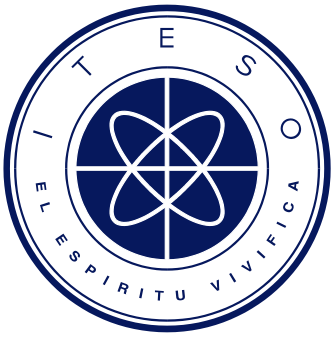
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 2. APUNTADORES A FUNCIONES

Autor: Méndez Álvarez, Raúl

CALIFICACIÓN: 100 pts

En un futuro tendrás la oportunidad de analizar las diversas técnicas que existen para calcular el valor de PI y aplicar el análisis de algoritmos para determinar su complejidad.

4 de junio de 2018. Tlaquepaque, Jalisco,

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	
2	
4	
8	
16	

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)

```
//Códlgo fuente de a versôn setial:
#include <stdio.h>
#include <time.h>

double CalculoPi();
int CalculoDenominador(int i);
int main()
{
    clock_t start = clock();
    double pi = 4*CalculoPi(5000000000);

    clock_t stop = clock();
    int ms = 1000 * (stop - start)/(int)CLOCKS_PER_SEC;
    printf("%1.10lf \n", pi);
    printf(" %d ms", ms);
    return 0;
}

double CalculoPi(int n)
{
    long int i;
    double acumpositivo=0;
    double acumNegativo=0;
    for( i=1;i<n;i++)
    {
        if( i%2==0)//si este elemento de la serie es impar
        {
            acumpositivo +=1/(double)CalculoDenominador(i);
        }
        else//par
        {
        acumNegativo += 1/(double)CalculoDenominador(i);
        }
    }
    return acumNegativo-acumpositivo;
}

int CalculoDenominador(int i)
{
    return (i*2)-1 ;
}
```

Código fuente de la versión paralelizada

```
//Código fuente de Ia version paralela (con varios hilos):
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#define cicles 50000000000

void pos1(void * arg)
{
    double *acc= (double*)arg;//casteamos apuntador
    long int i;
    *acc=0;
    for(i= 1;i<cicles;i+=8)
    * acc+=1/(float)i;
}

void pos2(void * arg)
{
    double *acc=(double*)arg; //casteamos apuntador
    long int i;
    *acc=0;
    for(i=3;i<cicles;i+=8)
    * acc+=1/(float)i;
}

void pos3(void * arg)
{
    double *acc= (double *)arg; //casteamos apuntador
    long int i;
    *acc=0;
    for(i=5;i<cicles;i+=8)
    *acc+=1/(float)i;
}

int main()
{
    pthread_t h1;
    pthread_t h2;
    pthread_t h3;
    double acc1=0,acc2=0,acc3=0,accMain=0;

    //tommamas el tiempo de inicio
    clock_t start = clock();

    //creamos hilos
    pthread_create(&h1,NULL,pos1,&acc1);
    pthread_create(&h2,NULL,pos2,&acc2);
    pthread_create(&h3,NULL,pos3,&acc3);
```

```
    long int i;
    for(i=7;i<cicles;i+=8)
        accMain+=1/(float)i;

    //unimos hilos
    pthread_join(h1,NULL);
    pthread_join(h2,NULL);
    pthread_join(h3,NULL);

//tomamos el tiempo de terminaciOn
    clock_t stop = clock();

    int ms = 1000 * (stop - start)/(int)CLOCKS_PER_SEC;
    double pi= 4*(acc1-acc2+acc3-accMain);
    printf("%1.10lf\n",pi);
    printf("%d ms", ms);

    return 0;
}
```

Ejecución:

Secuencial:

```
3.1415926553
68559 ms
```

Paralelizada:

```
3.1415926550
 4176 ms
```

Conclusiones (obligatorio):

En un inició, busqué codear mi solución empleando el método por series infinitas (me parece que fue propuesto por Leibniz), pero no logré identificar el porqué del devolverme ceros para el resultado final de la suma de Pi. Siendo que desarrollé el código para un solo hilio/no parametrizado para cómputo paralelo, se podría pensar que codear dicha serie habría de ser fácil si simplemente asignáramos cada uno de los términos a cada hilo posteriormente – tal cosa se volvía un tanto lenta, pues -a pesar de tratarse de operaciones simultáneas, el número de las mismas seguía siendo bastante elevado como para verse beneficiado por el hilado de manera considerable; fue entonces cuando comencé a indagar sobre las distintas opciones para cómputo de Pi que existen (puesto que había aprendido ya del tema en otra clase).

Me topé justamente con que los métodos más exhaustivos o con mayor número de operaciones para representar la serie eran justamente el de Leibniz, el de Wallis, siendo finalmente Nilakantha uno de los más rápidos *comprimidos*, por llamarlo de algún modo.

En dicha búsqueda me topé incluso con la técnica de aproximación por aleatorización por el "Método Monte Carlo", del cual -curiosamente- se derivó la campaña que cierto profesor me había contado sobre el tirar salchichas al aire de manera aparentemente aleatoria; y es que el método, precisamente y en su versión de análisis numérico, consta del generar enormes cantidades de números de manera aleatoria sobre un plano, establecer un rango (un cuarto de círculo en el cuartil número 1 de los ejes, para el caso particular de aproximación de Pi), para entonces aproximar así según el ratio o razón del número muestras aleatorias sobre el número de las mismas que se encuentran dentro de dicho círculo – cuestión que siendo honestos me resultó un tanto sorprendente.

Y, sí, parte de esta anécdota no aparentaría pertinencia con el curso, pero vi necesario el incluirla (o expresarla, al menos) en mis conclusiones. Me doy cuenta, luego de estas pequeñas aventuras, de que esta tarea se ha llevado la bien merecida apreciación de mi parte como el mejor proyecto o tarea de cualquier área de programación que me han asignado hasta la fecha – me parece que -al menos en mi caso- me llevó a analizar el problema desde un enfoque lógico, más allá del código.

Finalmente, el problema que simplemente no supe solucionar fue el hecho de que mi código arrojó errores de segmentación al tratar de asignarle -a la función original- un parámetro de N según el hilo en que ésta habría de ejecutarse. Con esto me refiero a que el programa perdió el sentido al yo especificar, por pase de referencia de argumentos, el "segmento" de términos con el cuál los hilos h1 hasta h3 habrían de computar para finalmente atribuirlos a la suma de Pi – y creo que es justamente aquí donde se encuentra el error al momento de "partir" nuestra N en diversas secciones según el número de hilos que el usuario elija: supongo que el hecho de que todos los hilos se encuentran ahora trabajando con la misma variable de suma (donde el valor final de Pi es almacenado) es justamente lo que ocasionaría el error, pues todos los hilos inciden en la dirección del dato al momento de buscar trabajar con éste, a pesar de estos trabajar en paralelo – necesitan, finalmente, del mismo recurso.