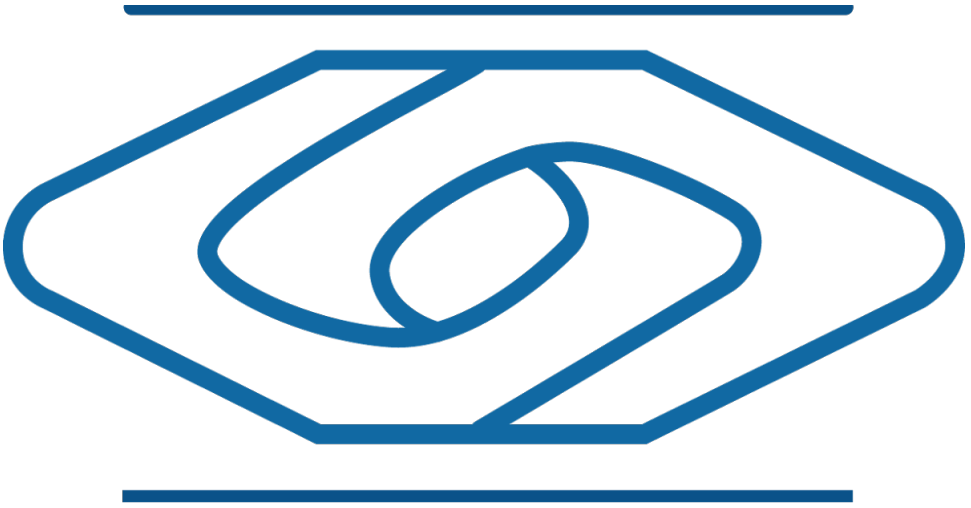

Instituto Politécnico Nacional
Escuela Superior de Computo

Práctica 6: Threads

Reporte
Profesor: Ulises Velez Saldaña
Alumno: Meza Madrid Raúl Damián
Clase: Sistemas operativos
Grupo: 2CM7

Contents

Introducción

1.1 Threads

En los sistemas operativos tradicionales, cada proceso tiene un espacio de direcciones y un solo hilo de control. De hecho, ésa es casi la definición de un proceso. Sin embargo con frecuencia hay situaciones en las que es conveniente tener varios hilos de control en el mismo espacio de direcciones que se ejecuta en cuasi-paralelo, como si fueran procesos (casi) separados (excepto por el espacio de direcciones compartido). LA principal razón para tener hilos es que en muchas aplicaciones se desarrollan varias actividades a la vez. Algunas de esas se pueden bloquear de vez en cuando. Al descomponer una aplicación en varios hilos secuenciales se ejecutan casi en paralelo, el modelo de programación se simplifica. A comparación de los procesos, con hilos se tiene la capacidad de compartir un espacio de direcciones y todos sus datos entre ellas. Esta habilidad es esencial para ciertas aplicaciones, razón por la cual no funcionara el tener diferentes procesos.
Otra razón para tener hilos es que son mas ligeros que los procesos; más fáciles de crear y de destruir. En muchos sistemas, la creación de un hilo es de 10 a 100 veces mas rápida.

1.2 Programas y herramientas utilizados

Esta práctica fue desarrollada en el sistema operativo Ubuntu 18.04.1 LTS. Estos son los programas y herramientas utilizados, junto con el comando de instalación, en caso de que no estuvieran instalados ya.

- Doxygen

```
git clone https://github.com/doxygen/doxygen.git
cd doxygen
mkdir build
cd build
cmake -G "Unix Makefiles" ..
make
make install
```

- make
- cmake
- C

Objetivo

Que el alumno aplique la teoría vista en clase implementando la creación de varios hilos cada uno encargado de sumar un determinado renglón dentro de una suma de matrices.

Desarrollo

El primer paso es leer los datos desde la entrada estándar; STDIN. El programa puede correr de esta manera, pero el archivo make utiliza el archivo ./src/input.txt como entrada para las matrices donde las primer linea denota las dimensiones n m de la matriz, las siguientes m lineas denotan los renglones con n números, separados por espacios, de la matriz A; los m siguientes renglones corresponden de la misma manera a la matriz B.

3.1 Codigo

El codigo fuente contiene comentarios que describen el programa. Son utilizados tambien para documentar con doxygen.

