Задача 2,7,14,19

Создать класс на языке C#, который:

- называется TaskQueue и реализует логику пула потоков;
- создает указанное количество потоков пула в конструкторе;
- содержит очередь задач в виде делегатов без параметров:
- delegate void TaskDelegate();
- обеспечивает постановку в очередь и последующее выполнение делегатов с помощью метода void EnqueueTask(TaskDelegate task);

Решение:

```
public delegate void TaskDelegate();

    public class TaskQueue
    {
        private List<Thread> threads;
        private Queue<TaskDelegate> tasks;
        private int activeTaskCount;

        public TaskQueue(int threadCount)
        {
            tasks = new Queue<TaskDelegate>();
            threads = new List<Thread>();
            for (int i = 0; i < threadCount; i++)
            {
                Interlocked.Increment(ref activeTaskCount);
                var t = new Thread(DoThreadWork);
                threads.Add(t);
                t.IsBackground = true;
                t.Start();
            }
        }

        public int ThreadCount
        {
            get { return threads.Count; }
        }

        public void Close()
        {
            for (int i = 0; i < threads.Count; i++)
                DoEnqueueTask(null);
            lock (this)
            {
                while (activeTaskCount > 0)
                  Monitor.Wait(this);
            }

            foreach (Thread t in threads)
                t.Join();
        }

        public void EnqueueTask(TaskDelegate task)
        {
            if (task != null)
                DoEnqueueTask(task);
            else
                throw new ArgumentNullException("task");
        }

        private void DoEnqueueTask(TaskDelegate task)
        {
            lock (tasks)
            {
                tasks.Enqueue(task);
                Monitor.Pulse(tasks);
            }
        }

        private TaskDelegate DequeueTask()
        {
            lock (tasks)
            {
                while (tasks.Count == 0)
                    Monitor.Wait(tasks);
                TaskDelegate t = tasks.Dequeue();
                return t;
            }
        }

        private void DoThreadWork()
        {
            TaskDelegate task;
            do
            {
                task = DequeueTask();
                try
                {
                    if (task != null)
                        task();
                }
                catch (ThreadAbortException)
                {
                    Thread.ResetAbort();
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.ToString());
                }
            } while (task != null);
            lock (this)
            {
                activeTaskCount--;
                if (activeTaskCount == 0)
                    Monitor.Pulse(this);
            }
        }
    }
```

Задача 4,16

Создать класс на языке C#, который:

- называется LogFile и обеспечивает добавление указанных строк в
- текстовый журнал работы программы;
- открывает файл в конструкторе;
- реализует интерфейс IDisposable и позволяет принудительно закрыть файл с помощью метода Dispose;
- реализует метод Write(string str), записывающий указанную строку в формате <год-месяц-
- день><пробел><часы:минуты:секунды.миллисекунды><пробел><номер программного потока, вызвавшего метод><пробел><str>

Решение:

```
public class LogFile : IDisposable
    {
        private bool disposed = false;
        private StreamWriter writer;
        private string path;
        public string Path
        {
            get {return path; }
            set {path = value; }
        }

        public LogFile(string path)
        {
            this.path = path;
            writer = new StreamWriter(path);
        }

        public void Write(string str)
        {
            string time = DateTime.Now.ToString("yyyy-mm-dd HH:mm:ss");
            writer.WriteLine(string.Format("{0} {1} {2}", time,
            Thread.CurrentThread.ManagedThreadId, str));
        }

        public void Dispose()
        {
             if (!disposed)
             {
                 Dispose(true);
                 GC.SuppressFinalize(this);
                 disposed = true;
             }
        }

        protected virtual void Dispose(bool disposing)
        {
            if (disposed)
                return;

            if (disposing)
            {
                writer.Close();
                writer.Dispose();
                disposed = true;
            }
        }
    }
```

Задача 5,17

Создать класс на языке C#, который:

- называется Mutex и реализует двоичный семафор с помощью атомарной операции Interlocked.CompareExchange.
- обеспечивает блокировку и разблокировку двоичного семафора с помощью public-методов Lock и Unlock.

