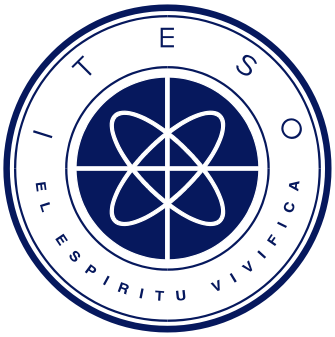
SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 2. APUNTADORES A FUNCIONES

Autor: Vázquez Martínez Ignacio Itamar

Pruebas: 20 pts

Funcionalidad: 50 pts

Presentación: 5 pts

4 de junio de 2018. Tlaquepaque, Jalisco,

La documentación de pruebas implica:
1) Descripción del escenario de cada prueba
2) Ejecución de la prueba
3) Descripción y análisis de resultados.

Todas las figuras e imagenes deben tener un título y utiilizar una leyenda que incluya número de la imagen ó figura y una descripción de la misma. Adicionalmente, debe de existir una referencia a la imagen en el texto.

Se tienen detalles de ortografía

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	1.533507
2	1.892010
4	1.865825
8	2.586665
16	2.825648

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)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define enter printf("\n");
int main() {
    clock_t t;
    t = clock();
    double total_time;
    long long int n;       // Number of iterations and control variable
    int i;
    double pi = 0;
    double temp = 0;
    double extra;
    printf("Approximation of the number PI through the Leibniz's series\n");
    //    printf("\nEnter the number of iterations: ");
    //    scanf("%lld",&n);
    printf("\nPlease wait. Running...\n");
    n = 50000;
    //      50000000000
    //      50000000000
    //      900000000
    for(i = 1; i <= n; i++){
        pi+= ((i+1)&1 ? -1.0 :1.0)/(2*i-1);
    }
```

```
    total_time =(clock()-(double)t);
```

```
    printf("This is the total time: %f",total_time/CLOCKS_PER_SEC);
    printf("\nAproximated value of PI = %1.16lf\n", 4*pi);
}
```

Código fuente de la versión paralelizada

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#define enter printf("\n");
#define max 50000000
Typedef struct{
    long long int inicio;
    long long int fin;
    long long int sumaparcial;
}rango;
Int part1 =0;
Int part2 = 0;
Int part4 = 0;
Int part8 = 0;
Int part16 = 0;
Long long int sum [16]={0};

void* sum_array1(void* arg);
void* sum_array2(void* arg);
void* sum_array4(void* arg);
void* sum_array8(void* arg);
void* sum_array16(void* arg);

#include <stdio.h>
```

```c
#include <stdlib.h>
#include <time.h>
#define enter printf("\n");

int main() {
    clock_t t;
    t = clock();
    double total_time;
    long long int n;       // Number of iterations and control variable
    int i;
    double pi = 0;
    double temp = 0;
    int num_threads = 1;
    printf("Approximation of the number PI through the Leibniz's series\n");
    printf("\nEnter the number of threads: ");
    scanf("%d",&num_threads);
    pthread_t threads[num_threads];
    switch (num_threads) {
        case 1:
            printf("Se ejecuto case1");
            for (int i = 0; i < num_threads; i++)
                pthread_create(&threads[i], NULL, sum_array1, (void*)NULL);



            for (int i = 0; i < num_threads; i++)
                pthread_join(threads[i], NULL);

            int total_sum1 = 0;
            for (int i = 0; i < num_threads; i++)
                total_sum1 += sum[i];
```

```
        break;
    case 2:
        printf("Se ejecuto case1");
        for (int i = 0; i < num_threads; i++)
            pthread_create(&threads[i], NULL, sum_array2, (void*)NULL);

        for (int i = 0; i < num_threads; i++)
            pthread_join(threads[i], NULL);

        int total_sum2 = 0;
        for (int i = 0; i < num_threads; i++)
            total_sum2 += sum[i];
        break;
    case 4:
        printf("Se ejecuto case1");
        for (int i = 0; i < num_threads; i++)
            pthread_create(&threads[i], NULL, sum_array4, (void*)NULL);

        for (int i = 0; i < num_threads; i++)
            pthread_join(threads[i], NULL);

        int total_sum4 = 0;
        for (int i = 0; i < num_threads; i++)
            total_sum4 += sum[i];
        break;
    case 8:
        printf("Se ejecuto case1");
        for (int i = 0; i < num_threads; i++)
            pthread_create(&threads[i], NULL, sum_array8, (void*)NULL);
```

```
        for (int i = 0; i < num_threads; i++)
            pthread_join(threads[i], NULL);


        int total_sum8 = 0;
        for (int i = 0; i < num_threads; i++)
            total_sum8 += sum[i];
        break;
    case 16:
        printf("Se ejecuto case1");
        for (int i = 0; i < num_threads; i++)
            pthread_create(&threads[i], NULL, sum_array16, (void*)NULL);


        for (int i = 0; i < num_threads; i++)
            pthread_join(threads[i], NULL);

        int total_sum16 = 0;
        for (int i = 0; i < num_threads; i++)
            total_sum16 += sum[i];
        break;

    default:
        break;
}


printf("\nPlease wait. Running...\n");
n = 500000000;
//      50000000000
```

```
    //      50000000000
    //      900000000
    for(i = 1; i <= n; i++){
        pi+= ((i+1)&1 ? -1.0 :1.0)/(2*i-1);
    }
    total_time =(clock()-(double)t);
    printf("This is the total time: %f",total_time/CLOCKS_PER_SEC);
    printf("\nAproximated value of PI = %1.16lf\n", 4*pi);
}

void* sum_array1(void* arg)
{

    // Each thread computes sum of 1/4th of array
    int thread_part = part1++;

    for (int i = thread_part * (MAX / 1); i < (thread_part + 1) * (MAX / 1); i++)
        sum[thread_part] += ((i+1)&1 ? -1.0 :1.0)/(2*i-1);

    return sum;
}
void* sum_array2(void* arg)
{

    // Each thread computes sum of 1/4th of array
    int thread_part = part1++;

    for (int i = thread_part * (MAX / 2); i < (thread_part + 1) * (MAX / 2); i++)
        sum[thread_part] +=((i+1)&1 ? -1.0 :1.0)/(2*i-1);
```

```
    return sum;
}
void* sum_array4(void* arg)
{

    // Each thread computes sum of 1/4th of array
    int thread_part = part4++;

    for (int i = thread_part * (MAX / 4); i < (thread_part + 1) * (MAX / 4); i++)
        sum[thread_part] +=((i+1)&1 ? -1.0 :1.0)/(2*i-1);

    return sum;
}
void* sum_array8(void* arg)
{

    // Each thread computes sum of 1/4th of array
    int thread_part = part8++;

    for (int i = thread_part * (MAX / 8); i < (thread_part + 1) * (MAX / 8); i++)
        sum[thread_part] +=((i+1)&1 ? -1.0 :1.0)/(2*i-1);

    return sum;
}
void* sum_array16(void* arg)
{

    // Each thread computes sum of 1/4th of array
```

```
    int thread_part = part16++;


    for (int i = thread_part * (MAX / 16); i < (thread_part + 1) * (MAX / 16); i++)
        sum[thread_part] += ((i+1)&1 ? -1.0 :1.0)/(2*i-1);


    return sum;
}
```

