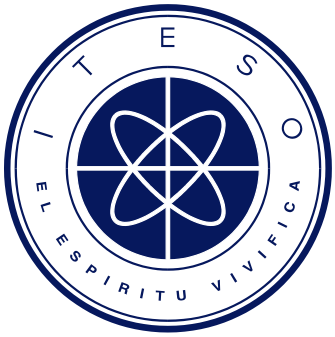
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 2. APUNTADORES A FUNCIONES

Autor: Arvizu Barragán, Paulina

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

4 de junio de 2018. Tlaquepaque, Jalisco,

- 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.

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

$$= \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right)$$

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 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	1 871 450
2	2 022 969
4	1 157 330
8	974 542
16	675 439

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.

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)

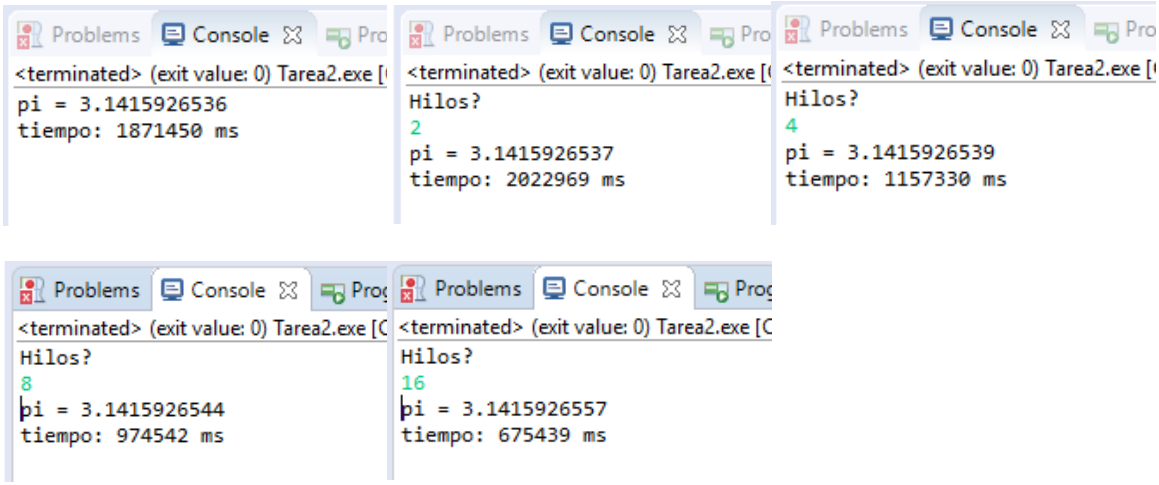
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void) {

    //programa secuencial
    clock_t start = clock();
    double pi = 0;

    for(unsigned long long int i = 1; i < 50000000000; i++) {
        if((i&1) == 0)
            pi += (-1.0) / ((2 * i) - 1);
        else
            pi += 1.0 / ((2 * i) - 1);
    }

    pi *= 4;

    printf("pi = %.10f\n", pi);

    clock_t stop = clock();
    int ms = 1000 * (stop - start)/CLOCKS_PER_SEC;
    printf("tiempo: %d ms", ms);

    return EXIT_SUCCESS;
}
```

Código fuente de la versión paralelizada

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef unsigned long long int ull;

typedef struct {
    ull inicio, fin;
    double sum;
}Rango;

DWORD WINAPI suma(void *param);

int main(void) {
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);
    //programa con hilos

    int n;
    double pi = 0;

    puts("Hilos?");
```

```
24.     scanf("%d", &n);
25.
26.     clock_t start = clock();
27.
28.     Rango *rango = (Rango*)malloc(n * sizeof(Rango));
29.     ull x = 50000000000 / n;
30.
31.     for(int i = 0; i < n; i++) {
32.         rango[i].inicio = (x * i) + 1;
33.         rango[i].fin = x * (i + 1);
34.         rango[i].sum = 0;
35.     }
36.
37.     HANDLE *h = (HANDLE*)calloc(n, sizeof(HANDLE));
38.
39.     for(int i = 0; i < n; i++)
40.         h[i] = CreateThread(NULL, 0, suma, (void *)&rango[i], 0, NULL);
41.
42.     WaitForMultipleObjects(n, h, TRUE, INFINITE);
43.
44.     for(int i = 0; i < n; i++)
45.         pi += rango[i].sum;
46.
47.     pi *= 4.0;
48.
49.     clock_t stop = clock();
50.     int ms = 1000 * (stop - start)/CLOCKS_PER_SEC;
51.
52.     printf("pi = %.10f\n", pi);
53.     printf("tiempo: %d ms", ms);
54.
55.     return EXIT_SUCCESS;
56. }
57.
58. DWORD WINAPI suma(void *param){
59.     Rango *r = (Rango*)param;
60.     for(ull i = r->inicio; i < r->fin; i++) {
61.         if((i&1) == 0)
62.             r->sum += (-1.0) / ((2 * i) - 1);
63.         else
64.             r->sum += 1.0 / ((2 * i) - 1);
65.         }
66.     return 0;
67. }
```

Ejecución



Conclusiones (obligatorio):

Esta tarea me ayudó a repasar el tema de hilos, ya que siento que no dedicamos mucho tiempo para hacer ejercicios en clase. Ver los ejemplos en Moodle me ayudó mucho a tener una idea de cómo hacer mi código con más de un hilo. Hacer el algoritmo fue en general sencillo, pero la parte en la que tuve más problemas fue en encontrar un error de tipo de dato en mi código, que hacía que el programa no funcionara con el límite establecido en la tarea.

Al comparar los tiempos del programa dependiendo de la cantidad de hilos creados pude ver la gran utilidad de estos en códigos muy grandes y cómo optimizan el tiempo de ejecución notablemente.