Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

СОЗДАНИЕ ПОТОКОВ, ПРОГРАММИРОВАНИЕ ПРИОРИТЕТОВ

Лабораторная работа № 1
по курсу «Операционные системы»

Вариант № 3

Выполнил: студент группы 220601 _____ Белым А.А.
(подпись)
Проверил: _____ Попов А.И.
(подпись)

Тула 2012

Цель работы

Целью работы является знакомство со средствами обеспечения многопоточности в системе Windows.

Задание

Напишите программу, которая завершает свои потоки, не дождавшись окончания их работы. На экране должна отображаться работа потоков, пользователь должен иметь возможность завершить любой из потоков в любой момент времени.

Теоретическая справка

Многопоточность – это способность операционной системы выполнять несколько программ одновременно. Для каждого из процессов выделяется процессорное время, в течение которого центральный процессор работает только с этим потоком. Это время небольшое, поэтому у пользователя создается иллюзия того, что процессы работают параллельно и постоянно. Чтобы все эти процессы работали, операционная система отводит каждому из них определенное процессорное время. Выделяя процессам отрезки времени (называемые квантами) по принципу карусели, она создает тем самым иллюзию одновременного выполнения процессов.

Многопоточность – это возможность программы быть поделенной на отдельные потоки выполнения.

В каждом процессе есть, по крайней мере, один поток. Поток никогда не выходит за границы своего процесса. Поток использует по своему назначению временные ресурсы, а также ресурсы памяти и центрального процессора, то только в рамках своего процесса.

В любой программе поток – это не более чем функция, создаваемая в ней и обладающая возможностью вызывать другие функции программы. Каждый поток обладает своим собственным стеком в ОП в рамках ОП программы, все локальные переменные потока – индивидуальны для каждого потока и недоступны для других потоков. При переключении между потоками

сохраняются и восстанавливаются состояния процессора. Следует заметить, что Windows поддерживает так называемую локальную память потока (thread local storage, TLS). Иногда бывает удобно использовать для двух и более потоков одну и ту же функцию, а статические данные использовать уникальные для каждого потока. Это и есть пример использования локальной памяти потока.

Любой поток состоит из двух компонентов:

• объекта ядра, через который операционная система управляет потоком. Там же хранится статистическая информация о потоке;

• стека потока, который содержит параметры всех функций и локальные переменные, необходимые потоку для выполнения кода

Поток (thread) определяет последовательность исполнения кода в процессе. При инициализации процесса система всегда создает первичный поток. Начинаясь со стартового кода из библиотеки дэльфи, который в свою очередь вызывает входную функцию из Вашей программы, он живет до того момента, когда входная функция возвращает управление стартовому коду и тот вызывает функцию ExitProcess. Большинство приложений обходится единственным, первичным потоком. Однако процессы могут создавать дополнительные потоки, что позволяет им эффективнее выполнять свою работу. Если аппаратное обеспечение компьютера включает в себя два и более процессоров, то каждый поток можно закрепить за отдельным процессором, и они все будут в работе.

Каждый поток начинает выполнение с некоей входной функции. Функция потока может выполнять любые задачи. Рано или поздно она закончит свою работу и вернет управление. В этот момент Ваш поток остановится, память, отведенная под его стек, будет освобождена, а счетчик пользователей его объекта ядра "поток" уменьшится на 1. Когда счетчик обнулится, этот объект ядра будет разрушен. Но, как и объект ядра "процесс", он может жить гораздо дольше, чем сопоставленный с ним поток. Рассмотрим создание потока. При вызове CreateProcess появляется первичный поток процесса. Если есть необходимость создать дополнительные потоки, это делается функцией CreateTheread:

HANDlF CreateThread(

PSECURITY_ATTRIBUTES psa, DWORD cbStack,

PTHREAD_START_ROUTINE pfnStartAddr, PVOID pvParam, DWORD tdwCreate, PDWORD pdwThreadID);

CreateThread - это Windows-функция, создающая поток.

Система выделяет память под стек потока из адресного пространства процесса. Новый поток выполняется в контексте того же процесса, что и родительский поток. Поэтому он получает доступ ко всем описателям объектов ядра, всей памяти и стекам всех потоков в процессе. За счет этого потоки в рамках одного процесса могут легко взаимодействовать друг с другом.

