

PRÁCTICA Nº 1: CONDICIONES DE COMPETENCIA

Jose Masri Salame, UAM-C

15/05/2020

Objetivo

El alumno estará consciente de los efectos que ocasionan las condiciones de competencia en programas concurrentes.

Actividades

Para experimentar con las condiciones de competencia deberán implementar dos programas concurrentes creando hilos.

1. Suma de Gauss

1.1. Introducción

1.1.1. Condiciones de competencia

Decimos que ocurren **condiciones de carrera** o de competencia cuando 2 o más procesos tratan de manipular una misma región o espacio de memoria. Cuando ocurren condiciones de carrera la salida del programa tiene una alta posibilidad de verse afectada. Imaginemos este ejemplo, tenemos una variable x en memoria y tenemos 2 tareas ejecutándose de manera concurrente. En la tarea 1 se le asigna 5 a la variable y después se le suma 5, mientras que en la tarea 2 se le asigna 1 y después se le suma 1. El resultado esperado es que en la tarea 1 x valga 10 y en la tarea 2 x tenga un valor de 2. Sin embargo esto solo sucede si la primera tarea se ejecuta antes que la segunda, o viceversa, pero si se ejecutan de manera simultánea el resultado puede variar con cada ejecución debido al orden en el que el procesador ejecute las instrucciones.

1.1.2. Sección crítica

Un concepto estrechamente relacionado con el anterior es el de **sección crítica**, el cual se refiere a porciones de código que provocan las condiciones de carrera. Toda manipulación a una variable compartida que se utilice de manera simultánea se puede considerar como sección crítica. A estas secciones se les debe dar un tratamiento especial y se debe tener cuidado al momento de manipularlas asegurándonos que ningún otro proceso este utilizándolas en ese momento.

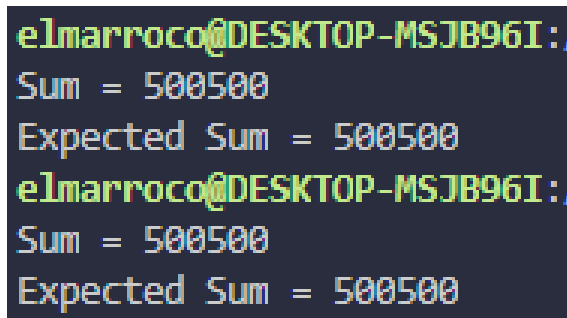
1.2. Instrucciones

Escribir un programa multihilado que obtenga la suma de los elementos consecutivos de un arreglo sin usar mecanismos de sincronización.

1.3. Lista de secciones críticas del algoritmo

En el Listing 2 podemos observar el código que resulto para resolver el problema. En la línea 10 podemos observar que tenemos una variable global de tipo entero con el nombre *totalSum* que se comparte con todos los hilos para calcular la suma. Si observamos la función *sum* que es la porción de código que ejecutara cada uno de los hilos de manera concurrente, en la línea 57 observamos que todos los hilos modifican la variable *totalSum* con lo que esta es la única sección crítica.

1.4. Resultados



```
elmarroco@DESKTOP-MSJB96I:
Sum = 500500
Expected Sum = 500500
elmarroco@DESKTOP-MSJB96I:
Sum = 500500
Expected Sum = 500500
```

Listing 1: Sección crítica.

```
1 totalSum += arr[i];
```

Listing 2: Suma de Gauss paralela.

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define NUM_THREADS 10
5 #define SIZE_ARR 1000
6
7 // Global variables declaration
8 // Accesible by anyone
9 int arr[SIZE_ARR];
10 int totalSum = 0;
11
12 // Functions declaration
13 void *sum(void *threadid);
```

```

14
15 int main(int argc, char *argv[])
16 {
17     // Creating thread array
18     pthread_t threads[NUM_THREADS];
19     // Variable for the return code
20     // of creating the thread
21     int rc;
22     // Initializing array
23     for (int i = 0; i < SIZE_ARR; i++)
24         arr[i] = i + 1;
25     // Creating threads with sum function
26     for (long t = 0; t < NUM_THREADS; t++)
27     {
28         // printf("In main: creating thread %ld\n", t);
29         rc = pthread_create(&threads[t], NULL, sum, (void *)t);
30         // Error handling
31         if (rc)
32         {
33             printf("ERROR; return code from pthread_create() is %ld\n", rc)↵
34             ;
35             exit(-1);
36         }
37     }
38     // Waiting for all threads to end
39     for (long t = 0; t < NUM_THREADS; t++)
40         pthread_join(threads[t], NULL);
41     // Outputing the result
42     printf("Sum = %ld\n", totalSum);
43     // Outputing the computed result
44     printf("Expected Sum = %ld\n", (SIZE_ARR * (SIZE_ARR + 1)) / 2);
45     // Exiting principal thread
46     pthread_exit(NULL);
47 }
48 // Sum function for threads
49 // Argument is the thread id
50 void *sum(void *threadid)
51 {
52     long tid;
53     int chunk_size = SIZE_ARR / NUM_THREADS;
54     tid = (long)threadid;
55     // printf("Hello World! It's me, thread #%ld!\n", tid);
56     for (int i = tid * chunk_size; i < (tid * chunk_size) + chunk_size; i↵
57         ++
58         totalSum += arr[i];
59     pthread_exit(NULL);

```

2. Multiplicacion de matrices

2.1. Instrucciones

Escribir un programa multihilado que realice la multiplicación de 2 matrices A y B de tamaños $n \times m$ y $m \times q$ para dar como resultado una matriz C de tamaño $n \times q$.

2.1.1. Versión secuencial

Tiempos de ejecución para N,M,Q=4000

- 10m 58.409s
- 10m 57.392s
- 10m 59.003s
- 10m 58.635s
- 10m 59.223s

2.1.2. Versión concurrente

Tiempos de ejecución para N.M,Q=4000

- 1m 55.689s
- 1m 57.446s
- 1m 56.178s
- 1m 56.958s
- 1m 59.254s
- 1m 58.325s

Listing 3: Multiplicación matrices secuencial.