Решение:

```
public class Mutex
    {
        private Thread lockingThread;

        //При использовании Lock из потока в котором он ране не использовался, входжение в цикл не будет обеспечено, а просто инициализируется lockingThread.
        public void Lock()
        {
            while (Interlocked.CompareExchange(
                ref lockingThread, Thread.CurrentThread, null) != null)
            {
                Thread.Sleep(1);
            }
        }
        public void Unlock()
        {
            Interlocked.Exchange(ref lockingThread, null);
        }
    }
```

Задача 9,21

Создать на языке C# статический метод класса Parallel.WaitAll, который:

- принимает в параметрах массив делегатов;
- выполняет все указанные делегаты параллельно с помощью пула потоков;
- дожидается окончания выполнения всех делегатов. Реализовать простейший пример использования метода Parallel.WaitAll.

Решение :

```
public delegate void Task(object done);
        public static void WaitAll(Task[] task)
        {
            ManualResetEvent[] doneEvents =
            new ManualResetEvent[task.Length];
            for(int i = 0; i < task.Length; i++)
            {
                doneEvents[i] = new ManualResetEvent(false);
                ThreadPool.QueueUserWorkItem(new WaitCallback(task[i]),
                doneEvents[i]);
            }
            //Реализация Parralels.WaitAll на основе WaitHandle.WaitAll (мы ждем получения сигналов в event'ах)
            WaitHandle.WaitAll(doneEvents);
            Console.WriteLine("All tasks are complete.");
        }

        public static int CalculateFib(int n)
        {
            if (n <= 1)
            {
                return n;
            }
            return CalculateFib(n - 1) + CalculateFib(n - 2);
        }

        public static void FibNumberTask(object obj)
        {
            ManualResetEvent doneEvent = (ManualResetEvent)obj;
            Random rnd = new Random();
            Console.WriteLine(CalculateFib(rnd.Next(20,40)));
            doneEvent.Set();
        }

        static void Main(string[] args)
        {
            Task[] tasks = new Task[63];
            for (int i = 0; i < tasks.Length; i++)
                tasks[i] = FibNumberTask;
            WaitAll(tasks);
            Console.ReadKey();
        }
```

Задача 12,24

Создать на языке C# обобщенный (generic-) класс DynamicList<T>, который:

- реализует динамический массив с помощью обычного массива
- T[];
- имеет свойство Count, показывающее количество элементов;
- имеет свойство Items для доступа к элементам по индексу; - имеет методы Add, Remove, RemoveAt, Clear для соответственно добавления, удаления, удаления по индексу и удаления всех элементов;
- реализует интерфейс IEnumerable<T>.

Реализовать простейший пример использования класса DynamicList<T> на языке C#.

Решение:

```
public class DynamicList<T> : IEnumerable
    {
        private const int INCOUNTER = 100;
        private int size;
        private int currentElement;
        private T[] array;

        public int Count
        {
            get { return array.Length; }
            private set { ; }
        }

        public T this[int index]
        {
            get{ return array[index]; }
            set { array[index] = value; }
        }

        public DynamicList(int size)
        {
            this.size = size;
            array = new T[size];
            currentElement = 0;
        }
        private T[] CopyAndExtend(T[] array)
        {
            int length = array.Length + INCOUNTER;
            T[] newArray = new T[length];
            for (int i = 0; i < array.Length; i++)
                newArray[i] = array[i];
            return newArray;
        }
        public void Add(T item)
        {
            if (currentElement > array.Length - 1)
                array = CopyAndExtend(array);
            array[currentElement] = item;
            currentElement++;
        }

        public void RemoveAt(int index)
        {
            var dest = new T[array.Length - 1];
            if (index > 0)
                Array.Copy(array, 0, dest, 0, index);

            if (index < array.Length - 1)
                Array.Copy(array, index + 1, dest, index,
                array.Length - index - 1);
            currentElement--;
            array = dest;
        }

        public void Remove(T name)
        {
            int numIndex = Array.IndexOf(array, name);
            array = array.Where((val, idx) => idx != numIndex).ToArray();
            currentElement--;
        }

        public void Clear()
        {
             array = new T[size];
             currentElement = 0;
        }

        public IEnumerator GetEnumerator()
        {
            return array.GetEnumerator();
        }
    }
```

Задача 3,15,27

Создать класс на языке C#, который:

- называется OSHandle и обеспечивает автоматическое или принудительное освобождение заданного дескриптора операционной системы;
- содержит свойство Handle, позволяющее установить и получить дескриптор операционной системы;
- реализует метод Finalize для автоматического освобождения дескриптора;
- реализует интерфейс IDisposable для принудительного освобождения дескриптора;