Ejecución

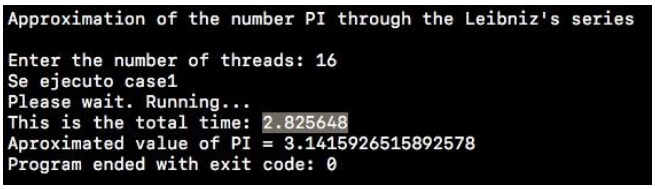
<<Inserte capturas de pantalla de una ejecución secuencial y de la ejecución de su solución paralelizada para todos los casos>>

```
Approximation of the number PI through the Leibniz's series

Enter the number of threads: 2
Se ejecuto case1
Please wait. Running...
This is the total time: 1.892010
Aproximated value of PI = 3.1415926515892578
Program ended with exit code: 0
```

```
Approximation of the number PI through the Leibniz's series

Enter the number of threads: 1
Se ejecuto case1
Please wait. Running...
This is the total time: 1.533507
Aproximated value of PI = 3.1415926515892578
Program ended with exit code: 0
```

```
Approximation of the number PI through the Leibniz's series

Enter the number of threads: 8
Se ejecuto case1
Please wait. Running...
This is the total time: 2.586665
Aproximated value of PI = 3.1415926515892578
Program ended with exit code: 0
```

```
Approximation of the number PI through the Leibniz's series

Enter the number of threads: 4
Se ejecuto case1
Please wait. Running...
This is the total time: 1.865825
Aproximated value of PI = 3.1415926515892578
Program ended with exit code: 0
```



Conclusiones (obligatorio):

- ✓ Lo que aprendí con esta práctica. Lo que ya sabía.
- ✓ Lo que me costó trabajo y cómo lo solucioné.
- ✓ Lo que no pude solucionar.

Con esta practica aprendi como se usan los hilos y con que cosas tienes que tener cuidado cuando los usas como por ejemplo con la variables globales. Ya sabia como trabajar con funciones de punteros. Me costó bastante trabajo lograr que mi programa funcionara con numeros grandes de la serie porque por alguna razon mi maquina se tardaba horas en hacer el proceso. Lo solucioné separando las operaciones del tiempo que le tardaba, pero no quedo del todo bien. Tambien lo separe en funciones diferentes para que fuera mas facil abordar el tema de las variables globales.