Universidad Autónoma de San Luis Potosí
Facultad de ingeniería.
Área Ciencias De La Computación.

Sistemas Operativos

Proyecto final – Manual de programador

González Monsiváis Gerardo de Jesús.

Carrera: Ingeniería En Computación

Profesor: Ing. Hernández García Agustín

Semestre 2022-2023/II

FECHA: 02 de junio de 2023

Introducción:

En este documento se describen todos los aspectos técnicos para la realización del proyecto final del tema de sincronización de procesos correspondiente a la materia de Sistemas Operativos, el cual consiste en una simulación grafica entre la interacción de abejas y un oso tomando como patrón el modelo productor-consumidor. El código esta escrito en el lenguaje C# y este documento será de ayuda a aquel programador/a que desee realizar modificaciones al código fuente.

Diseño:

Al iniciar la compilación, se crean 2 hilos los cuales nos ayudaran a simular el trabajo continuo de las abejas y el oso. después de esto las abejas producen una cantidad aleatoria de miel y esa cantidad de miel la introducen un tarro, si el tarro alcanza su capacidad máxima (25 unidades), las abejas esperan a que el oso consuma esa miel antes de continuar produciendo mas miel. Ya que el oso consume la miel del tarro ahora debe notificar a un hilo de una abeja que el tarro este vacío para que los demás hilos(abejas) continúen trabajando. En la parte grafica se hace mediante Windows forms

Estructura del código:

El código se divide mediante clases que son en las que se realiza todo el trabajo para poder llevar a cabo la simulación, dichas clases son las siguientes:

“Clase OsoYAbejas”:

Esta clase es la principal en ella podemos encontrar diferentes atributos y métodos los cuales son los siguientes:

- Atributos:
  capacidadTarro: entero que representa la capacidad máxima del tarro de miel.
  capacidadActualTarro: entero que almacena la cantidad actual de miel en el tarro.
  sinc: objeto para sincronizar el acceso al tarro de miel.
  tarroLleno: indica si el tarro está lleno.
  tarroVacio: indica si el tarro está vacío.
  bufferMiel: instancia de la clase Buffer que se utiliza para almacenar la miel producida por las abejas.
  form1: instancia de la clase Form1 para actualizar la interfaz gráfica.

- Métodos:

