I OBJETIVOS

- Medir y comparar la ejecución de procesos que requieren uso intensivo del CPU en arquitecturas Multi-Core.
- Utilizar la API Windows para la gestión de hilos, y de esta forma optimizar una aplicación que requiere uso intensivo del CPU para procesadores Multi-Core.

II BIBLIOGRAFÍA

- Apuntes de programación paralela en Moodle.
- Manual de las funciones de la API de Windows para la creación, manejo y sincronización de hilos.

III RECURSOS

- Una estación de trabajo con Visual Studio posterior a 2005.
- Computadoras con procesadores Multi-Core.

IV ACTIVIDADES

1 Cálculo de distancia entre aviones que se mueven en el espacio aéreo

En un espacio aéreo de 3 dimensiones (*x*,*y*,*z*) como el que se muestra en la Figura 1 hay mil aviones en movimiento. Para evitar colisiones entre los aviones, estos deben estar a una distancia mínima que está en función de su velocidad, si un avión vuela casi al nivel del mar su velocidad es 300 Km/h, si vuela a 42,000 pies vuela a 900 Km/h. Los aviones inician en una posición aleatoria y conforme transcurre el tiempo (cada iteración) estos se mueven en el espacio de acuerdo a una dirección.

Si un avión *a* detecta que está a una distancia menor que la mínima de otro avión *b*, entonces se debe enviarles un mensaje de advertencia a ambos aviones.

El problema de los aviones es un problema de cómputo intensivo ya que requiere en cada iteración calcular la distancia de cada avión con respecto a todos los demás que están en el espacio, de manera que si son 1000 aviones, cada iteración requiere

calcular $(1000^2+1000)/2$ distancias basándonos en las posiciones *x*,*y*,*z* de cada avión.

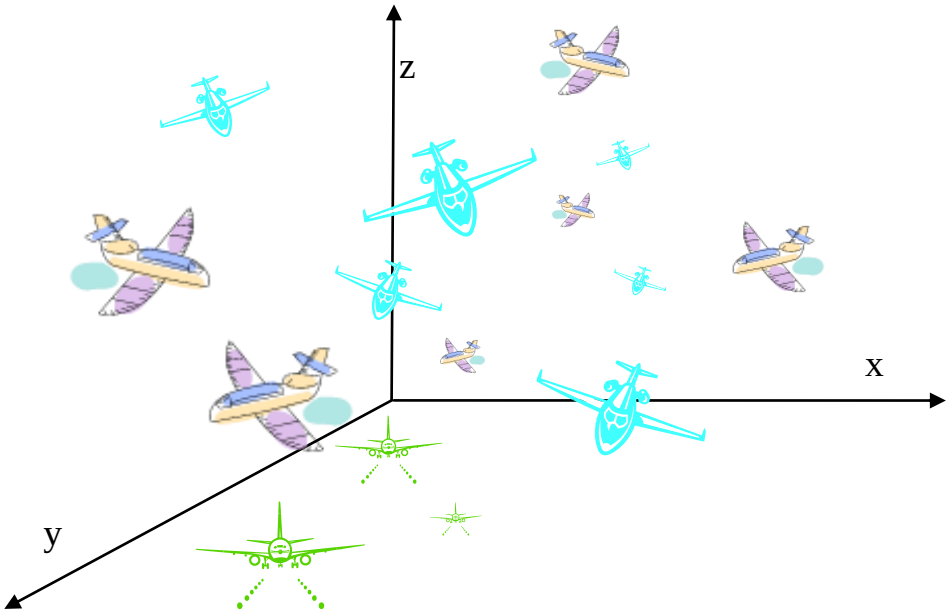

Figura 1. Aviones en el espacio aéreo

1.1 Primera versión paralela

La versión paralela del programa que calcula la distancia y envía advertencias a los aviones que se muestra en el Ejemplo 1 tiene al menos dos problemas de rendimiento.