3.1.1 Código fuente

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int n=0,m=0;
int *matrixA,*matrixB, *indexes;
pthread_t *threads;

/// Funcion para no tener que hacer un arreglo bidimensional
/// \code
int myIndex(int i,int j){
  return i + j*m;
}
/// \endcode


/// Funcion para agregar todos los elementos dentro de la misma
 ↪  fila del arrelgo  dado el indice de la misma
```

```
18 /// \code
19 void *addRow(void *arg){
20   int j = *(int*)arg;
21   for (size_t i = 0; i < n; i++) {
22     matrixA[myIndex(i,j)]+=matrixB[myIndex(i,j)];
23   }
24   pthread_exit(NULL);
25 }
26 /// \endcode
27
28 int main(int argc, char const *argv[]) {
29   system("clear");
30   /// Se leen los tamaños de las matrices, y se le asigna la
     ↪  memoria correspondiente junto al arreglo de threads
31   /// \code
32   scanf("%d %d",&n,&m);
33   matrixA = (int*)malloc(sizeof(int)*(n+1)*(m+1));
34   matrixB = (int*)malloc(sizeof(int)*(n+1)*(m+1));
35   threads = (pthread_t*)malloc(sizeof(pthread_t)*(m+1));
36   indexes = (int*)malloc(sizeof(int)*(m+1));
37   /// \endcode
38
39   /// Se lee la matriz A
40   /// \code
41   for (size_t j = 0; j < m; j++) {
42     for (size_t i = 0; i < n; i++) {
43       scanf("%d", &matrixA[myIndex(i, j)]);
44     }
45   }
46   printf("\nMatriz A: \n");
47   for (size_t j = 0; j < m; j++) {
48     for (size_t i = 0; i < n; i++) {
49       printf("%d ", matrixA[myIndex(i, j)]);
50     }
51     printf("\n");
52   }
53
54   /// \code
55
56   /// Se lee la matriz A
```

```
57    /// \code
58    printf("\nMatriz B: \n");
59    for (size_t j = 0; j < m; j++) {
60      for (size_t i = 0; i < n; i++) {
61        scanf("%d", &matrixB[myIndex(i, j)]);
62      }
63    }
64    for (size_t j = 0; j < m; j++) {
65      for (size_t i = 0; i < n; i++) {
66        printf("%d ", matrixB[myIndex(i, j)]);
67      }
68      printf("\n");
69    }
70
71    /// \code
72
73    /// Se crean los hilos y los datos correspondientes para cada
      ↪  uno de ellos
74    /// \code
75    for (int j = 0; j < m; j++) {
76      indexes[j] = j;
77      pthread_create(&threads[j], NULL, addRow, (void *)&indexes[j]);
78    }
79    /// \endcode
80
81    /// Se imprime la matriz
82    /// \code
83    for (int j = 0; j < m; j++) {
84      pthread_join(threads[j], NULL);
85    }
86    printf("\nMatriz resultante: \n");
87    for (size_t j = 0; j < m; j++) {
88      for (size_t i = 0; i < n; i++) {
89        printf("%d ", matrixA[myIndex(i, j)]);
90      }
91      printf("\n");
92    }
93    /// \code
94  }
```

Resultados

El programa funciona de manera adecuada. A continuacion se muestran dos test case para ilustrar la salida del programa.

4.1 Caso de prueba: A

4.1.1 Entrada (input.txt)

```
3 4
3 4 5
2 3 5
10 2 11
10 2 11
7 6 5
8 7 5
0 8 -1
10 2 11
```

4.1.2 Salida (terminal)

```
Matriz A:
3 4 5
2 3 5
10 2 11
10 2 11

Matriz B:
7 6 5
8 7 5
0 8 -1
10 2 11

Matriz resultante:
10 10 10
10 10 10
10 10 10
20 4 22
```

4.2 Caso de prueba: B

4.2.1 Entrada (input.txt)

```
3 5
0 1 2
1 2 3
2 3 4
3 4 5
4 5 6
0 -1 -2
-1 -2 -3
-2 -3 -4
-3 -4 -5
-4 -5 -6
```

4.2.2 Salida (terminal)

```
Matriz A:
0 1 2
1 2 3
2 3 4
3 4 5
4 5 6

Matriz B:
0 -1 -2
-1 -2 -3
-2 -3 -4
-3 -4 -5
-4 -5 -6

Matriz resultante:
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
```

Errores y problemas

Fue necesario crear un arreglo para que los hilos pudieran acceder a sus argumentos sin que estos cambien durante la ejecución.

Codigo (Github)

Todo el codigo de esta practica se puede encontrar en :`https://github.com/asdf1234Damian/Operating-Systems/tree/master/Practica06`

References

[1] GeeksforGeeks. MultiThreading in C - GeeksforGeeks. `https://www.geeksforgeeks.org/multithreading-c-2/`. [Online; consultado en 5 de mayo 2019].

[2] GeeksforGeeks. void pointer in C/C++ - Linux manual page. `https://www.geeksforgeeks.org/void-pointer-c-cpp/`. [Online; consultado en 5 de mayo 2019].

[3] Michael Kerrisk. pthread_create(3) - Linux manual page. `http://man7.org/linux/man-pages/man3/pthread_create.3.html`. [Online; consultado en 5 de mayo 2019].

[4] perreal. re: How to make main thread wait for all child threads finish? `https://stackoverflow.com/questions/11624545/how-to-make-main-thread-wait-for-all-child-threads-finish`. [Online; consultado en 5 de mayo 2019].

[5] Andrew S. Tanenbaum and Garcîa Roberto Escalona. *Sistemas operativos modernos*. Pearson Educación, 2 edition, 2003.