Поток можно завершить четырьмя способами:

• функция потока возвращает управление

• поток самоуничтожается вызовом функции ExitThread

• один из потоков данного или стороннего процесса вызывает функцию TerminateThread

• завершается процесс, содержащий данный поток.

Рассмотрим каждый из способов более подробно.

1. Функцию потока следует проектировать так, чтобы поток завершался только после того, как она возвращает управление. Это единственный способ, гарантирующий корректную очистку всех ресурсов, принадлежавших потоку. При этом:

• любые C++-объекты, созданные данным потоком, уничтожаются соответствующими деструкторами;

• система корректно освобождает память, которую занимал стек потока;

• система устанавливает код завершения данного потока (поддерживаемый объектом ядра "поток») — его и возвращает функция потока;

• счетчик пользователей данного объекта ядра "поток" уменьшается на 1

2. Поток можно завершить принудительно, вызвав:

VOID ExitThread(DWORD dwExitCode);

ExitThread — это Windows-функция, которая уничтожает поток. При этом освобождаются все ресурсы операционной системы, выделенные дан ному потоку, но C/C++ - ресурсы (например, объекты, созданные из C++-классов) не очищаются. Именно поэтому лучше возвращать управление из функции потока,

чем самому вызывать функцию ExitThread. В параметр dwExitCode помещается значение, которое система рассматривает как код завершения потока. Возвращаемого значения у этой функции нет, потому что после ее вызова поток перестает существовать.

3. Вызов этой функции также завершает поток:

BOOL TerminateThread(HANDLE hThread, DWORD dwExitCode);

В отличие от ExitThread, которая уничтожает только вызывающий поток, эта функция завершает поток, указанный в параметре hThread. В параметр dwExitCode помещается значение, которое система рассматривает как код завершения потока. После того как поток будет уничтожен, счетчик пользователей его объекта ядра «поток» уменьшится на 1. TerminateThread — функция асинхронная, т.е. она сообщает системе, что надо завершить поток, но к тому времени, когда она вернет управление, поток может быть еще не уничтожен. Так что, если нужно точно знать момент завершения потока, надо использовать WaitForSingleObject или аналогичную функцию, передав ей описатель этого потока. Корректно написанное приложение не должно вызывать эту функцию, поскольку поток не получает никакого уведомления о завершении; из-за этого он не может вы полнить должную очистку ресурсов. Уничтожение потока при вызове ExitThread или возврате управления из функции потока приводит к разрушению его стека. Но если он завершен функцией TerminateThread, система не уничтожает стек, пока не завершится и процесс, которому принадлежал этот поток. Так сделано потому, что другие потоки могут использовать указатели, ссылающиеся на данные в стеке завершенного потока. Если бы они обратились к несуществующему стеку, произошло бы на рушение доступа.

4. Функции ExitProcess и TerminateProcess, рассмотренные в главе 4, тоже завершают потоки. Единственное отличие в том, что они прекращают выполнение всех потоков, принадлежавших завершенному процессу. При этом гарантируется высвобождение любых выделенных процессу ресурсов, в том числе стеков потоков. Однако эти две функции уничтожают потоки принудительно — так, будто для каждого из них вызывается функция TerminateThread. А это означает,

что очистка проводится некорректно, деструкторы C++-объектов не вызываются, данные на диск не сбрасываются и т.д.

Инструкция пользователю

Данная программа позволяет продемонстрировать функции Win32API для работы с потоками.

Программа может создавать и завершать потоки. Для создания потока нажмите соответствующую кнопку. После запуска потока его дескриптор появляется в списке в левой части окна программы. Во время работы поток с некоторым интервалом печатает свой дескриптор в текстовое поле в окне программы. Чтобы изменить этот интервал, можно ввести новое значение интервала в миллисекундах в специальное поле, и нажать кнопку «Применить». Чтобы завершить поток, выберите его дескриптор из списка, и нажмите соответствующую кнопку.

Инструкция программисту

Данная программа состоит из двух частей:

1. Часть, непосредственно работающая с потоковыми функциями Win32API: Threads.{h,cpp}.
2. Часть, реализующая графический интерфейс пользователя с помощью библиотеки MFC: файлы Lab1.{h,cpp} , Lab1Dlg.{h,cpp}.

В первой части реализуются функция-поток и вспомогательные функции для запуска и завершения потоков, а также некоторые необходимые переменные.

Макрос:

```
#define WM_UPDATELOG (WM_APP+1)
```

Сообщение Windows, которое поток посылает окну программы.