Решение :

```
public class OSHandle : IDisposable
    {
        [DllImport("kernel32")]
        private static extern bool CloseHandle(IntPtr handle);
        private bool disposed = false;

        public OSHandle(IntPtr handle)
        {
            this.handle = handle;
        }

        ~OSHandle()
        {
            Dispose(false);
        }

        private IntPtr handle;
        public IntPtr Handle
        {
            get
            {
                if (!disposed)
                    return handle;
                else
                    throw new ObjectDisposedException(ToString());
            }
            set
            {
                if (!disposed)
                    handle = value;
                else
                    throw new ObjectDisposedException(ToString());
            }
        }

        public void Dispose()
        {
            if (!disposed)
            {
                Dispose(true);
                GC.SuppressFinalize(this);
                disposed = true;
            }
        }

        protected virtual void Dispose(bool disposing)
        {
            if (handle != IntPtr.Zero)
                CloseHandle(handle);
            if (disposing)
            {
                //Dispose
            }
            else
            {
                //Finalize
            }
        }
    }
```

Задача 6,10,22

Реализовать консольную программу на языке C#, которая:

- принимает в параметре командной строки путь к сборке .NET
- (EXE- или DLL-файлу);
- загружает указанную сборку в память;
- выводит на экран полные имена всех public-типов данных этой сборки, упорядоченные по пространству имен (namespace) и по имени.

Решение :

```
Assembly assembly = Assembly.LoadFile(Console.ReadLine());
List<Type> t = assembly.GetTypes().ToList();
t = t.OrderBy(x => (x.ToString())).ToList();
foreach (Type m in t)
    Console.WriteLine(m.ToString());
Console.ReadLine();
```

C ExportClass

Решение:

```
[AttributeUsage(AttributeTargets.Class)]
public class ExportClass : Attribute
{
    //Вид строки для пометки класса атрибутом[ExportClass(true/false)]
    private bool available;
    public bool Available
    {
        get { return available; }
        set { available = value; }
    }

    public ExportClass(bool availability)
    {
        available = availability;
    }

    public override string ToString()
    {
        return "IsWindowEnable: " + ((available) ? "Enabled" : "Disabled")
        + ".";
    }
}

static void Main(string[] args)
{
    Assembly assembly = Assembly.LoadFile(Console.ReadLine());
    List<Type> t = assembly.GetTypes().ToList();
    t = t.OrderBy(x => (x.ToString())).ToList();
    foreach (Type m in t)
    {
        foreach (Attribute attrityte in m.GetCustomAttributes().ToList())
        {
            if (attrityte is ExportClass)
            {
                Console.WriteLine(m.ToString());
                break;
            }
        }
    }
    Console.ReadLine();
}
```

Задача 8,20

Создать класс на языке C#, который:
Создать класс LogBuffer, который:

- представляет собой журнал строковых сообщений;
- предоставляет метод public void Add(string item);
- буферизирует добавляемые одиночные сообщения и записывает их пачками в конец текстового файла на диске;
- периодически выполняет запись накопленных сообщений, когда их количество достигает заданного предела;
- периодически выполняет запись накопленных сообщений по истечение заданного интервала времени (вне зависимости от наполнения буфера);
- выполняет запись накопленных сообщений асинхронно с добавлением сообщений в буфер;

Решение:

```
public class LogBuffer : IDisposable
{
    private bool disposed = false;
    string filePath = "file.txt";
    int flushTime = 1000;
    int flushCount = 5;
    StreamWriter writer;
    Timer timer;

    ConcurrentQueue<string> list = new ConcurrentQueue<string>();

    public LogBuffer()
    {
        try
        {
            writer = new StreamWriter(new FileStream(filePath,
            FileMode.Append, FileAccess.Write));
        }
        catch
        {
            if (writer != null)
                writer.Dispose();
            throw new Exception("gg");
        }
        timer = new Timer(TimerCallback, null, flushTime, -1);
    }

    public void Add(string item)
    {
        list.Enqueue(item);
        if (list.Count >= flushCount)
        {
            WriteBufferAsync();
        }
    }

    private void WriteBufferAsync()
    {
        while (list.Count != 0)
        {
            string item;
            if (list.TryDequeue(out item))
            {
                writer.WriteLineAsync(item);
            }
        }
    }

    private void TimerCallback(object state)
    {
        WriteBufferAsync();
    }

    public void Dispose()
    {
        if (!disposed)
        {
            Dispose(true);
            GC.SuppressFinalize(this);
            disposed = true;
        }
    }

    protected virtual void Dispose(bool disposing)
    {
        if (disposing)
        {
            WriteBufferAsync();
            writer.Flush();
            writer.Dispose();
        }
    }
}
```

Задача 11,23

Создать на языке C# статический метод Where статического класса EnumeratorExtension, который:

- имеет вид:
  public static IEnumerable<T> Where(this IEnumerable<T> source, Func<T, bool> predicate);
- реализует шаблон Enumerator и возвращает только те элементы переданного виртуального списка source, для которых делегат predicate возвращает значение true.