Uno de los problemas de rendimiento de esta versión paralela que calcula la distancia y envía advertencias a los aviones se debe a que crea hilos y espera que terminen cada iteración como puede apreciarse en el código que está en la Figura 2 y en la gráfica de la Figura 3 se muestra como el hilo principal crea y termina hilos en cada iteración. Esto además de causar un problema de rendimiento, genera un desperdicio de recursos del sistema, ya que cada hilo creado requiere recursos del sistema operativo.

El otro problema de rendimiento de esta versión paralela que calcula la distancia y envía advertencias a los aviones se debe a la protección de datos que pueden ser accedidos y modificados por todos los hilos.

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

#define THREADS 4

#define PI 3.14159

#define SIN(a) sin(a*PI/180.0)
#define COS(a) cos(a*PI/180.0)

#define MAX 1000
#define MINSPEED 300.0  // Km/h
#define MAXSPEED 900.0  // Km/h
#define MAXALT 13000.0  // Metros
#define MINDIST 4000.0 // Distancia mínima en metros entre aviones si estos
están

struct AVION
{
      double pos_x;
      double pos_y;
      double pos_z;
      double currspeed;
      double dir;
      double incl;
      int warnings;
};

struct AVION avion[MAX];
int totwarnings=0;
CRITICAL_SECTION sc;

double dist(struct AVION a,struct AVION b)
{
      double dist_x,dist_y,dist_z,dist;

      dist_x=a.pos_x-b.pos_x;
      dist_y=a.pos_y-b.pos_y;
      dist_z=a.pos_z-b.pos_z;

      dist=sqrt(dist_x*dist_x+dist_y*dist_y+dist_z*dist_z);
      return(dist);
}

double distmin(struct AVION a,struct AVION b)
{
      double speed;
      double distmin;
```

```
	if(a.currspeed>b.currspeed)
		speed=a.currspeed/3.6;
	else
		speed=b.currspeed/3.6;

	distmin=MINDIST+speed*speed*1.6;
	return(distmin);
}



void inicializa_aviones(struct AVION *x)
{
	int i;
	for(i=0;i<MAX;i++)
	{
		x[i].pos_x=1000*(5000.0 - (double) (rand()%10000));
		x[i].pos_y=1000*(5000.0 - (double) (rand()%10000));
		x[i].pos_z=(double) (rand()%13000);
		x[i].currspeed=MINSPEED+x[i].pos_z/21;
		x[i].dir=(double) (rand()%360);
	}
	return;
}

// Se actualiza cada 1/1000 segundos
void actualiza_avion(struct AVION *a)
{
	if(a->pos_z<13000.0)
		a->pos_z+=0.001;

	a->currspeed=MINSPEED+a->pos_z/21;

	a->pos_x+=0.001*(a->currspeed/3.6)*SIN(a->dir);
	a->pos_y+=0.001*(a->currspeed/3.6)*COS(a->dir);

	return;
}

DWORD WINAPI hilo(LPVOID arg)
{
	int hnilo=*((int *) arg);
	int rinic=(MAX/THREADS)*hnilo;
	int rfin=rinic+(MAX/THREADS);
	int i,j;

	if(hnilo==THREADS-1)
		rfin--;

	// printf("Hilo=%d, inicio=%d, fin=%d\n",hnilo,rinic,rfin);

	for(i=rinic;i<rfin;i++)
	{
		for(j=i;j<MAX;j++)
			if(i!=j)
				if(dist(avion[i],avion[j])<distmin(avion[i],avion[j]))
```

```
                        {
                                EnterCriticalSection(&sc);
                                avion[i].warnings++;
                                avion[j].warnings++;
                                totwarnings++;
                                LeaveCriticalSection(&sc);
                        }
        }
        return 0;
}


int main()
{
        int n;
        int i,j;
        int t;
        int par[THREADS];
        HANDLE tids[THREADS];

        clock_t t_inicial,t_final;

        inicializa_aviones(avion);

        InitializeCriticalSection(&sc);
        t_inicial=clock();


        for(n=0;n<2000;n++)
        {
                for(t=0;t<THREADS;t++)
                {
                        par[t]=t;
                        tids[t]=CreateThread(NULL,0,hilo,&par[t],0,NULL);
                }


