**Instituto Tecnológico y de Estudios Superiores de Occidente**

Departamento de Electrónica, Sistemas e Informática

Ingeniería en Sistemas Computacionales



PROGRAMACIÓN CON MEMORIA DINÁMICA

Tarea 2. Apuntadores a Funciones

Autor: Flores Ruiz David

10 de junio de 2018. Tlaquepaque, Jalisco,

Presentación: 5 pts.

Funcionalidad: 60 pts. (Necesito verificar tu código en el repositorio)

Pruebas: 10 pts.

- Tienes problemas de redacción y no tienes el texto alineado.

- Las figuras deben tener un número y descripción.

- Las figuras, tablas, diagramas y algoritmos en un documento, son material de apoyo para transmitir ideas.

- Sin embargo deben estar descritas en el texto y hacer referencia a ellas. Por ejemplo: En la Figura 1….

- Falta describir las pruebas (escenario, y resultados de la experimentación).

- Cuando se tienen resultados que se pueden comparar, se recomienda hacer uso de diagramas o tablas que permitan observar el resultado de los diversos casos y contrastas los resultados (en el tiempo por ejemplo).

**Instrucciónes para entrega de tarea**

Esta tarea, como el resto, es ***IMPRESCINDIBLE*** entregar los entregables de esta actividad de la siguiente manera:

* **Reporte:** vía *moodle* en **un archivo PDF**.
* **Código:** vía su repositorio **Github**.

La evaluación de la tarea comprende:

* 10% para la presentación
* 60% para la funcionalidad
* 30% para las pruebas

Es necesario responder el apartado de conclusiones, pero no se trata de llenarlo con paja. Si no se aprendió nada al hacer la práctica, es preferible escribir eso. Si el apartado queda vacío, se restarán puntos al porcentaje de presentación.

**Objetivo de la actividad**

El objetivo de la tarea es que el alumno aplique los conocimientos y habilidades adquiridos en el tema de apuntadores a funciones y la distribución de tareas mediante el uso de hilos para la resolución de problemas utilizando el lenguaje ANSI C.

**Descripción del problema**

Existen diversas técnicas para generar una aproximación del valor del número irracional **Pi**. En este caso utilizaremos la serie de Gregory y Leibniz.

**Procedimiento**

1. Codificar una solución secuencial (sin el uso de hilos) que calcule el valor de Pi, su solución debe basarse en la serie de Gregory y Leibniz para calcular los primeros diez dígitos decimales de Pi. Para esto, utilice los primeros tomando los primeros 50,000’000,000 términos de la seria.
2. Utilice las funciones definidas en la librería **time.h** (consulte diapositivas del curso) para medir el tiempo (en milisegundos) que requiere el cálculo del valor de **Pi**. Registre el tiempo.
3. Parametrice la solución que se implemento en el paso 1.
4. Utilice hilos para repartir el trabajo de calcular el valor de **Pi**. Pruebe su solución con los siguientes casos: 2 hilos, 4 hilos, 8 hilos y 16 hilos.
5. Tomar el tiempo en milisegundos que toma el programa para calcular el valor de **Pi** en cada uno de los casos mencionados en el paso 4.
6. Registre los tiempos registrados para cada caso en la siguiente tabla:

|  |  |
| --- | --- |
| No. de Hilos | Tiempo (milisegundos) |
| Sin hilos (secuencial) | 917,412 ms |
| 1 | 931,076 ms |
| 2 | 810,953 ms |
| 4 | 775,912 ms |
| 8 | 683,262 ms |
| 16 | 484,795 ms |

**Descripción de la entrada**

El usuario deberá indicar al programa cuantos hilos quiere utilizar para el calcular el valor de **Pi**.

**Descripción de la salida**

En un renglón imprimirá el valor calculado de **Pi**, con exactamente 10 dígitos decimales. En el siguiente renglón mostrará el número de milisegundos que se requirió para realizar el cálculo.

