Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра «»

Лабораторная работа №5

по дисциплине «Операционные системы»

«Потоки»

Выполнил: студент гр..

Проверил:.

Тамбов, 20

***Цели и задачи****.*

Получение практических навыков по использованию Win32 API для исследования потоков.

Задание: Разработать программу, реализующую синхронную работу над общим обьектом с визуализацией результатов работы. В программе должно быть предусмотрено сравнение выполнения задания с синхронизацией потоков и без них.

***Решение задачи****.*

Критические секции — это механизм, предназначенный для синхронизации потоков внутри одного процесса.

Перед использованием критической секции необходимо *инициализировать* ее функцией:

**InitializeCriticalSection**(var lpCriticalSection: **RTLCriticalSection** );

После создания объекта поток, перед доступом к защищаемому ресурсу, должен вызвать функцию:

**EnterCriticalSection**( var lpCriticalSection: **TRTLCriticalSection** );

Если в этот момент ни один из потоков в процессе не владеет объектом, то поток становится *владельцем* критической секции и продолжает выполнение. Если секция уже захвачена другим потоком, то выполнение потока, вызвавшего функцию, приостанавливается до ее освобождения.

Поток, владеющий критической секцией, может повторно вызывать функцию **EnterCriticalSection** без блокирования своего исполнения. По *завершении* работы с защищаемым ресурсом поток должен вызвать функцию LeaveCriticalSection:

**LeaveCriticalSection**( var lpCriticalSection: **TRTLCriticalSection** );

Эта функция *освобождает* объект независимо от количества предыдущих вызовов потоком функции EnterCriticalSection. Если имеются другие потоки, ожидающие освобождения секции, один из них становится ее владельцем и продолжает исполнение.

По *завершении* работы с критической секцией она должна быть *уничтожена* вызовом функции DeleteCriticalSection:

**DeleteCriticalSection**(var lpCriticalSection: **TRTLCriticalSection** );

class TickTok

{

/\*Код, помещенный между ::EnterCriticalSection() и ::LeaveCriticalSection()

\* с одной и той же критической секцией в качестве параметра,

\* никогда не будет выполняться параллельно.\*/

[StructLayout(LayoutKind.Sequential)]

public struct CRITICAL\_SECTION

{

public IntPtr DebugInfo; // Используется операционной системой

public int LockCount; // Счетчик использования этой критической секции

public int RecursionCount; // Счетчик повторного захвата из нити-владельца

public IntPtr OwningThread; // Уникальный ID нити-владельца

public IntPtr LockSemaphore; // Объект ядра используемый для ожидания

public UIntPtr SpinCount; // Количество холостых циклов перед вызовом ядра

}

// INIT CRITICAL SECTION

[DllImport("kernel32.dll")]//Заполняют поля структуры, адресуемой lpCriticalSection. После вызова критическая секция готова к работе.

public static extern bool InitializeCriticalSection(ref CRITICAL\_SECTION lpCriticalSection);

// DELETE CRITICAL SECTION

[DllImport("kernel32.dll")]//Освобождает ресурсы, занимаемые критической секцией.

public static extern void DeleteCriticalSection(ref CRITICAL\_SECTION lpCriticalSection);

// ENTER CRITICAL SECTION

[DllImport("kernel32.dll")]//Осуществляют "захват" критической секции.

public static extern void EnterCriticalSection(ref CRITICAL\_SECTION lpCriticalSection);

// LEAVE CRITICAL SECTION

[DllImport("kernel32.dll")]//Освобождает критическую секцию

public static extern void LeaveCriticalSection(ref CRITICAL\_SECTION lpCriticalSection);

public static List<string> Ticker = new List<string>(50);

CRITICAL\_SECTION lockOn = new CRITICAL\_SECTION();

public void Tick(bool running)

{

try

{

EnterCriticalSection(lockOn);

if(!running)

{//остановить часы

if(!Form1.State) Monitor.Pulse(lockOn);//Уведомить любые ожидающие потоки

return;

}

Ticker.Add("ТИК");

if (!Form1.State)