Тип:

```
typedef CList<HANDLE> THREADLIST;
```

Реализует список потоков.

Переменные:

```
HWND win=NULL;
```

Окно, принимающее сообщения от потоков.

```
DWORD sleep_t=0;
```

Интервал в миллисекундах, на который потоки засыпают с помощью функции Sleep() после отправки сообщения.

Функции:

```
DWORD WINAPI ThreadFunc(VOID* PARAMS);
```

Функция-поток. Принимает указатель на свой дескриптор(типа HANDLE); в бесконечном цикле (`while (true)`) отсылает сообщение `WM_UPDATELOG` окну `win`, после чего засыпает на время `sleep_t`.

```
bool SpawnThread(THREADLIST& threads);
```

Запускает поток функцией `CreateThread`, и добавляет его дескриптор в список `threads`.

```
bool KillThread(THREADLIST& threads,INT_PTR num);
```

Завершает поток номер `num` из списка `threads` функцией `TerminateThread`, и удаляет его из списка.

Текст программы

Ниже представлен текст программы, демонстрирующей работу с потоками и написанной на языке C++, в среде Visual Studio 2012 с использованием библиотеки MFC.

В файлах Threads.h и Threads.cpp представлены функция-поток и вспомогательные функции для запуска и завершения потоков.

Threads.h:

```
#ifndef LAB1_THREADS_H
#define LAB1_THREADS_H
#include <afxtempl.h>

#define WM_UPDATELOG (WM_APP+1)

typedef CList<HANDLE> THREADLIST;

extern HWND win;
extern DWORD sleep_t;

DWORD WINAPI ThreadFunc(VOID* PARAMS);
bool SpawnThread(THREADLIST& threads);
bool KillThread(THREADLIST& threads,INT_PTR num);
```

```
#endif //LAB1_THREADS_H
```

Threads.cpp:

```
#include "stdafx.h"
#include "Threads.h"
HWND win=NULL;
DWORD sleep_t=0;
DWORD WINAPI ThreadFunc(VOID* PARAMS){
	HANDLE my=*(HANDLE*)PARAMS;

	while (true){
		SendMessage(win,WM_UPDATELOG,WPARAM(my),0);
		Sleep(sleep_t);
	}
	return 0;
}

bool SpawnThread(THREADLIST& threads){
	threads.AddTail(HANDLE());
	HANDLE& new_handle=threads.GetTail();
	if((new_handle=CreateThread(NULL,0,&ThreadFunc,(void*)&new_handle,0,NULL))==
NULL){
		CString msg("Ошибка запуска потока!");
		AfxMessageBox(msg);
		threads.RemoveTail();
		return false;
	} ;
	return true;
}

bool KillThread(THREADLIST& threads,INT_PTR num){
	POSITION pos=threads.FindIndex(num);
	HANDLE del_handle=threads.GetAt(pos);
	if (TerminateThread(del_handle,1)){
		threads.RemoveAt(pos);
		return true;
	} else return false;
}
```

В файлах Lab1Dlg.h и Lab1Dlg.cpp представлена часть программы, реализующая основную форму программы.

Lab1Dlg.h:

```
// Lab1Dlg.h : файл заголовка
//

#pragma once
#include "afxwin.h"
#include "Threads.h"

// диалоговое окно CLab1Dlg
class CLab1Dlg : public CDialogEx
{
// Создание
public:
	CLab1Dlg(CWnd* pParent = NULL);	// стандартный конструктор

// Данные диалогового окна
	enum { IDD = IDD_LAB1_DIALOG };
```

```
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);     // поддержка DDX/DDV
	virtual void CLab1Dlg::OnOK();

// Реализация
protected:
	HICON m_hIcon;
	THREADLIST mythreads;
	// Созданные функции схемы сообщений
	virtual BOOL OnInitDialog();
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnBnClickedOk();
	afx_msg LRESULT OnUpdateLog(WPARAM wParam, LPARAM lParam);
	afx_msg void OnBnClickedButton2();
	CListBox m_lstThreads;
	afx_msg void OnBnClickedButton3();
	CString m_strLog;
	CEdit m_ctlLog;
	DWORD m_SleepTime;
	afx_msg void OnBnClickedButton1();
};
```