Ejemplo de ejecución:

Hilos? 4

Pi: 3.1415926535

Tiempo: 24487 ms

Solución del alumno, pruebas y Conclusiones

Código fuente de la versión secuencial (sin el uso de hilos)

/\*

============================================================================

Name : -.c

Author :

Version :

Copyright : Your copyright notice

Description : Hello World in C, Ansi-style

============================================================================

\*/

**#include** <stdio.h>

**#include** <stdlib.h>

//#include <math.h>

**#include** <time.h> //Para funcion clock() que regresa el tiempo en MILISEGUNDOS en que se ejecuto el programa

**int** **main**(**void**) {

**setvbuf**(stdout, NULL, \_IONBF, 0);

clock\_t inicial, final, total;

**double** numerador, denominador;

**double** sumatoria = 0;

**double** PI;

**unsigned** **long** **long** n;

//printf("Los numeros de la serie son:");

inicial = **clock**();

**printf**("Going to scan a big loop, inicial = %ld\n", inicial);

**for**(n=1; n<=500000000; n++){ //Debe ser la prueba final con: 50\_000\_000\_000

numerador =( (n+1) & 1 )? -1 : 1; // if ( (n+1) %2 == 0 )

// numerador = 1;

// else

// numerador = -1;

//numerador = pow(-1,n+1);

denominador = (2\*n) - 1;

//printf("Going to scan a big loop, cambiando = %ld\n", clock());

sumatoria = sumatoria + (numerador/denominador);

//printf("\n %.15lf", sumatoria);

}

PI = 4.0\*sumatoria; // Multiplicar 4 \* sumatoriaFinal

final = **clock**();

**printf**("End of the big loop, final = %ld\n", final);

total = (1000 \*(final - inicial )) /CLOCKS\_PER\_SEC;

**printf**("\nTiempo de ejecución: %ld ms", total);

**printf**("\nPI:\t %.10lf", PI);

**return** EXIT\_SUCCESS;

}

Código fuente de la versión paralelizada

/\*

============================================================================

Name : -.c

Author :

Version :

Copyright : Your copyright notice

Description : Hello World in C, Ansi-style

============================================================================

\*/

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h> //Para funcion clock() que regresa el tiempo en MILISEGUNDOS en que se ejecutó el programa

**#include** <windows.h> // Para el uso y creación de hilos (en windows)

**typedef** **struct** {

**unsigned** **long** **long** ini;

**unsigned** **long** **long** fin;

**double** sumatoriaParcial; //Genera mis 4 "n" acumuladores, para sumarlos al final

} RANGO;

DWORD WINAPI **aproxPI**(**void** \*);

**int** **main**(**void**) {

**setvbuf**(stdout, NULL, \_IONBF, 0);

clock\_t inicial, final, total;

**int** numHilos;

**printf**("Hilos? \t");

**scanf**("%d", &numHilos);

// printf("%d \n", numHilos); //Ya lee la cantidad de hilos que usará!!!

HANDLE hilo[16];

**int** i;

**double** SumatoriaFinal = 0, valorPI;

RANGO arrRangos[16]; // EL nombre tiene la direccion inicial

**unsigned** **long** **long** Razon;

Razon = 50000000000 / numHilos;

// printf("El valor de razon es: %I64u\n", Razon);

**for** (i = 0; i < numHilos; i++) { //Repartir los Rangos segun el numHilos

arrRangos[i].ini = (Razon \* (i + 1)) - Razon + 1; //Inicios del for

arrRangos[i].fin = (Razon \* (i + 1));//fin de los for

arrRangos[i].sumatoriaParcial = 0;//Inicializar los acumuladores

}

/\*

for (i = 0; i < numHilos; i++) { //Comprobar en pantalla el valor de los rangos

printf("Ini %I64u\n", arrRangos[i].ini);

printf("Fin %I64u\n", arrRangos[i].fin);

printf("Suma Parcial en0: %lf\n", arrRangos[i].sumatoriaParcial);

}

\*/

inicial = **clock**();

**printf**("Going to scan a big loop, inicial = %ld\n", inicial);

**for** (i = 0; i < numHilos; i++) { //Crear los hilos y ejecutar sus funciones

hilo[i] = **CreateThread**(

NULL, //Seguridad por default

0, //tamaño de stack por default (memoria estática)

aproxPI, //dirección de la funcion que ejecuta el hilo

(**void** \*) &arrRangos[i], //(direción) del parametro de la funcion tipo void

0, //ejecutar inmediatamente la función

NULL //sin ID para el Hilo

);

}

**WaitForMultipleObjects**(numHilos, hilo, TRUE, INFINITE);

RANGO \*p = arrRangos;

**for** (i = 0; i < numHilos; i++) {

SumatoriaFinal = SumatoriaFinal + (p + i)->sumatoriaParcial;

}

valorPI = 4.0 \* SumatoriaFinal; // Multiplicar 4 \* sumatoriaFinal

final = **clock**();