                WaitForMultipleObjects(THREADS,tids,TRUE,INFINITE);
                for(i=0;i<MAX;i++)
                        actualiza_avion(&avion[i]);

                if(n%100==0)
                        printf("Total Warnings = %d\n",totwarnings);
        }


        t_final=clock();
        printf("En %3.6f segundos\n",((float) t_final- (float)t_inicial)/
CLOCKS_PER_SEC);
        DeleteCriticalSection(&sc);
}
```

Ejemplo 1. Versión paralela del programa que busca aviones a una distancia menor que la mínima en un espacio de 3 dimensiones.

```
for(n=0;n<2000;n++)
{
	for(t=0;t<THREADS;t++)
	{
		par[t]=t;
		tids[t]=CreateThread(NULL,0,hilo,&par[t],0,NULL);
	}

	WaitForMultipleObjects(THREADS,tids,TRUE,INFINITE);
	for(i=0;i<MAX;i++)
		actualiza_avion(&avion[i]);

	if(n%100==0)
		printf("Total Warnings = %d\n",totwarnings);
}
```

Figura 2.- Sección del programa que crea y termina los hilos de acuerdo repetitivamente y por lo tanto afecta el rendimiento

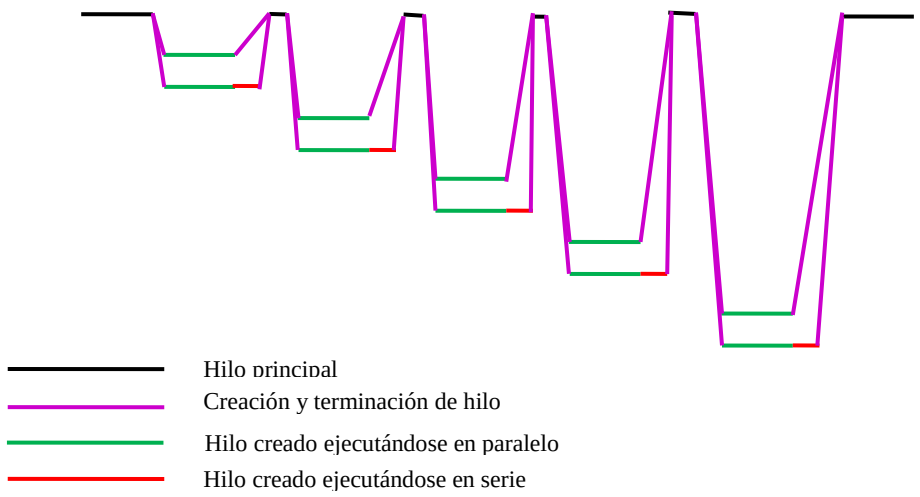

Figura 3.- Tiempo llevado en la creación y terminación de hilos en un ciclo

2 Encontrar los problemas de rendimiento

Utiliza la herramienta Parallel Amplifier y contesta las siguientes preguntas que se piden a continuación. Considera que es probable que para realizar el análisis con Parallel Amplifier tengas que disminuir el número de iteraciones en el hilo principal o el número de aviones que hay en el espacio aéreo.

1. ¿Cuál es el porcentaje de utilización del CPU durante la ejecución del programa?

2. Si la concurrencia es pobre, explica por qué.

3. ¿Cuáles son las partes del programa donde se gasta la mayor parte del tiempo?

4. ¿Cuántos hilos fueron creados y terminados durante la ejecución del programa?

3 Resolver los problemas de rendimiento a través de la técnica llamada Thread Pools con eventos de la API de Windows

En computación paralela el patrón llamado Thread Pool es donde se crea un número de hilos para ejecutar un número de tareas, las cuales usualmente están organizadas en una cola. Típicamente son mucho más tareas que hilos. Tan pronto como un hilo termina de realizar una tarea, este solicita la siguiente tarea de la cola hasta que todas las tareas son terminadas. Cuando no hay tareas para realizarse el hilo puede esperar a que haya nuevas tareas a realizar o en su caso terminar.

El número de hilos usados es un parámetro que puede establecerse para lograr el mejor rendimiento, usualmente este equivale al número de procesadores disponibles.

Para evitar los problemas de desperdicio de recursos que se describen anteriormente por la creación y terminación de hilos, modifica la versión para que esta inicie creando los hilos que esperarán el trabajo, los cuales llamaremos *hilos esclavos*. Estos *hilos esclavos* iniciarán y estarán esperando a que el hilo principal les indique que estos pueden iniciar la verificación de las distancias entre los diferentes aviones. Una vez que estos terminen el hilo principal pasará a la siguiente iteración. Utiliza eventos de la API de Windows para lograr la sincronización entre el hilo principal y los *hilos esclavos*.

V EVALUACIÓN

1 Equipos

Esta práctica se hará en equipos (máximo 2 integrantes), es necesario que en la revisión esté el equipo completo ya que el integrante que no se presente no tendrá calificación en la práctica.