```
1 #include <pthread.h>
2 #include <time.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5
6 // Constanats
```

```

7  #define MAX_INT 10
8  // Matrixes size
9  #define N 4000
10 #define M 4000
11 #define Q 4000
12
13 // Matrixes declaration
14 int A[N][M], B[M][Q], C[N][Q];
15
16 // Function declaration
17 void multiply();
18
19 int main(int argc, char const *argv[])
20 {
21     // Initilizing random seed
22     srand(time(NULL));
23     // Initialize matrixes randomly
24     for (int i = 0; i < N; i++)
25         for (int j = 0; j < M; j++)
26             A[i][j] = rand() % MAX_INT;
27
28     for (int i = 0; i < M; i++)
29         for (int j = 0; j < Q; j++)
30             B[i][j] = rand() % MAX_INT;
31
32     // Printing matrixes
33     // for (int i = 0; i < N; i++)
34     //     for (int j = 0; j < M; j++)
35     //         printf("A[%d][%d] = %d\n", i, j, A[i][j]);
36
37     // for (int i = 0; i < M; i++)
38     //     for (int j = 0; j < Q; j++)
39     //         printf("B[%d][%d] = %d\n", i, j, B[i][j]);
40
41     // Multiplying matrixes
42     multiply();
43
44     // Displaying result
45     // for (int i = 0; i < N; i++)
46     //     for (int j = 0; j < Q; j++)
47     //         printf("C[%d][%d] = %d\n", i, j, C[i][j]);
48     return 0;
49 }
50
51 void multiply()
52 {
53     for (int i = 0; i < N; ++i)

```

```

54         for (int j = 0; j < Q; ++j)
55             for (int k = 0; k < M; ++k)
56                 C[i][j] += A[i][k] * B[k][j];
57     }

```

2.1.3. Versión concurrente

Listing 4: Multiplicación matrices secuencial.

```

1  #include <pthread.h>
2  #include <time.h>
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  // Constants
7  #define MAX_INT 10
8  // Matrixes size
9  #define N 4000
10 #define M 4000
11 #define Q 4000
12 #define NUM_THREADS 8
13
14 // Matrixes declaration
15 int A[N][M], B[M][Q], C[N][Q];
16
17 // Function declaration
18 void *multiply();
19
20 int main(int argc, char const *argv[])
21 {
22     // Initializing random seed
23     srand(time(NULL));
24     // Initialize matrixes randomly
25     for (int i = 0; i < N; i++)
26         for (int j = 0; j < M; j++)
27             A[i][j] = rand() % MAX_INT;
28
29     for (int i = 0; i < M; i++)
30         for (int j = 0; j < Q; j++)
31             B[i][j] = rand() % MAX_INT;
32
33     // Printing matrixes
34     // for (int i = 0; i < N; i++)
35     //     for (int j = 0; j < M; j++)

```

```

36     //          printf("A[%d][%d] = %d\n", i, j, A[i][j]);
37
38     // for (int i = 0; i < M; i++)
39     //     for (int j = 0; j < Q; j++)
40     //         printf("B[%d][%d] = %d\n", i, j, B[i][j]);
41
42     // Creating thread array
43     pthread_t threads[NUM_THREADS];
44     // Variable for the return code
45     // of creating the thread
46     int rc;
47     // Launching threads
48     // Creating threads with sum function
49     for (long t = 0; t < NUM_THREADS; t++)
50     {
51         // printf("In main: creating thread %ld\n", t);
52         rc = pthread_create(&threads[t], NULL, multiply, (void *)t);
53         // Error handling
54         if (rc)
55         {
56             printf("ERROR; return code from pthread_create() is %d\n", rc)↵
57             ;
58             exit(-1);
59         }
60     }
61     // Waiting for all threads to end
62     for (long t = 0; t < NUM_THREADS; t++)
63         pthread_join(threads[t], NULL);
64
65     // Displaying result
66     // for (int i = 0; i < N; i++)
67     //     for (int j = 0; j < Q; j++)
68     //         printf("C[%d][%d] = %d\n", i, j, C[i][j]);
69     return 0;
70 }
71 void *multiply(void *threadid)
72 {
73     long tid = (long)threadid;
74     // Cells to compute
75     long cellsToCompute = N * Q / NUM_THREADS;
76     long start = cellsToCompute * tid;
77     long end = start + cellsToCompute;
78
79     long row;
80     long col;
81

```

```
82     for (int k = start; k < end; k++)
83     {
84         row = k / cellsToCompute;
85         col = k % cellsToCompute;
86         // One cell calculation
87         for (int i = 0; i < M; ++i)
88             C[row][col] += A[row][i] * B[i][col];
89     }
90 }
```