ProductorAbejas: método que simula el trabajo de las abejas, produciendo miel y almacenándola en el tarro.
ConsumidorOso: método que simula al oso consumiendo la miel del tarro.
Constructor: inicializa los atributos, crea instancias de las clases Buffer y Form1, y crea los hilos para las abejas y el oso.

```
3 referencias
internal class OsoYAbejas
{
    int capacidadTarro = 25; // Acepta hasta 25 unidades de miel
    public int capacidadActualTarro = 0; // Es lo que tiene actualmente de unidades de miel
    object sinc = new object(); // object para sincronizar el acceso al tarro
    bool tarroLleno = false; // Indica si el tarro está lleno
    bool tarroVacio = true; // Indica si el tarro está vacio
    public Buffer<int> bufferMiel;
    Form1 form1 = new Form1();
    private ManualResetEvent finalizacionEvento = new ManualResetEvent(false);
    1 referencia
    public OsoYAbejas()
    {
        bufferMiel = new Buffer<int>();

        Thread abejas, oso;

        abejas = new Thread(ProductorAbejas);
        oso = new Thread(ConsumidorOso);

        abejas.Start();
        oso.Start();
    }
```

```
public void ProductorAbejas()
{
    int numAbejas = 9; //Número de abejas
    List<int> miel = new List<int>(); //Lista de enteros que cada abeja producirá de miel
    Random r = new Random();
    Formulario.imprimeTamTarro("Tamaño del tarro: " + capacidadTarro.ToString());
    Formulario.imprimeCantidadDeAbejas("Num de abejas: " + numAbejas.ToString());


    for (int i = 0; i < numAbejas; i++)
    {
        miel.Add(r.Next(1, 6)); // Genera una cantidad aleatoria de miel entre 1 y 5 para cada abeja
    }

    int abeja = 0;
    while (!finalizacionEvento.WaitOne(0)) // Bucle infinito para simular el trabajo continuo de las abejas
    {
        lock (sinc) // Bloquea el acceso al tarro de miel para evitar condiciones de carrera
        {
            capacidadActualTarro += miel[abeja]; // Añade la cantidad de miel producida por la abeja actual al tarro
            Formulario.actualizarCapacidadTarro("Capacidad actual del tarro " + capacidadActualTarro.ToString());
            Formulario.determinaCapacidadTarro(capacidadActualTarro);

            if (capacidadActualTarro >= capacidadTarro) // Verifica si el tarro está lleno
            {
                tarroLleno = true; // Marca el tarro como lleno
                tarroVacio = false;
                Formulario.actualizarEstadoAbejas("Tarro lleno, abejas dejan de trabajar");
                Monitor.Pulse(sinc); // Notificar al oso que el tarro está lleno
                Monitor.Wait(sinc); // Esperar a que el oso consuma la miel
                Formulario.actualizarEstadoAbejas("Abejas continúan con su trabajo");
                tarroLleno = false; // Marca el tarro como no lleno
                Thread.Sleep(1500);
            }
        }
        Thread.Sleep(1000);
        abeja++;
        if (abeja >= numAbejas)
            abeja = 0; // Reinicia el índice de abejas si se alcanza el final de la lista
    }
}
```

```
        21 referencias
        public Form1 Formulario { get; set; }

        1 referencia
        public void ConsumidorOso()
        {
            while (!finalizacionEvento.WaitOne(0)) // Bucle infinito para que el oso siempre esté activo
            {
                lock (sinc) // Bloquea el acceso al tarro de miel para evitar condiciones de carrera
                {
                    while (!tarroLleno) // Espera a que el tarro esté lleno
                        Monitor.Wait(sinc);

                    Formulario.actualizarEstadoOso("Oso se despierta");
                    //Console.WriteLine("Oso se despierta");
                    Thread.Sleep(1500);
                    capacidadActualTarro = 0; // Reinicia la capacidad actual del tarro a cero
                    //Console.WriteLine("Oso se come| la miel");
                    Formulario.actualizarEstadoOso("Oso se come la miel");
                    Thread.Sleep(1500);
                    tarroVacio = true; // Marca el tarro como vacío
                    Monitor.Pulse(sinc); // Notificar a una abeja que el tarro está vacío
                }
            }
        }

    }
}
```

“Clase Buffer<T>”:

- Atributos:
  cola: cola genérica utilizada para almacenar los elementos del buffer. cont: contador que lleva el registro del número de elementos en el buffer.

- Métodos:
  end: agrega un elemento al buffer, esperando si el buffer está lleno.
  Size: devuelve el número actual de elementos en el buffer.
  Receive: retira y devuelve un elemento del buffer, esperando si el buffer está vacío.

```
3 referencias
public class Buffer<T>
{
    private Queue<T> cola;
    private int cont;
    1 referencia
    public Buffer()
    {
        cola = new Queue<T>();
        cont = 0;
    }

    0 referencias
    public void Send(T item)
    {
        // SI EL BUFFER ESTÁ LLENO, ESPERAR A QUE EXISTA UN ESPACIO LIBRE
        // IMPRIMIR UN MENSAJE DICIENDO QUE EL BUFFER ESTÁ LLENO
        cola.Enqueue(item);
        cont++;
    }

    0 referencias
    public int Size()
    {
        return cont;
    }

    0 referencias
    public T Receive()
    {
        if (cola.Count == 0)
        {
            // SI EL BUFFER ESTÁ VACÍO, ESPERAR A QUE EXISTA UN DATO
            // IMPRIMIR UN MENSAJE DICIENDO QUE EL BUFFER ESTÁ VACÍO
            return default(T);
        }
```

```
            T item = cola.Dequeue();
            cont--;
            return item;
        }
    }
```