Lab1Dlg.cpp:

```
// Lab1Dlg.cpp : файл реализации
//

#include "stdafx.h"
#include "Lab1.h"
#include "Lab1Dlg.h"
#include "afxdialogex.h"
#include <afxtempl.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// диалоговое окно CLab1Dlg


void CLab1Dlg::OnOK(){
}


CLab1Dlg::CLab1Dlg(CWnd* pParent /*=NULL*/)
	: CDialogEx(CLab1Dlg::IDD, pParent)
	, m_strLog(_T(""))
	, m_SleepTime(0)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CLab1Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_LIST1, m_lstThreads);
	DDX_Text(pDX, IDC_EDIT2, m_strLog);
	DDX_Control(pDX, IDC_EDIT2, m_ctlLog);
	DDX_Text(pDX, IDC_EDIT1, m_SleepTime);
}
```

```
BEGIN_MESSAGE_MAP(CLab1Dlg, CDialogEx)
	ON_MESSAGE(WM_UPDATELOG, &CLab1Dlg::OnUpdateLog)
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDOK, &CLab1Dlg::OnBnClickedOk)
	ON_BN_CLICKED(IDC_BUTTON2, &CLab1Dlg::OnBnClickedButton2)

//	ON_LBN_SELCHANGE(IDC_LIST1, &CLab1Dlg::OnLbnSelchangeList1)
ON_BN_CLICKED(IDC_BUTTON3, &CLab1Dlg::OnBnClickedButton3)
ON_BN_CLICKED(IDC_BUTTON1, &CLab1Dlg::OnBnClickedButton1)
END_MESSAGE_MAP()


// обработчики сообщений CLab1Dlg

BOOL CLab1Dlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// Задает значок для этого диалогового окна. Среда делает это автоматически,
	//  если главное окно приложения не является диалоговым
	SetIcon(m_hIcon, TRUE);			// Крупный значок
	SetIcon(m_hIcon, FALSE);		// Мелкий значок

	// TODO: добавьте дополнительную инициализацию
	win=GetSafeHwnd();
	sleep_t=1000;
	UpdateData();
	m_SleepTime=sleep_t;
	UpdateData(FALSE);
	return TRUE;  // возврат значения TRUE, если фокус не передан элементу
управления
}

// При добавлении кнопки свертывания в диалоговое окно нужно воспользоваться при-
веденным ниже кодом,
//  чтобы нарисовать значок. Для приложений MFC, использующих модель документов
или представлений,
//  это автоматически выполняется рабочей областью.

void CLab1Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // контекст устройства для рисования

		SendMessage(WM_ICONERASEBKGND, reinter-
pret_cast<WPARAM>(dc.GetSafeHdc()), 0);

		// Выравнивание значка по центру клиентского прямоугольника
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Нарисуйте значок
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialogEx::OnPaint();
	}
}
```

```
// Система вызывает эту функцию для получения отображения курсора при перемещении
//  свернутого окна.
HCURSOR CLab1Dlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}



void CLab1Dlg::OnBnClickedOk()
{
	// TODO: добавьте свой код обработчика уведомлений
	CDialogEx::OnOK();
}


void CLab1Dlg::OnBnClickedButton2()
{
	// TODO: добавьте свой код обработчика уведомлений
	if(SpawnThread(mythreads)){
		HANDLE new_handle=mythreads.GetTail();
		CString ThreadName;
		ThreadName.Format(_T("Поток %p"),new_handle);
		m_lstThreads.AddString(ThreadName);
	}
   // Handle message here.
}



void CLab1Dlg::OnBnClickedButton3()
{
	// TODO: добавьте свой код обработчика уведомлений
	int pos=m_lstThreads.GetCurSel();
	if (pos>=0){
		if (KillThread(mythreads,pos)){
			m_lstThreads.DeleteString(pos);
		} ;
	};
}
afx_msg LRESULT CLab1Dlg::OnUpdateLog(WPARAM wParam, LPARAM lParam){
   UNREFERENCED_PARAMETER(wParam);
   UNREFERENCED_PARAMETER(lParam);
   CString msg; msg.Format(_T("Я поток %p\r\n"),wParam);
   UpdateData();
   m_strLog+=msg;
   UpdateData(FALSE);
   m_ctlLog.LineScroll(m_ctlLog.GetLineCount()-1);
   // Handle message here.

   return 0;
}


void CLab1Dlg::OnBnClickedButton1()
{
	// TODO: добавьте свой код обработчика уведомлений
	UpdateData();
	sleep_t=m_SleepTime;
	UpdateData(FALSE);
}
```

Тестовый пример

На рисунке 1 представлены результаты работы программы, демонстрирующей функции Win32API для работы с потоками.

