**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: Durán Padilla, Mauricio

4 de junio de 2018. Tlaquepaque, Jalisco,

Presentación: 5 pts.

Funcionalidad: 60 pts. (Buena observación sobre el tiempo de ejecución al usar variables globales vs paso de parámetros)

Pruebas: 20 pts.

- 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 suo 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) |
| 1 |  |
| 2 | 72254 |
| 4 | 38996 |
| 8 | 38198 |
| 16 | 37739 |

**Descripción de la entrada**

El usuario eberá 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

Solución del alumno, pruebas y Conclusiones

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

**#include** <stdio.h>

**#include** <stdlib.h> // esta permite trabajar con memoria dinámica

**#include** <math.h>

**#include** <string.h>

**#include** <time.h>

**double** **pi**();

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

clock\_t start = clock();

**printf**("Pi:\t%.10lf\n",pi() );

clock\_t stop = clock();

**long** ms = 1000 \* (stop - start)/CLOCKS\_PER\_SEC;

**printf**("Tiempo:\t%ldms\n", ms);

**return** EXIT\_SUCCESS;

}

**double** **pi**(){

**double** suma = 0;

**int** w = 1;

**for** (**long** **long** i = 1; i <= 50000000000; i++) {

suma += (( i + 1 ) & 1 ? -1.0 :1.0)/(2\*i-1);

}

suma \*= 4;

**return** suma;

}

**Código fuente de la versión paralelizada**

**#include** <stdio.h>

**#include** <stdlib.h> // esta permite trabajar con memoria dinámica

**#include** <math.h>

**#include** <string.h>

**#include** <time.h>

**#include** <pthread.h>

**typedef** **struct**{

**long** low;

**long** high;

}limit;

**double** valorPi = 0;

**void** \* **pi**(**void** \* arg);

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

**struct** timespec start, finish; //

**int** elapsed;//

limit lim[16];

pthread\_t h[16];

**int** n;

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

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

clock\_gettime(CLOCK\_MONOTONIC, &start);

**long** r = 50000000000/n;

**for** (**int** i = 0; i < n; i++) {

lim[i].low= r\*i+1;

lim[i].high =r\*(i+1);

pthread\_create(&h[i], NULL, pi, (**void** \*) &lim[i]);

}

**double** resp;

**for** (**int** k = 0; k < n; k++) {

pthread\_join(h[k] , NULL);

}

**printf**("Pi:\t%.10lf\n", valorPi);

//clock\_t stop = clock();

clock\_gettime(CLOCK\_MONOTONIC, &finish);

//long ms = (stop - start)/CLOCKS\_PER\_SEC;

elapsed = (finish.tv\_sec - start.tv\_sec)\*1000;

elapsed += (finish.tv\_nsec - start.tv\_nsec)\*1000 / 1000000000.0;

**printf**("Tiempo:\t%d ms\n", elapsed);

**return** EXIT\_SUCCESS;

}

**void** \* **pi**(**void** \* arg){

limit \*lim = (limit\*) arg;

**double** suma = 0;

**for** (**long** **long** i = lim->low; i <= lim->high; i++) {

//suma += (pow(-1,i+1)/(2\*i-1));

suma += (( i + 1 ) & 1 ? -1.0 :1.0)/(2\*i-1);

}

suma \*= 4;

valorPi += suma;

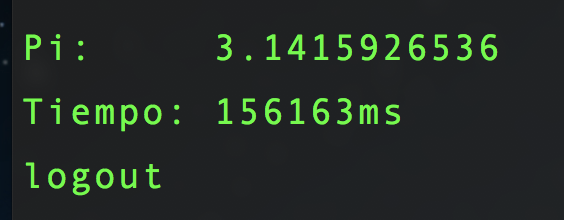
**return** NULL;

