<Nthreads;i++)
<     {
<         suma=suma+sump[i];
<     }
<
<     delete [] sump;
117c106
<
---
>     pthread_mutex_init(&mutexsum, NULL);
134a124
>     pthread_mutex_destroy(&mutexsum);
```

Fichero 7.5: Salida del comando `diff`