2 Entrega

Subir en el apartado correspondiente en Moodle hasta el día Jueves 5 de Marzo a las 11:59 PM un archivo en formato **.zip** que contenga:

1. El documento en **formato PDF** con las respuestas a las preguntas y que contiene el análisis con Parallel Inspector que se pide en la parte IV2.

2. El o los archivos con el **programa fuente** que se piden en la parte IV3.

No incluya líneas de código en sus programas de las cuales desconozca su funcionamiento. El código no conocido será anulado en el funcionamiento de la práctica.

3 Evaluación

Puntualidad en las revisiones	El equipo estuvo completo y puntual en todas las sesiones de revisión.	Si hubo dos o más sesiones con el equipo, el equipo estuvo completo y puntual en casi todas las sesiones de revisión	Si solo hubo una sesión de revisión, el equipo no estuvo completo o no fue puntual. Si fueron dos o más sesiones de revisión, en más de una sesión el equipo no estuvo completo o fue puntual
	+10	+5	0
Funcionamiento	El producto cumple con todas las especificaciones indicadas en el documento y no tiene fallas	El producto muestra una falla no esperada o el producto está casi completo, puede funcionar excepto la parte no completada	El producto muestra más de una falla inesperada o no funciona, esto puede ser debido a que no esté completo.
	+75	+37.5	0
Interfaz con el usuario	El producto funciona y pudo ser utilizado sin necesidad de recibir indicaciones por el desarrollador, tiene instrucciones claras para ser utilizado.	El producto funciona, pero hubo necesidad de recibir alguna indicación para su uso por parte del desarrollador del producto	El producto carece de instrucciones claras para ser utilizado y requiere que alguno de los desarrolladores esté presente para su utilización o no puede utilizarse debido a que no está completo
	+10	+5	0
Claridad en el código	El código es claro, usa nombres de variables adecuadas, está debidamente comentado e indentado. Puede ser entendido por cualquier otra persona que no intervino en su desarrollo.	El código carece de claridad, puede ser entendido por cualquier persona ajena a su desarrollo pero con cierta dificultad.	El código carece de comentarios, está mal indentado, usa nombres de variables no adecuadas.
	+5	+2.5	0
Defensa del producto	Todos los integrantes son capaces de explicar cualquier parte del producto presentado	Alguno de los integrantes muestra dudas sobre alguna parte del desarrollo del producto presentado	Más de un integrante, o si el trabajo fue individual , el desarrollador duda sobre cómo está desarrollado producto.
	x 1(puntos se multiplican por 1)	x 0.5 (puntos se multiplican por 0.5)	x 0 (puntos se multiplican por 0)
Sobresaliente 20 %	Tiene 1 en todos los puntos anteriores. El producto entregado es sobresaliente, muestra tener la calidad para ser expuesto como un producto representativo de la carrera Hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado		No tiene 1 en todos los puntos anteriores, o el producto entregado no es sobresaliente y no muestra tener la calidad para ser expuesto como un producto representativo de la carrera o no hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado
	+20		0