Реализовать простейший пример использования метода EnumeratorExtension.Where.

Решение:

```
public static class EnumeratorExtension<T>
{
    public static IEnumerable<T> Where(IEnumerable<T> source, Func<T, bool>
predicate)
    {
        foreach (T item in source)
        {
            if (predicate(item))
            {
                yield return item;
            }
        }
    }
}
public static void Example()
{
    List<int> list = new List<int>();
    for (int i = 0; i < 20; i++)
    {
        list.Add(i);
    }
    foreach (int i in EnumeratorExtension<int>.Where(list, e => e % 2 == 1))
    {
        Console.WriteLine(i);
    }
    Console.ReadLine();
}
```

Задача 1,13,25

Реализовать консольную программу на языке C#, которая:

- принимает в параметре командной строки путь к исходному и целевому каталогам на диске;
- выполняет параллельное копирование всех файлов из исходного каталога в целевой каталог;
- выполняет операции копирования параллельно с помощью пула потоков;
- дожидается окончания всех операций копирования и выводит в консоль информацию о количестве скопированных файлов.

Решение:

```
public static List<string> GetLocalPath<T>(List<T> directories, string path) where T :
FileSystemInfo
        {
            List<string> result = new List<string>();
            foreach(T dir in directories)
                result.Add(dir.FullName.Substring(path.Length + 1));
            return result;
        }

        public static void CopyFile(object o)
        {
            KeyValuePair<KeyValuePair<string, string>, ManualResetEvent> param =
            (KeyValuePair<KeyValuePair<string, string>, ManualResetEvent>)o;
            File.Copy(param.Key.Key, param.Key.Value);
            param.Value.Set();
        }

        public static void CopyDirectories(string source, string dest)
        {
            DirectoryInfo info = new DirectoryInfo(source);
            string sourceDirectoryName = info.Name;
            string destDirectoryPath = dest + "\\" + sourceDirectoryName;
            Directory.CreateDirectory(dest + "\\" + sourceDirectoryName);
            List<DirectoryInfo> directories = info.GetDirectories("*.*",
            SearchOption.AllDirectories).ToList();
            List<FileInfo> files = info.GetFiles("*.*",
            SearchOption.AllDirectories).ToList();
            List<string> localDirectoriesPath = GetLocalPath(directories, source);
            foreach (string dir in localDirectoriesPath)
                Directory.CreateDirectory(dest + "\\" + sourceDirectoryName + "\\" +
                dir);
            List<string> filesToCopy = GetLocalPath(files, source);
            Dictionary<string, string> copyInfo = new Dictionary<string, string>();
            int i = 0;
            foreach (FileInfo file in files)
            {
                copyInfo.Add(file.FullName, dest + "\\" + sourceDirectoryName + "\\" +
                filesToCopy[i]);
                i++;
            }
            ManualResetEvent[] doneEvents = new ManualResetEvent[files.Count];
```

```
    KeyValuePair<KeyValuePair<string, string>, ManualResetEvent> param =
    new KeyValuePair<KeyValuePair<string, string>, ManualResetEvent>();
    i = 0;
    foreach (KeyValuePair<string, string> kvp in copyInfo)
    {
        doneEvents[i] = new ManualResetEvent(false);
        param = new KeyValuePair<KeyValuePair<string, string>,
        ManualResetEvent>(new KeyValuePair<string, string>(kvp.Key, kvp.Value),
        doneEvents[i]);
        ThreadPool.QueueUserWorkItem(new WaitCallback(CopyFile), param);
        i++;
    }
    WaitHandle.WaitAll(doneEvents);
}

static void Main(string[] args)
{
    string sourceDirectory = args[0];
    string destDirectory = args[1];
    CopyDirectories(sourceDirectory, destDirectory);
    Console.ReadLine();
}
```