Бруцкая Анастасия

Лабораторная работа №2

Задание 1

Mutex.cs

namespace lapa2.\_1;

public class Mutex

{

private int state = 0; // 0 - свободен, 1 - занят

public void Lock()

{

while (Interlocked.CompareExchange(ref state, 1, 0) != 0)

{

Thread.Yield(); // Уступаем процессор другим потокам

}

}

public void Unlock()

{

Interlocked.Exchange(ref state, 0);

}

}

Program.cs

namespace lapa2.\_1;

class Program

{

static Mutex mutex = new Mutex();

static int sharedResource = 0;

static void Worker()

{

for (int i = 0; i < 5; i++)

{

mutex.Lock();

int current = sharedResource;

Thread.Sleep(00); // Симуляция работы

sharedResource = current + 1;

Console.WriteLine($"Поток {Thread.CurrentThread.ManagedThreadId} увеличил sharedResource до {sharedResource}");

mutex.Unlock();

}

}

static void Main()

{

Thread t1 = new Thread(Worker);

Thread t2 = new Thread(Worker);

Thread t3 = new Thread(Worker);

t1.Start();

t2.Start();

t3.Start();

t1.Join();

t2.Join();

t3.Join();

Console.WriteLine($"Финальное значение sharedResource: {sharedResource}");

}

}

Задание 2

OsHandle.cs

using System.Runtime.InteropServices;

namespace lapa2.\_2;

public class OsHandle : IDisposable

{

private IntPtr \_handle = IntPtr.Zero;

private bool \_disposed = false;

public IntPtr Handle

{

get => \_handle;

set

{

if (\_handle != IntPtr.Zero)

{

CloseHandle(\_handle);

}

\_handle = value;

}

}

public OsHandle(IntPtr handle)

{

\_handle = handle;

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

~OsHandle()

{

Dispose(false);

}

private void Dispose(bool disposing)

{

if (!\_disposed)

{

if (\_handle != IntPtr.Zero)

{

CloseHandle(\_handle);

\_handle = IntPtr.Zero;

}

\_disposed = true;

}

}

[DllImport("kernel32.dll", SetLastError = true)]

private static extern bool CloseHandle(IntPtr hObject);

}

Program.cs

using System.Runtime.InteropServices;

namespace lapa2.\_2;

class Program

{

[DllImport("kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]

private static extern IntPtr CreateFile(

string lpFileName,

uint dwDesiredAccess,

uint dwShareMode,

IntPtr lpSecurityAttributes,

uint dwCreationDisposition,

uint dwFlagsAndAttributes,

IntPtr hTemplateFile);

[DllImport("kernel32.dll", SetLastError = true)]

private static extern bool WriteFile(

IntPtr hFile, byte[] lpBuffer, uint nNumberOfBytesToWrite,

out uint lpNumberOfBytesWritten, IntPtr lpOverlapped);

private const uint GENERIC\_READ = 0x80000000;

private const uint GENERIC\_WRITE = 0x40000000;

private const uint FILE\_SHARE\_READ = 0x00000001;

private const uint FILE\_SHARE\_WRITE = 0x00000002;

private const uint OPEN\_ALWAYS = 4;

private const uint FILE\_ATTRIBUTE\_NORMAL = 0x80;

static void Main()

{

string filePath = "C:\\BSUIR\\MoPsP\\lapa2.2\\lapa2.2\\file.txt";

File.WriteAllText(filePath, "Hello, world!"); // Создадим файл заранее

IntPtr fileHandle = CreateFile(filePath, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

IntPtr.Zero, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, IntPtr.Zero);

if (fileHandle == IntPtr.Zero || fileHandle == new IntPtr(-1))

{

Console.WriteLine("Ошибка открытия файла.");

return;

}

Console.WriteLine($"Получен дескриптор: {fileHandle}");

using (OsHandle osHandle = new OsHandle(fileHandle))

{

Console.WriteLine($"Освобождение ресурса {osHandle.Handle} через using...");

}

byte[] data = System.Text.Encoding.UTF8.GetBytes("Test data");

if (!WriteFile(fileHandle, data, (uint)data.Length, out uint bytesWritten, IntPtr.Zero))

{

Console.WriteLine($"Ошибка при записи в закрытый дескриптор! Код ошибки: {Marshal.GetLastWin32Error()}");

}

else

{

Console.WriteLine("Неожиданно запись прошла успешно! (Это не должно было случиться)");

}

}

}