**printf**("End of the big loop, final = %ld\n", final);

total = (1000 \* (final - inicial)) / CLOCKS\_PER\_SEC;

**printf**("\nTiempo de ejecución: %ld ms", total);

**printf**("\nPI:\t %.10lf", valorPI);

**return** EXIT\_SUCCESS;

}

DWORD WINAPI **aproxPI**(**void** \*losRangos) {

RANGO \*p = (RANGO \*) losRangos;

**double** numerador, denominador; //PI

**unsigned** **long** **long** n;

**for** (n = p->ini; n <= p->fin; n++) { //Debe ser la prueba final con: 50\_000\_000\_000

numerador = ((n + 1) & 1) ? -1 : 1;

denominador = (2 \* n) - 1;

p->sumatoriaParcial = p->sumatoriaParcial + (numerador / denominador);

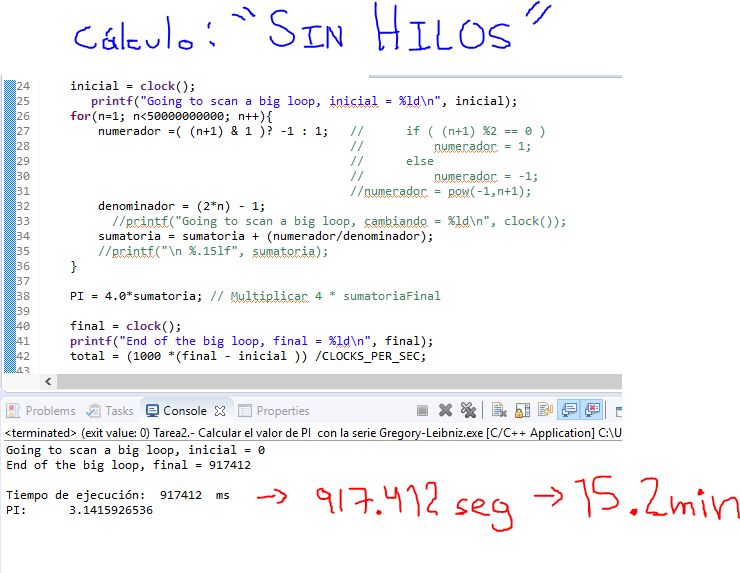
}

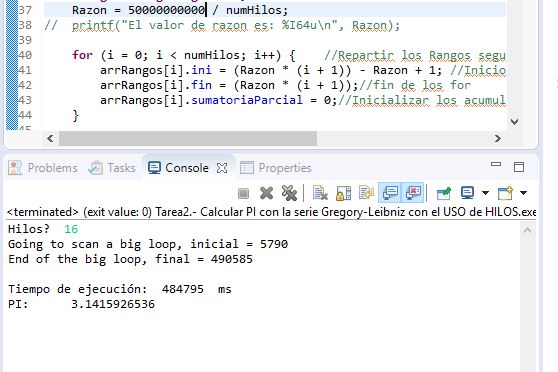
**return** 0;

}

Ejecución

<<Inserte capturas de pantalla de una ejecución secuencial y de la ejecución de su solución paralelizada para todos los casos>>





Conclusiones (obligatorio):

* Lo que aprendí con esta práctica. Lo que ya sabía.

Aprendí a como se usaban los hilos en un programa y repartir los límites de las responsabilidades, me di cuenta que al momento de crear los hilos se ejecutaba su funcionalidad, y que debo ser muy cuidadoso con los tipos de datos que uso porque puede que algún número exceda en su representación o rango.

Fue muy difícil encontrar la forma de repartir las responsabilidades pero con unas pocas operaciones funcionó todo bien.

* Lo que me costó trabajo y cómo lo solucioné.

Saber que mi programa iba bien hasta cierto punto, usando cantidades pequeñas e ir imprimiendo los números, me costó encontrar el modificador para long long unsigned, y tener paciencia para ejecutarlo con el número de la tarea porque como tardaba mucho yo no sabía si se estaba ejecutando.

* Lo que no pude solucionar.

Si se pudieron solucionar las dificultades, hubo una que no me reconocía la función del hilo y era porque faltaba una llave para cerrar el main, otra fue que en números chicos si funcionaba pero con grandes no y era por el tipo de dato de un int dentro del for de la función del hilo.

Entendí mucho más el trabajo de los hilos con esta tarea y creo que lo más difícil es repartir las responsabilidades según el número de hilos.