}

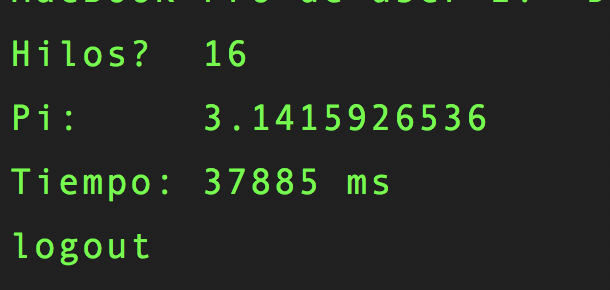
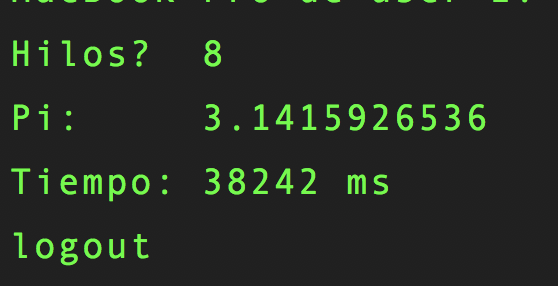
Ejecución

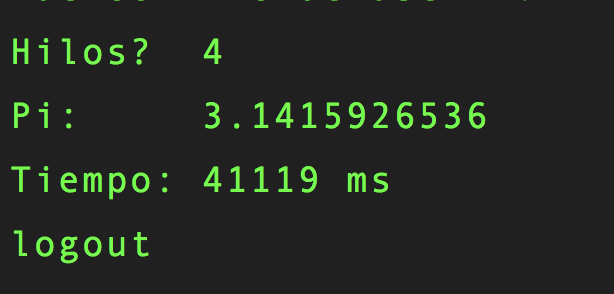
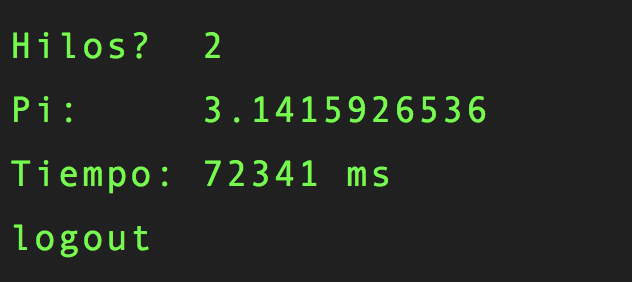
<<Inserte capturas de pantalla de una ejecución secuencial y de la ejecución de su solución paralelizada para todos los casos>>

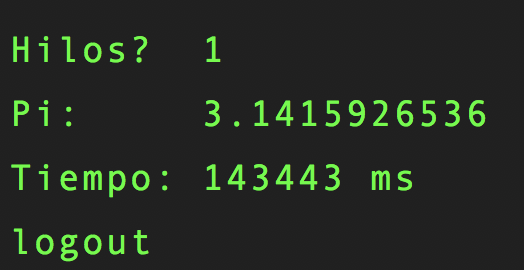
Solución Secuencial:

****

Solución Paraleliza:



Conclusiones (obligatorio):

Esta tarea en cuanto a implementación resultó ser bastante sencilla sin embargo, la parte laboriosa consistía en usar los hilos para acelerar la ejecución. Con la tarea, aprendí a utilizar los hilos y ver cómo pueden ayudar a que una ejecución sea más rápida.

Uno de los problemas que tuve fue que al principio utilicé la función **pow** en la sumatoria y esto hacía que tardara muchísimo más. Esto se resolvió sustituyendo la función con una operación a nivel de bits lo que hizo que corriera el programa 10 veces más rápido.

Además, como utilicé una función global a la que sumaban todos los hilos, la función de clock() no trabajaba como se esperaba así que busqué en stackoverflow una función similar en la misma librería (time.h).

Algo que no pude resolver por mi cuenta fue hacer que al finalizar el hilo retornara un valor busqué en línea pero no había una solución clara y varios posts estaban incorrectos. En lugar de retornar ese valor yo utilicé la variable global como se mencionó antes. Sin embargo esta forma no es una buena práctica y no se recomienda hacerlo. El profe me mostró como puedo retornar un valor mediante el pase de parámetro pero me di cuenta de que la ejecución tardaba un poco más por lo que decidí quedarme con mi implementación.