

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: Maris Mora, Michel

Presentación: 10 pts.
Funcionalidad: 60 pts.
Pruebas: 20 pts.

4 de junio de 2018. Tlaquepaque, Jalisco, Guadalajara

- 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 contrastar los resultados (en el tiempo por ejemplo).

Instrucciones para entrega de tarea

Esta tarea, como el resto, es **IMPRESINDIBLE** 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.

$$\pi = 4 \left(\sum_{n=1}^{\infty} \left(\frac{(-1)^{(n+1)}}{(2n-1)} \right) \right)$$
$$= \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots \right)$$

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 serie.
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 implementó 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	945942
2	551870
4	332462
8	331490
16	328246

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)

```
//Directivas al preprocesador
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//Función principal
int main(){
    //Configurando el buffer de salida
    setvbuf(stderr, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    //Declaración de variables
    clock_t start = clock();
    long long int i;
    double Acum;
    for(i=1;i<5000000000;i++){
        Acum += (((i + 1) & 1 ? -1.0 : 1.0) / (i + i - 1));
    }
    Acum*=4;
    clock_t stop = clock();
    int ms = 1000 * (stop - start) / CLOCKS_PER_SEC;
    printf("%lf ",Acum);
    printf("%d ms\n", ms);

    return 0;
}
//Desarrollo de funciones
```

Código fuente de la versión paralelizada

```
//Directivas al preprocesador
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <windows.h>
#include <inttypes.h>
//Variables Globales y nuevos tipos de datos
typedef struct {
    long long inicio, fin;
} PI;
double Acumu = 0;
//prototipos de funciones
DWORD WINAPI PIPP(void *);
//Función principal
int main() {
    //Configurando el buffer de salida
    setvbuf(stderr, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    //Declaración de variables
```

```

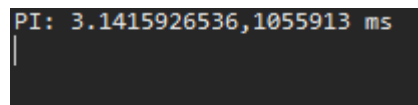
HANDLE h[16];
PI pi[16];
int hilos;
clock_t start = clock();
printf("Hilos?");
scanf("%d", &hilos);
for (int i = 0; i < hilos; i++) {
    pi[i].inicio = 50000000000 / hilos * i + 1;
    pi[i].fin = 50000000000 / hilos * (i + 1);
    h[i] = CreateThread(NULL, 0, PIPP, (void *) &pi[i], 0, NULL);
}
WaitForMultipleObjects(hilos, h, 1, INFINITE);
Acumu *= 4;
printf("PI: %.10lf\n", Acumu);
clock_t stop = clock();
int ms = 1000 * (stop - start) / CLOCKS_PER_SEC;
printf("Tiempo: %d ms\n", ms);

return 0;
}
//Desarrollo de funciones
DWORD WINAPI PIPP(void *pi) {
PI *P = (PI *) pi;
long long i;
double cont = 0;
for (i = P->inicio; i <= P->fin; i++) {
    cont += (((i + 1) & 1 ? -1.0 : 1.0) / (i + i - 1));
}
Acumu += cont;
return 0;
}

```

Ejecución

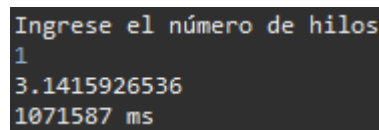
<<Inserte capturas de pantalla de una ejecución secuencial y de la ejecución de su solución paralelizada para todos los casos>>



```

PI: 3.1415926536,1055913 ms

```



```

Ingrese el número de hilos
1
3.1415926536
1071587 ms

```

```
Ingrese el número de hilos
2
3.1415926536
597632 ms

Ingrese el número de hilos
4
3.1415926536
369418 ms

Ingrese el número de hilos
8
3.1415926536
414276 ms

Ingrese el número de hilos
16
3.1415926536
363699 ms
```

Conclusiones (obligatorio):

- ✓ Lo que aprendí con esta práctica. Lo que ya sabía.
 - En el desarrollo de esta práctica aprendí mayormente dos cosas, la primera es el uso de hilos y la eficiencia que pueden brindar al ejecutar un código que puede llegar a ser complicado en una ejecución lineal, y la segunda cosa que aprendí fue que como programadores nos podemos enfrentar a ejecuciones que pueden consumir mucho tiempo, pero si tenemos en cuenta esto, podemos hacer que nuestro código funcione de una forma más eficiente mediante el uso de hilos e incluso distintas operaciones que son más fáciles para el ordenador de realizar.
- ✓ Lo que me costó trabajo y cómo lo solucioné.
 - Lo que me costó más trabajo fue detectar un error con el uso de una estructura y para solucionarlo utilicé un arreglo de estructuras, sigo sin entender por qué no funcionaba con una simple estructura.
- ✓ Lo que no pude solucionar.
 - Todo pudo ser solucionado