{

Monitor.Pulse(lockOn);//Разрешить выполнение метода Tock()

Monitor.Wait(lockOn);//Ожидать завершения метода Tock()

}

}finally

{

LeaveCriticalSection(lockOn);

}

}

public void Tock(bool running)

{

try

{

EnterCriticalSection(lockOn);

if (!running)

{//остановить часы

if (!Form1.State) Monitor.Pulse(lockOn);//Уведомить любые ожидающие потоки

return;

}

Ticker.Add("ТАК");

if (!Form1.State)

{

Monitor.Pulse(lockOn);//Разрешить выполнение метода Tock()

Monitor.Wait(lockOn);//Ожидать завершения метода Tock()

}

}finally

{

LeaveCriticalSection(lockOn);

}

}

}

internal class MyThread

{

public Thread Thrd;

private TickTok ttOb;

//сконструировать новый поток

public MyThread(string name, TickTok tt)

{

InitializeCriticalSection(tt);

Thrd=new Thread(this.Run);

ttOb = tt;

Thrd.Name = name;

Thrd.Start();

DeleteCriticalSection(tt);

}

void Run()//начать выполнение нового потока

{

if(Thrd.Name=="Tick")

{

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

{

ttOb.Tick(true);

}

ttOb.Tick(false);

}

else//"Tock"

{

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

{

ttOb.Tock(true);

}

ttOb.Tock(false);

}

}

}

И вызов с формы:

if (er.Index == 0)

{

/\*Когда выполнение потока временно заблокировано, он он вызывает метод Wait()

\* В итоге поток переходит в сост ожидания, а блокировка с ресурса снимается

\* что даст возможность использовать ресурс в другом потоке \*/

TickTok.Ticker.Clear();

//создать обьект общий ресурс

TickTok tt = new TickTok();

//создать два патока с вызывом общего ресурса

MyThread mt1 = new MyThread("Tick", tt);

MyThread mt2 = new MyThread("Tock", tt);

mt1.Thrd.Join();//ждать завершения потока 1

mt2.Thrd.Join();//ждать завершения потока 2

for (int i = 0; i < TickTok.Ticker.Count; i++)

{

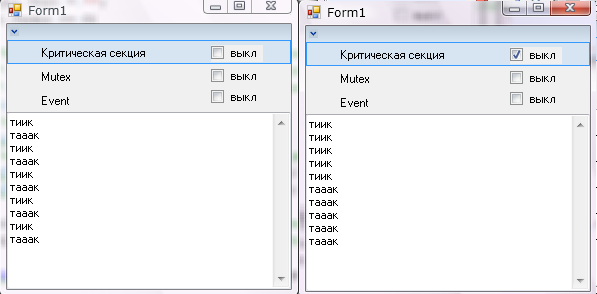
if (TickTok.Ticker[i] == "ТИК") textBox1.Text += "тиик";

else textBox1.Text += "тааак";

textBox1.Text += Environment.NewLine;

}

}

Результат:

Mutex (Mutually Exclusive)

Мьютекс — это объект синхронизации, который находится в сигнальном состоянии только тогда, когда не принадлежит ни одному из процессов. Как только хотя бы один процесс запрашивает владение мьютексом, он переходит в несигнальное состояние и остается таким до тех пор, пока не будет освобожден владельцем. Такое поведение позволяет использовать мьютексы для синхронизации совместного доступа нескольких процессов к разделяемому ресурсу. Для создания мьютекса используется функция:

function **CreateMutex**(

lpMutexAttributes: PSecurityAttributes; // Адрес структуры

// TSecurityAttributes

bInitialOwner: BOOL; // Задает, будет ли процесс владеть

// мьютексом сразу после создания

lpName: PChar // Имя мьютекса

): **THandle**

Функция возвращает идентификатор созданного объекта либо 0.

