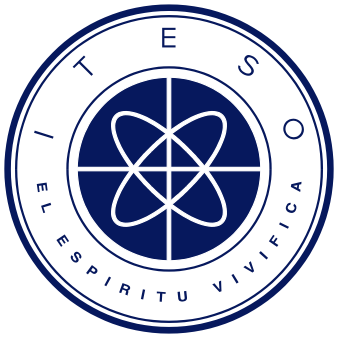
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 2. APUNTADORES A FUNCIONES

Autor: Aguilar Aguirre, Alfonso

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	559,941
2	561,334
4	280,335
8	140,158
16	70,115

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 <time.h>

int main(){

    clock_t start = clock();
    long long n=50000000000;
    long double x = 0.0;
    long long i;
    for(i=1;i<=n;i++){
        if((i&1)==1)
            x+=(long double)(1)/(long double)((2*i)-1);
        else
            x+=(long double)(-1)/(long double)((2*i)-1);
    }
    clock_t stop = clock();
    int ms = 1000 * (stop - start)/CLOCKS_PER_SEC;
    printf("Pi:     %.10Lf\n", 4*x);
    printf("Tiempo: %d ms\n", ms);
    return 0;
}
```

Código fuente de la versión paralelizada

```
#include <stdio.h>

#include <time.h>

#include <stdlib.h>
#include <pthread.h>

typedef struct{
    long long inicio;
    long long fin;
    long double sumaParcial;
}Rango;

long double serieGregory(Rango);
void * suma(void *param);

int main(int argc, const char * argv[]) {
```

```
    clock_t start = clock();

    int nHilos,i,j=0;
    scanf("%d",&nHilos);
    long long incremento=50000000000/nHilos;
    Rango *rango =(Rango *)malloc(sizeof(Rango )*nHilos);
    rango->inicio=1;
    rango->fin=incremento;
    rango->sumaParcial=0.0;
    for(i=1;i<nHilos;i++){
        (rango+i)->inicio=(rango+j)->inicio+(rango+j)->fin;
        (rango+i)->fin=(rango+j)->fin+incremento;
        (rango+i)->sumaParcial=0.0;
        j+=1;
    }

    pthread_t *r =(pthread_t *)malloc(sizeof(pthread_t )*nHilos);
    for(i=0;i<nHilos;i++){
        *(r+i)=NULL;
        pthread_create(&r[i], NULL, suma, (void *) &rango[i]);
        pthread_join(r[i] , NULL);
    }
    long double resultado;
    for(i=0;i<nHilos;i++){
        resultado+=4*((rango+i)->sumaParcial);
    }

    clock_t stop = clock();

    int ms = 1000 * (stop - start)/CLOCKS_PER_SEC;
    printf("Hilos:  %d\n", nHilos);
    printf("Pi:     %.10Lf\n", resultado);
    printf("Tiempo: %d ms\n", ms);

    return 0;
}

void * suma(void *param){
    Rango *r=(Rango*)param;
    long long i;
    for(i=r->inicio;i<=r->fin;i++){
        if((i&1)==1)
            r->sumaParcial+=(long double)(1)/(long double)((2*i)-
1);
        else
            r->sumaParcial+=(long double)(-1)/(long double)((2*i)-
1);
    }
    return NULL;
```

```
}
```

Ejecución

```
Pi:     3.1415926536
Tiempo: 690703 ms
Program ended with exit code: 0
```

```
1
Hilos:  1
Pi:     3.1415926536
Tiempo: 559941 ms
Program ended with exit code: 0
```

```
2
Hilos:  2
Pi:     3.1415926536
Tiempo: 561334 ms
Program ended with exit code: 0
```

```
4
Hilos:  4
Pi:     3.1415926535
Tiempo: 280635 ms
Program ended with exit code: 0
```

```
8
Hilos:  8
Pi:     3.1415926535
Tiempo: 140158 ms
Program ended with exit code: 0
```

```
16
Hilos:  16
Pi:     3.1415926534
Tiempo: 70115 ms
Program ended with exit code: 0
```

Conclusiones (obligatorio):

- ✓ Lo que aprendí con esta práctica. Lo que ya sabía.
  Aprendí muy bien el manejo de los hilos y reforce el uso de apuntadores como un método mucho mas práctico que tener una función que regresse un párametro y ese párametro utilizarlo.

  Ademas pude observar la gran diferencia que hay en la inclusión de hilos, realmente la disminución en el tiempo que tomo la ejecución fue brutal, pasando de varios minutos a solo poco mas de un minuto.

- ✓ Lo que me costó trabajo y cómo lo solucioné.
  Me costó mas trabajo pensar en la lógica correcta para la implementación de hilos dinámicos, es decir, poder cambiar según se necesitara un arreglo para los hilos, mas después de pensarlo como un tipo de dato fue fácil utilizar la misma lógica.

- ✓ Lo que no pude solucionar.
- ✓ No hubo algo me parece ser en esta ocasión que no pudiera solucionar, fue de gran ayuda el ejemplo del profesor para la suma con hilos.