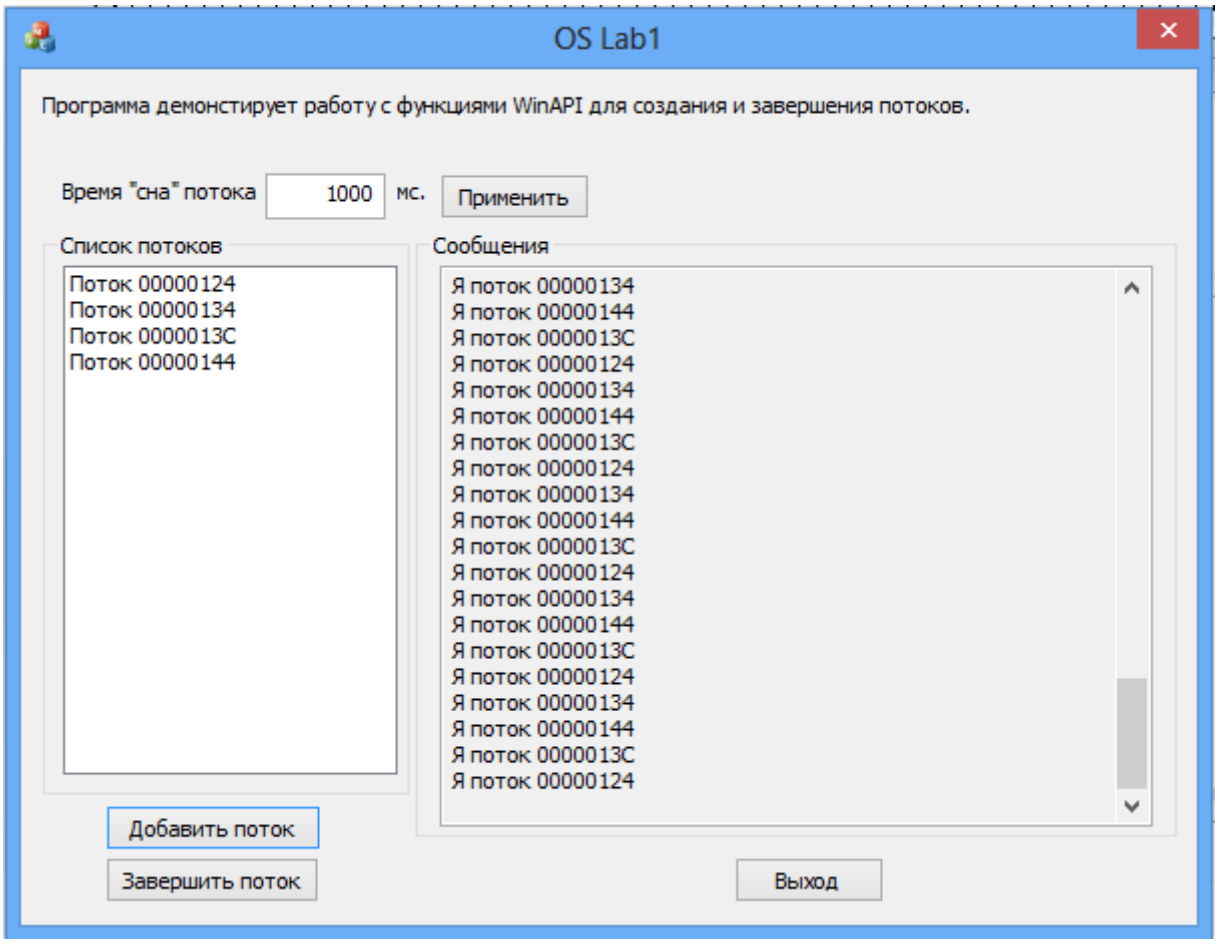

Рисунок 1— Пример работы программы анализа

Вывод

В ходе выполнения данной лабораторной работы я научился использовать функции Win32API для работы с потоками. Потоки позволяют программе выполнять несколько функций квазиодновременно.

Новый поток создается с помощью функции CreateThread. Завершить поток можно разными способами, например, функцией TerminateThread; но лучше просто позволить завершиться функции потока с помощью ключевого слова return.