Для возврата в *несигнальное* состояние служит функция ReleaseMutex:

function **ReleaseMutex**(hMutex: **THandle**): BOOL

/// <summary>

/// Общий ресурс - переменая Count, а также мютекс - управляющий доступом к ней

/// </summary>

internal class SharedRes

{

public static int Count = 0;

public static Handle Mtx = CreateMutex(0, true, Count.ToString());

}

/// <summary>

/// В этом потоке переменая Shared.Count инкрементируется

/// </summary>

internal class IncThread

{

private int num;

public Thread Thrd;

public IncThread (string name,int n)

{

Thrd=new Thread(this.Run);

num = n;

Thrd.Name = name;

Thrd.Start();

}

/// <summary>

/// Точка входа в поток

/// </summary>

void Run()

{

Form1.console.Add(Thrd.Name+" ожидает мьютекс");

//получить мютекс

if (!Form1.State) SharedRes.Mtx.WaitOne();

Form1.console.Add(Thrd.Name + " получает мьютекс");

do

{

Thread.Sleep(500);

SharedRes.Count++;

Form1.console.Add("В потоке "+Thrd.Name + ", SharedRes.Count="+SharedRes.Count);

num--;

} while (num>0);

Form1.console.Add(Thrd.Name + " освобождает мьютекс");

//освободить мютекс

if (!Form1.State) ReleaseMutex(Mtx);

}

}

/// <summary>

/// В этом потоке переменная SharedRes.Count дектементируется

/// </summary>

internal class DecThread

{

private int num;

public Thread Thrd;

public DecThread(string name,int n)

{

Thrd=new Thread(new ThreadStart(this.Run));

num = n;

Thrd.Name = name;

Thrd.Start();

}

//точка входа в поток

void Run()

{

Form1.console.Add(Thrd.Name+" ожидает мютекс");

//получить мютекс

if (!Form1.State) SharedRes.Mtx.WaitOne();

Form1.console.Add(Thrd.Name + " получает мютекс");

do

{

Thread.Sleep(500);

SharedRes.Count--;

Form1.console.Add("В потоке " +Thrd.Name + ", SharedRes.Count="+SharedRes.Count);

num--;

} while (num>0);

Form1.console.Add(Thrd.Name + " освобождает мютекс");

//освободить мютекс

if (!Form1.State) ReleaseMutex(Mtx);

}

}

Вызов с формы:

if (er.Index == 1)

{

/\*Mutex представляет собой взаимно исключающий синхронизирующий обьект, т.е

он может быть получен потоком только по очереди \*/

console.Clear();

//сконструировать два потока

IncThread mt1=new IncThread("Инкриментирующий поток", 5);

Thread.Sleep(1);//разрешить начать инкрим поток

DecThread mt2=new DecThread("Декрементирующий поток", 5);

mt1.Thrd.Join();//ждать завершения потока 1

mt2.Thrd.Join();//ждать завершения потока 2

for (int i = 0; i < console.Count; i++)

{

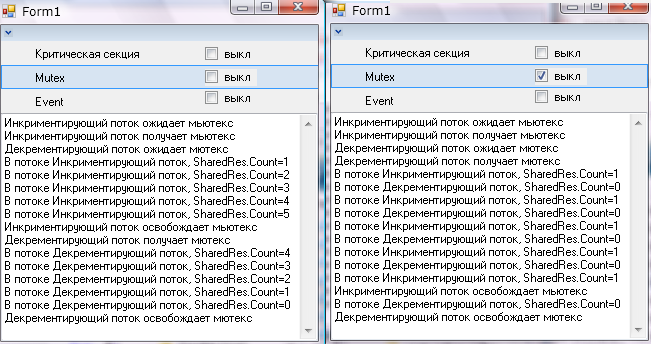
textBox1.Text += console[i];

textBox1.Text += Environment.NewLine;

}

}

Отработка в программе:



Event (событие)

Event позволяет известить один или несколько ожидающих потоков о наступлении события.

Для создания объекта используется функция CreateEvent:

function **CreateEvent**(

lpEventAttributes: PSecurityAttributes; // Адрес структуры

// TSecurityAttributes

bManualReset, // Задает, будет Event переключаемым

// вручную (TRUE) или автоматически (FALSE)

bInitialState: BOOL; // Задает начальное состояние. Если TRUE -

// объект в сигнальном состоянии

lpName: PChar // Имя или NIL, если имя не требуется

): THandle; // Возвращает идентификатор созданного объекта

Структура **TSecurityAttributes** описана, как:

TSecurityAttributes = record

nLength: DWORD; // Размер структуры, должен

// инициализироваться как SizeOf(TSecurityAttributes)

lpSecurityDescriptor: Pointer; // Адрес дескриптора защиты. В

// Windows 95 и 98 игнорируется. Обычно можно указывать NIL

bInheritHandle: BOOL; // Задает, могут ли дочерние

// процессы наследовать объект

end;

После получения идентификатора можно приступать к его использованию. Для этого имеются следующие функции:

function **SetEvent**(hEvent: THandle): BOOL; stdcall;

— устанавливает объект в сигнальное состояние

function **ResetEvent**(hEvent: THandle): BOOL; stdcall;

— сбрасывает объект, устанавливая его в несигнальное состояние

function **PulseEvent**(hEvent: THandle): BOOL; stdcall

— устанавливает объект в сигнальное состояние, дает отработать всем функциям ожидания, ожидающим этот объект, а затем снова сбрасывает его.

/// <summary>

/// Этот поток уведомляет о том, что событие передано его конструктору

/// </summary>

internal class MyThreadEvent

{

public Thread Thrd;

Handle mre = CreateEvent(null, false, true, Thrd.Name);

public static List<string > text=new List<string>(50);

public MyThreadEvent(string name,Handle evt)

{

Thrd=new Thread(this.Run);

Thrd.Name = name;

mre = evt;

Thrd.Start();

}

/// <summary>

/// точка входа в поток

/// </summary>

void Run()

{

text.Add("Внутри потока "+Thrd.Name);

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

{

text.Add(Thrd.Name);

Thread.Sleep(500);

}

text.Add(Thrd.Name+" завершен!");

//уведомить о событии

SetEvent(mre);

}

}

Вызов с формы:

if (er.Index == 2)

{

MyThreadEvent.text.Clear();

Handle evtObj= CreateEvent(null, false, false, Thrd.Name); MyThreadEvent mt1=new MyThreadEvent("Событийный поток 1", evtObj);

textBox1.Text += "Основной поток ожидает события" + Environment.NewLine;

for (int i = 0; i < MyThreadEvent.text.Count; i++)

{

textBox1.Text += MyThreadEvent.text[i];

textBox1.Text += Environment.NewLine;

}

//ожидать уведомления о событии

if(!checkBox3.Checked)PulseEvent(evtObj);

textBox1.Text += "Основной поток получил уведомление о событии от первого потока" + Environment.NewLine;

//Установить событийный обьект в исходное состояние

ResetEvent(evtObj);

mt1=new MyThreadEvent("Событийный поток 2", evtObj);

//Ожидать уведомления о событии

if (!checkBox3.Checked) PulseEvent(evtObj);

for (int i = 0; i < MyThreadEvent.text.Count; i++)

{

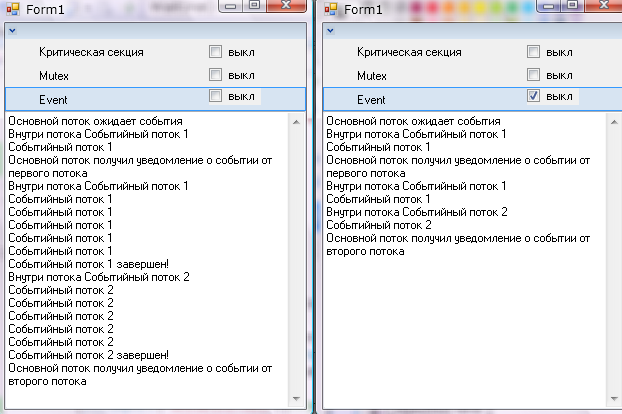
textBox1.Text += MyThreadEvent.text[i];

textBox1.Text += Environment.NewLine;

}

textBox1.Text += "Основной поток получил уведомление о событии от второго потока" + Environment.NewLine;

}

В результате: