Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информатика»

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ПРАКТИКУМ
по выполнению лабораторных работ
по одноименной дисциплине
для студентов специальности 1-40 04 01
«Информатика и технологии программирования»
дневной формы обучения

Гомель 2021

УДК 004.921(075.8)
ББК 32.973.1я73
С40

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 7 от 02.03.2020 г.*)

Составитель *Г. П. Косинов*

Рецензент: проф. каф. «Информационные технологии» ГГТУ им. П. О. Сухого»
д-р. техн. наук *И. А. Мурашко*

С40 **Системное** программирование : практикум по выполнению лаборатор. работ по одноим. дисциплине для студентов специальности 1-40 04 01 «Информатика и технологии программирования» днев. формы обучения / сост. Г. П. Косинов. – Гомель : ГГТУ им. П. О. Сухого, 2021. – 55 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

Рассмотрены вопросы взаимодействия между процессами, создания диспетчера задач, управления различными видами памяти, DLL-библиотеки, создание процессов и потоков, а также их синхронизация при доступе к общим ресурсам.

Для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения.

УДК 004.921(075.8)
ББК 32.973.1я73

Содержание

Лабораторная работа № 1

Тема работы: управление Windows-приложениями

Цель работы: изучить основы создания WinAPI приложения, научиться организовывать управление вспомогательным приложением из основного.

Теоретические сведения по изучению темы

Интерфейс прикладной программы (API) – это набор функций, совместно вызываемых из одной программы (или из связанного набора программ), который программист использует при создании приложений.

Win32 API в своем ядре имеет три главных компонента, обеспечивающих его функциональность:

– компонент USER32.DLL отвечает за управление окном (включая сообщения), работу с курсорами, коммуникации, таймеры и множество других функций, не имеющих отношения к отображению окон, но предназначенных управлять окном;

– компонент GDI32.DLL – это интерфейс графических устройств; он отвечает за рисование элементов пользовательского интерфейса и графики, включая файлы Windows, растровые изображения, контексты устройств и шрифты;

– компонент KERNEL32.DLL контролирует все низкоуровневые функции управления памятью, распределения задач и ресурсов, что является сердцевиной Windows.

При программировании для Windows осуществляется работа с объектом, называемым «окном».

Приложение взаимодействует с GDI при помощи контекста устройства. Контекст устройства (devicecontext) – это структура данных, определяющая набор графических объектов, а также связанных с ними атрибутов и графических режимов, которые влияют на вывод. Множество атрибутов, которые содержит контекст устройства, определяют поведение функций GDI.

Для того чтобы рисовать на каком-либо устройстве графического вывода(принтере или экране дисплея), нужно получить дескриптор контекста устройства. Windows предоставляет право на использование устройства, возвращая дескриптор после вызова соответствую-

щих функций. Далее, для того, чтобы идентифицировать устройство, на котором будет выполняться рисование, дескриптор контекста устройства передается в качестве параметра в функции GDI.

Для освобождения и создания контекста применяются функции: ***EndPaint*** и ***BeginPaint*** или ***GetDC*** и ***ReleaseDC***.

В первых двух функциях в качестве параметров передаются: дескриптор окна и адрес переменной типа структуры PAINTSTRUCT, определяемой в процедуре. Функция ***BeginPaint*** заполняет поля этой структуры и возвращает дескриптор контекста окна, который запоминается в переменной типа HDC. Функция ***EndPaint*** же освобождает дескриптор контекста окна и используется при обработке сообщения WM_PAINT.

В случае же когда контекст получается через функцию ***GetDC***, перед выходом после завершения рисования необходимо освободить контекст при помощи функции ***ReleaseDC***. Эта функция используется при обработке других сообщений.

Методы GDI для вывода текста и векторной графики

Поддерживается множество инструментов для рисования: палитры, растровые изображения, шрифты, перья, кисти.

Перья используются для рисования линий и кривых.Перо создается функцией **CreatePen**. Первый параметр данной функции указывает на стиль линий, второй параметр отвечает за толщину линии, а третий на ее цвет. Данная функция возвращает дескриптор пера.

Кисть создается функцией **CreateSolidBrus**h и отвечает за заполнение цветом графического окна. Данная функция возвращает дескриптор кисти.

Для создания отрезка нужно для начала установить начальную точку в нужные координаты с помощью функции ***MoveToEx***, а затем использовать функцию ***LineTo***. В данных функциях параметрами являются дескриптор графического контекста окна, а также координаты X и Y точки: **MoveToEx** (hdc,FirstCoord.x,FirstCoord.y,NULL); **LineTo** (hdc,SecondCoord.x,SecondCoord.y);

Для вывода текста используется функция **TextOut** (hdc,FirstCoord.x,FirstCoord.y, str, wcslen(str)); где первый параметр – дескриптор контекста устройства, второй – x – координата начала, третий – y – координата начала, четвертый – строка LPCTSTR, пятый – число символов в строке (функция wcslen вычисляет эту длину).

Окна и элементы управления

Главным элементом программы в среде Windows является окно. Окно может содержать элементы управления: кнопки, списки, окна редактирования и др. Эти элементы также являются окнами, но обладающими особым свойством: события, происходящие с этими элементами (и самим окном), приводят к приходу сообщений в процедуру окна. Основные элементы управления представлены в табл. 1.1.

Таблица 1.1

Основные объекты управления

Системный класс	Назначение
BUTTON	Кнопка
COMBOBOX	Комбинированное окно
EDIT	Окно редактирования текста
LISTBOX	Окно со списком
SCROLLBAR	Полоса прокрутки
STATIC	Статический элемент(текст)

Создание элементов управления окна осуществляется функцией CreateWindow.

```
    HWND WINAPI CreateWindow(
 _In_opt_ LPCTSTR lpClassName, // имя предопределенного класса
  _In_opt_ LPCTSTR lpWindowName, // текст
  _In_ DWORD dwStyle, // стиль
  _In_ int x, // координата x
  _In_ int y, // координата y
  _In_ int nWidth, // ширина
  _In_ int nHeight, // высота
  _In_opt_ HWND hWndParent, // дескриптор родительского окна
  _In_opt_ HMENU hMenu, // номер пункта меню
  _In_opt_ HINSTANCE hInstance, // дескрипторприложения
  _In_opt_ LPVOID lpParam ); // NULL
```

Рис. 1.1. Функция создания элементов управления

Указанная функция возвращает дескриптор элемента управления окна, который может быть впоследствии использован для анализа

элемента управления, с которым связано обрабатываемое событие. Дескриптор кнопки, например, передается в оконную функцию в качестве параметра **lparam**.

При нажатии кнопки операционная система генерирует сообщение **WM_COMMAND** с параметром lParam, соответствующим дескриптору кнопки.

Пример обработки нажатия кнопки с дескриптором hBtn представлен на рис. 1.2.

```
    LONG WINAPI WndProc(HWND hwnd, UINT Message, WPARAM wparam, LPARAM
lparam) {
  ...
  switch (Message) {
    case WM_COMMAND:
      if(lparam == (LPARAM)hBtn) {
        //обработка нажатия кнопки
      }
      break;
    ...
  }
}
```

Рис. 1.2. Пример обработки нажатия кнопки

Для считывания информации из поля редактирования используется функция **GetWindowText**:

```
    int WINAPI GetWindowText(
  _In_   HWND hWnd,    // дескриптор поля
  _Out_  LPTSTR lpString,   // указатель на текстовую строку
  _In_   int nMaxCount );      // максимальное количество символов
```

Рис. 1.3. Функция считывания текста из поля

Возвращаемое значение – длина считанной текстовой строки. Для установки текста в поле редактирования используется функция, данные:

```
    BOOL WINAPI SetWindowText(
  _In_      HWND hWnd,    //                дескриптор                  поля
  _In_opt_  LPCTSTR lpString ); // указатель на текстовую строку
```

Рис. 1.4. Заполнение поля ввода строкой символов

Для передачи сложных объектов используется структура ***COPYDATASTRUCT***, содержащая данные, которые будут переданы в другую программу. typedefstruct tagCOPYDATASTRUCT {DWORD dwData; DWORD cbData; PVOID lpData;}

В данной структуре dwData – устанавливает до 32 битов данных, которые будут переданы в принимающую программу; cbData – устанавливает размер, в байтах, указанных элементом структуры lpData; lpData – указывает на данные, передающиеся в принимающую программу (может быть NULL). При передаче данной структуры в другую программу используется сообщение **WM_COPYDATA**. Для данного сообщения необходимо сформировать следующие параметры: hwnd – идентифицирует окно, передающее данные; pcds – указывает на структуру COPYDATASTRUCT, содержащую данные для передачи, wParam = (WPARAM) (HWND)hwnd; // дескриптор передающего окна, lParam = (LPARAM) (PCOPYDATASTRUCT) pcds; //указатель на структуру с данными.

Определение собственных сообщений

При организации взаимодействия в программе может потребоваться использовать собственные сообщения, не относящиеся ни к одному из «системных» типов. Для этого предусмотрены два диапазона «пользовательских» типов сообщений – от значения **WM_USER** до 0x7FFF и от **WM_APP** до 0xBFFF. Сообщения из этих диапазонов могут использоваться прикладными программами для собственных целей.

В пределах одного приложения (одного локального оконного класса) сообщения диапазона WM_USER не требуют регистрации в системе, и приложение может произвольно выбирать и использовать их. При взаимодействии между приложениями требуется предварительная регистрация сообщения в системе функцией **RegisterWindowMessage**(). В качестве аргумента она принимает текстовую строку, идентифицирующую сообщение (она предполагается известной обоим взаимодействующим программам), и возвращает числовой идентификатор зарегистрированного сообщения.

Для передачи сообщения используется функция ***SendMessage***, которая отправляет заданное сообщение окну или окнам. Функция вызывает оконную процедуру для заданного окна и не возвращает значение до тех пор, пока оконная процедура не обработает сообще-

ние. Синтаксис данной функции: LRESULT **SendMessage**(HWND hWnd, UINT Msg, WPARAM wParam; LPARAM lParam), где hWnd - дескриптор окна, оконная процедура которого примет сообщение, Msg – определяет сообщение, которое будет отправлено, wParam и lParam – определяют дополнительную конкретизирующую сообщение информацию.

Функция ***PostMessage*** помещает (вставляет в очередь) сообщение в очередь сообщений, связанную с потоком, который создал заданное окно и возвращает значение без ожидания потока, который обрабатывает сообщение. Синтаксис данной функции: LRESULT **PostMessage**(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam). Параметры данной функции такие же, как и у функции SendMessage. Если функция завершается успешно, величина возвращаемого значения – не нуль. Если же функция завершается ошибкой, величина возвращаемого значения нулевая.

Функция ***FindWindow*** извлекает дескриптор окна верхнего уровня, имя класса и имя окна которого соответствуют заданным строкам. Эта функция не ищет дочерние окна. Синтаксис данной функции: HWND **FindWindow**(LPCTSTRlpClassName, LPCTSTR lpWindowName). Параметрами данной функции являются: lpClassName – указатель на строку с нулевым символом в конце, которая определяет имя класса; lpWindowName – указывает на строку с нулевым символом в конце, которая определяет имя окна (заголовок окна). Если этот параметр – ПУСТО (NULL), соответствуют именам всех окон.

Эти параметры можно определить следующим образом: открыть Spy++ во вкладке Средства в VisualStudio, в появившемся окне выбрать вкладку Поиск –> Найти окно, перетащить инструмент поиска на нужное окно и получить заголовок и класс данного окна.

Задание

В каждом задании необходимо создать два приложения. Необходимо осуществлять обработку событий в первом (основном) приложении, из него отправлять сообщения во второе (вспомогательное) приложение. Второе приложение должно реагировать на события первого приложения согласно своего варианта. Индивидуальные задания приведены в табл. 1.1.

Таблица 1.1

Индивидуальные задания

Вариант	Задание
1	1 приложение (на основном окне расположить 4 кнопки с указанием направления: направо, налево, вверх, вниз), 2 приложение (на окне нарисовать синий квадрат ; при нажатии кнопки в первом приложении запустить таймер по движению квадрата в соответствующем направлении)
2	1 приложение (на основном окне расположить Радио бутон кнопки с указанием направления: направо, налево, вверх, вниз и кнопка), 2 приложение (при нажатии кнопки в первом приложении запустить таймер по движению окна второго приложения в соответствующем направлении)
3	1 приложение (на основном окне расположить Лист-бокс с названиями фигур (ромб, квадрат, круг, звезда) и кнопка), 2 приложение (в центре окна появляется выбранная в первом приложении фигура)
4	1 приложение (на основном окне расположить Комбо-бокс с названиями цветов и кнопку), 2 приложение (по нажатию кнопки закрашивает свое окно в один из 8-ми цветов, выбранных в комбо-боксе)
5	1 приложение (обработка сообщений о передвижении мыши, щелчков ЛКМ и ПКМ), 2 приложение (перемещение рисунка по направлению движения мыши в первом окне, ЛКМ – начало перемещения, ПКМ – стоп)
6	1 приложение (обработка клавиш перемещения курсора, Home, End), 2 приложение (перемещение рисунка в окне в соответствии с нажатой в первом приложении клавишей)
7	1 приложение (на основном окне расположить Текст-бокс и кнопку), 2 приложение (закрашивает квадрат в центре окна в цвет, введенный в текст-бокс)
8	1 приложение (на основном окне расположить Текст-бокс (ввод времени) и кнопку), 2 приложение (отображает часовую и минутную стрелку для времени, введенного в текст-бокс первого приложения)

Продолжение табл 1.1

Вариант	Задание
9	1 приложение (на основном окне расположить 4 Текст-бокса для направлений вверх,вниз, вправо и влево и кнопку), 2 приложение (увеличивает свое окно в соответствующем направлении на число, введенное в текст-боксах)
10	1 приложение (на основном окне расположить Радио-буттон с названиями элементов управления и кнопку), 2 приложение (Размещает на своем окне выбранный в первом приложении элемент управления)
11	1 приложение (на основном окне расположить 5 кнопок), 2 приложение (каждая из 5 кнопок первого приложения запускает свой таймер во втором приложении по перемещению текст-бокса)
12	1 приложение (на основном окне расположить Чек-бокс и кнопку), 2 приложение (по нажатию кнопки в первом приложении рисует и переворачивает картинку)
13	1 приложение (на основном окне расположить Лист-бокс и кнопку, 4 картинки), 2 приложение (при нажатии кнопки первого приложения необходимо изменять фон второго приложения на соответствующую картинку)
14	1 приложение (на основном окне расположить 2 кнопки), 2 приложение (набор картинок, при нажатии кнопок организовать запуск таймера по смене картинок в прямом или обратном порядке)
15	1 приложение (на основном окне расположить Радио бутон и кнопку), 2 приложение (закрашивает окно в выбранный цвет в Радио-бутон)
16	1 приложение (на основном окне расположить Текст-бокс и кнопку), 2 приложение (на основном окне расположить Статик, при нажатии кнопки в первом приложении запускается таймер, который двигает этот статик под углом, введенным в текст-бокс)
17	1 приложение (на основном окне расположить Текст-бокс и кнопку), 2 приложение (на основном окне расположить Текст-бокс, при нажатии кнопки в первом приложении запускается таймер по перемещению информации из одного текст-бокса в другой(меняет содержимое)

Продолжение табл. 1.1

Вариант	**Задание**
18	1 приложение (обработка щелчков ЛКМ), 2 приложение (рисует линии во втором приложении по координатам щелчка ЛКМ в первом приложении)
19	1 приложение (обработка клавиш перемещения курсора, Home, End), 2 приложение (перемещение самого окна 2 приложения в соответствии с нажатыми клавишами в первом приложении)
20	1 приложение (на основном окне расположить Текст-бокс и 2 кнопки - Добавить и Удалить), 2 приложение (на основном окне расположить Лист-бокс, в который добавляется или удаляется текст, введенный в текст-бокс первого приложения)
21	1 приложение (на основном окне расположить Лист-бокс и кнопку), 2 приложение (при нажатии кнопки строит график функции из лист-бокса первого приложения)
22	1 приложение (на основном окне расположить Лист-бокс и кнопку), 2 приложение (при нажатии на кнопку первого приложения заполняет Лист-бокс второго приложения элементами из Лист-бокса первого приложения)
23	1 приложение (на основном окне расположить 10 кнопок), 2 приложение (создается такая же кнопка, как на первом окне)
24	1 приложение (на основном окне расположить Лист-бокс, Текст-бокс и кнопка Добавить), 2 приложение (при нажатии кнопки Добавить в первом приложении, и в первом и во втором приложении в лист-бокс добавляется , как и в первом приложении)
25	1 приложение (на основном окне расположить Чек-бокс и кнопку), 2 приложение (закрашивает круг или в желтый или в красный цвет в зависимости от состояния чек-бокса первого окна)

Окончание табл. 1.1

Вариант	Задание
26	1 приложение (на основном окне расположить Лист-бокс и кнопку, 4 картинки), 2 приложение (при нажатии кнопки в первом приложении в центре окна размещает выбранную картинку из лист-бокса)
27	1 приложение (на основном окне расположить 4 кнопки с направлениями увеличения размера второго окна), 2 приложение (увеличивает свой размер окна на 50 пикселей при нажатии соответствующей кнопки вправо, влево, вверх или вниз)
28	1 приложение (обработка щелчков мыши ЛКМ , ПКМ, 2ЛКМ), 2 приложение (создается кнопка, текстовое поле или лист-бокс соответственно во 2 приложении)
29	Написать два приложения, первое каждые 10 секунд запускает второе и завершает первое , а второе каждые 10 секунд запускает первое и завершает второе

Контрольные вопросы:

1. Что такое контекст устройства?
2. Какие функции применяются для создания контекста?
3. Какая функция используется для создания элементов управления окна?
4. Как передавать между приложениями сложные объекты?
5. Какие функции используются для передачи сообщения?
6. Для чего предназначено сообщение WM_USER?
7. Как можно передать собственное сообщение другому окну?
8. Как сообщение может быть обработано в другом окне?
9. Как происходит регистрация собственных сообщений?
10. Какие сообщения приходят при нажатии, передвижении и отпускании кнопок «мыши». Какие параметры данных сообщений?
11. Основные свойства элемента управления Button?
12. Основные свойства элемента управления ListBox?
13. Как можно создать RadioButton и CheckBox. Каким образом можно обрабатывать сообщения от данных элементов управления и изменять их состояние?
14. Основные свойства элемента управления Edit?
15. Основные свойства элемента управления RadioButton?

Лабораторная работа № 2

Тема работы: разработка диспетчера задач с использованием WinAPI

Цель работы: научиться создавать снимки системы; управлять, запускать и завершать приложения из диспетчера задач.

Теоретические сведения по изучению темы

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами.

Процесс (или по-другому, задача) – абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

В многозадачной (многопроцессной) системе процесс может находиться в одном из трех основных состояний:

ВЫПОЛНЕНИЕ – активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором.

ОЖИДАНИЕ – пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса.

ГОТОВНОСТЬ – также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

Приоритет процесса – это число, характеризующее степень привилегированности процесса при использовании ресурсов вычисли-

тельной машины, в частности, процессорного времени: чем выше приоритет, тем выше привилегии.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной операционной системе

Основные функции для работы с процессами

Функция HANDLE WINAPI **CreateToolhelp32Snapsho**t (DWORDdwFlags, DWORDth32ProcessID) – возвращает моментальный снимок процессов, модулей или кучи, в зависимости от флага dwFlags. Входные параметры: DWORDdwFlags – определяет тип моментального снимка; DWORDth32ProcessID – необходим только в случае создания снимка модулей или снимка кучи, указывает идентификатор процесса, для которого будет создаваться моментальный снимок.

Функция BOOL WINAPI **Process32First** (HANDLE hSnapshot, LPPROCESSENTRY32 lppe) – возвращает TRUE, в случае успеха получения информации о первом процессе из снимка процессов и FALSE в обратном случае. Заполняет структуру lppe необходимой информацией о первом процессе из снимка, сделанного функцией **CreateToolhelp32Snapshot**. Входной параметр: HANDLE hSnapshot – моментальный снимок, созданный функцией CreateToolhelp32Snapshot с флагом dwFlags установленным в TH32CS_SNAPPROCESS; Выходной параметр: LPPROCESSENTRY32 lppe – указатель на структуру, содержащую базовую информацию о процессе.

Функция BOOL WINAPI **Process32Next** (HANDLE hSnapshot,LPPROCESSENTRY32 lppe) – возвращает TRUE, в случае если список процессов, созданный снимком еще не окончен и FALSE в обратном случае. Заполняет структуру lppe необходимой информцией о процессе. Входной параметр: HANDLE hSnapshot– моментальный снимок, созданный функцией CreateToolhelp32Snapshot с флагом dwFlags установленным в TH32CS_SNAPPROCESS; LPPROCESSENTRY32 lppe – указатель на структуру, содержащую базовую информацию о процессе.

Функция BOOL **GetProcessMemoryInfo** (HANDLEProcess, PPROCESS_MEMORY_COUNTER SppsmemCounters, DWORDcb) –

получает информацию о памяти, используемой процессом, возвращет TRUE, если запрос прошел успешно и FALSE в обратном случае. Входные параметры: HANDLEProcess – дескриптор процесса, полученный с помощью функции OpenProcess, у которой входной параметр, отвечающий за флаг доступа к процессу, установлен в значение PROCESS_QUERY_INFORMATION; DWORD cb – размер структуры PROCESS_MEMORY_COUNTERS; Выходной параметр: PROCESS_MEMORY_COUNTER SppsmemCounters – указатель на структуру, содержащую базовую информацию о памяти, используемой процессом с дескриптором Process;

Функция HANDLE **OpenProcess** (DWORD dwDesiredAccess,BOOLbInheritHandle,DWORDdwProcessId) – возвращает дескриптор процесса. Входные параметры: DWORD dwDesiredAccess – флаг доступа к процессу, определяет тип операции, которую потом можно производить с процессом; BOOL bInheritHandle – определяет, может ли данный дескриптор быть унаследован, в данной задаче должен быть установлен в FALSE; DWORD dwProcessId – идентификатор процесса для определения его дескриптора.

Функция BOOL **TerminateProcess** (HANDLE hProcess, UINT uExitCode) – завершает процесс, определенный дескриптором hProcess. Входные параметры: HANDLE hProcess – дескриптор процесса, полученный с помощью функции OpenProcess, у которой входной параметр, отвечающий за флаг доступа к процессу, установлен в значение PROCESS_TERMINATE;UINTuExitCode – определяет код выхода для процесса и всех потоков, запущенных процессом.

Функция BOOL WINAPI **Module32First** (HANDLE hSnapshot, LPMODULEENTRY32 lpme) – возвращает TRUE, в случае успеха получения информации о первом модуле процесса и FALSE в обратном случае. Заполняет структуру lpme необходимой информацией о первом модуле процесса из снимка, сделанного функцией CreateToolhelp32Snapshot, вызванной с флагом dwFlags, равным TH32CS_SNAPMODULE. Входной параметр: HANDLE hSnapshot– моментальный снимок, созданный функцией CreateToolhelp32Snapshot с флагом dwFlags, установленным в TH32CS_SNAPMODULE; Выходной параметр: LPMODULEENTRY32 lpme – указатель на структуру, содержащую базовую информацию о первом модуле процесса.

Функция BOOL WINAPI **Module32Next** (HANDLE hSnapshot, LPMODULEENTRY32 lpme) – возвращает TRUE, в случае успеха получения информации о модуле процесса и FALSE в обратном случае. Заполняет структуру lpme необходимой информацией модуле процесса из снимка, сделанного функцией CreateToolhelp32Snapshot, вызванной с флагом dwFlags, равным TH32CS_SNAPMODULE.

Функция DWORD **GetPriorityClass** (HANDLE hProcess)– возвращает приоритет процесса, заданного дескриптором hProcess. Входной параметр: HANDLE hProcess – дескриптор процесса, полученный с помощью функции OpenProcess, у которой входной параметр, отвечающий за флаг доступа к процессу, установлен в значение PROCESS_QUERY_INFORMATION.

Функция BOOL **SetPriorityClass** (HANDLE hProcess, DWORD dwPriorityClass) – возвращает TRUE, если приоритет процесса hProcess был установлен, и FALSE в обратном случае. Входные параметры: HANDLE hProcess – дескриптор процесса, полученный с помощью функции OpenProcess, у которой входной параметр, отвечающий за флаг доступа к процессу, установлен в значение PROCESS_SET_INFORMATION.

Структура **PROCESS_INFORMATION** заполняется функцией CreateProcess с информацией о недавно созданном процессе и его первичном потоке.

```
typedef struct _PROCESS_INFORMATION {
  HANDLE hProcess;
  HANDLE hThread;
  DWORD dwProcessId;
  DWORD dwThreadId;
} PROCESS_INFORMATION;
```

Члены структуры:

hProcess – дескриптор недавно созданного процесса. Дескриптор используется для заданного процесса всеми функциями, которые выполняют операции на объекте процесса.

hThread – дескриптор первичного потока недавно созданного процесса. Дескриптор используется для заданного потока всеми функциями, которые выполняют операции на объекте потока.

dwProcessId – значение, которое может быть использовано для идентификации процесса. Значение правильно с момента создания процесса до момента, пока процесс не завершит работу.

dwThreadId – значение, которое может быть использовано для идентификации потока. Значение правильно с момента создания потока до момента, пока поток не завершит работу.

Пример интерфейса приложения

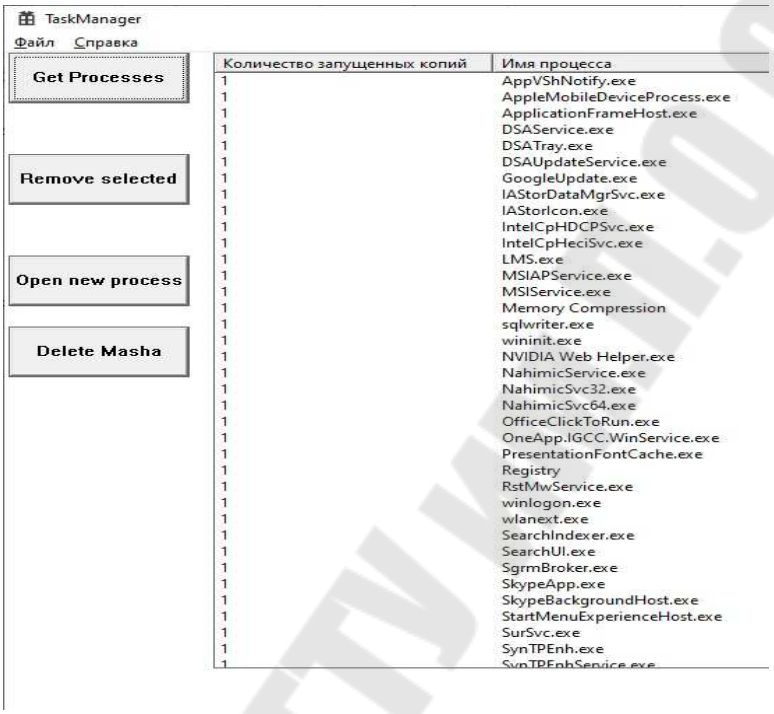

Рис. 2.1. Пример интерфейса диспетчера задач

Запущенные процессы отображаются в ListBox. Кнопка Remove selected отвечает за удаление выбранного процесса. Кнопка Open new process за запуск нового процесса. Кнопка Get processes заполняет ListBox. Пример создания и заполнения ListView выполняющимися процессами:

```
    listView = CreateWindowEx(0L, WC_LISTVIEW, L"",
                WS_VISIBLE | WS_CHILD | WS_BORDER | LVS_REPORT |
                LVS_EDITLABELS,
                windowRect.left + 100, 20, 400, 400,
                hWnd, (HMENU)4, hInst, NULL);
            LV_COLUMN  column;
            column.mask = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT |
LVCF_SUBITEM;
            column.cx = 200;
            column.fmt = LVCFMT_LEFT;
            column.iSubItem = 0;
            column.pszText = (LPWSTR)L"Имяпроцесса";
            ListView_InsertColumn(listView, 0, &column);
```

```
column.pszText = (LPWSTR)L"Idпроцесса";
column.iSubItem = 0;
column.cx = 200;
ListView_InsertColumn(listView, 1, &column);

processInfo.dwSize = sizeof(PROCESSENTRY32);
snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
Process32First(snapshot, &processInfo);
Int i = 0;
do
{
      LV_ITEM item;
      item.mask = LVIF_TEXT;
      item.iItem = i;
      item.iSubItem = 0;
      item.pszText = processInfo.szExeFile;
      ListView_InsertItem(listView, &item);
      wchar_tstrBuffer[60];
      wsprintfW(strBuffer, L"%d", processInfo.th32ProcessID);
      item.pszText = strBuffer;
      item.iItem = i;
      item.iSubItem = 1;
      ListView_SetItem(listView, &item);
      i++;
} while (Process32Next(snapshot, &processInfo));
ListView_SortItems(listView, 0, 0);
```

Рис. 2.2. Пример организации диспетчера задач

Контрольные вопросы

1. Что такое процесс?
2. В каких состояниях может находится процесс?
3. Что такое приоритет процесса?
4. Какой функцией можно получить список всех процессов?
5. Какая функция используется для завершения процесса?
6. Назначение, входные и выходные параметры функции CreateWindow.
7. Назначение, входные и выходные параметры функции CreateToolHelp32Snapshot.
8. Назначение, входные и выходные параметры функции GetProcessMemoryInfo.
9. Назначение, входные и выходные параметры функции OpenProcess.
10. Назначение, входные и выходные параметры функции TerminateProcess.

11. Назначение, входные и выходные параметры функции GetPriorityClass.

12. Назначение, входные и выходные параметры функции SetPriorityClass.

13. Назначение, входные и выходные параметры функции Process32First.

14. Назначение, входные и выходные параметры функции Process32Next.

15. Назначение, входные и выходные параметры функции CreateProcess.

Задание

Разработать программный комплекс (диспетчер задач), в котором отображаются все запущенные процессы в системе. Программа должна удалять запущенные процессы, а также запускать новые. Индивидуальные задания приведены в табл. 2.1.

Таблица 2.1

Индивидуальные задания

Вариант	**Задание**
1	Запуск всех процессов, находящихся в заданной папке по маске символа *
2	Удаление всех процессов, запущенных дважды
3	Запустить процессы из указанной папки, в имени которых более 10 символов
4	Пакетный запуск процессов, заполненных во втором Лист-боксе
5	Запуск всех процессов, находящихся в заданной папке по маске символа ?
6	Сортировка процессов по критерию — количество запусков на выполнение
7	В лист-боксе рядом с названием процесса отображать число его запусков(копий)
8	Запрещать запуск приложений не из диспетчера задач
9	Удалить все процессы, в имени которых есть хотя бы 2 одинаковые буквы
10	Установить таймер (10 с) на автоматическое обновление списка процессов

Окончание таблицы 2.1

Вариант	**Задание**
11	Удалять все процессы, в имени которых присутствует введенная подстрока
12	Все новые процессы запускать по 3 раза (как из диспетчера задач, так и извне)
13	Удалить все несистемные процессы
14	Удалить все процессы, в имени которых нет повторяющихся букв
15	Запрещать выполнение более одной копии приложения
16	Удаление пакета процессов, заполненных во втором Лист-боксе
17	Запускать все процессы, в имени которых присутствует введенная подстрока
18	Во второй лист бокс занести имена исполняемых файлов. Если такой процесс уже запущен – удалять все такие процессы, если нет – то запустить такой процесс
19	Отсортировать все процессы в порядке убывания имени процесса
20	Запустить все процессы из указанной папки. в имени которых есть хотя бы 2 одинаковые буквы
21	Запустить все процессы из указанной папки. в имени которых нет повторяющихся букв
22	Удалить процессы, в имени которых более 10 символов
23	Разделить процессы (по приоритету) на 2 лист-бокса (системные и прикладные)
24	Удалить все процессы по маске символа *
25	Раздельное отображение процессов, запущенных из разработанной программы и без ее использования
26	Удалить все процессы по маске символа ?
27	Пакетное удаление процессов, имена которых присутствуют во втором лист-боксе
28	Пакетный запуск процессов, имена которых присутствуют во втором лист-боксе

Лабораторная работа № 3

Тема работы: Использование потоков

Цель работы: Научиться создавать и управлять многопоточными приложениями

Теоретические сведения по изучению темы

Поток (Thread) – системный объект, соответствующий последовательности («потоку») команд, выполняемой независимо и асинхронно по отношению к другим подобным последовательностям. Поток – базовый объект, которому планировщик задач ОС распределяет время центрального процессора. Каждый процесс имеет как минимум один главный (первичный –primary) поток. Главный поток может порождать подчиненные (вторичные) потоки, которые будут выполняться одновременно с ним, равно как и с потоками прочих процессов.

Первичный поток создается системой при запуске исполняемого файла либо при создании дочернего процесса вызовом функции API **CreateProcess**. Для создания вторичных потоков необходимо воспользоваться функцией API **CreateThread**, которая создает новый поток в адресном пространстве процесса. Прототип данной функции представлен на рис. 3.1

```
HANDLE CreateThread(
      LPSECURITY_ATTRIBUTES lpThreadAttributes, // указатель на структуру,
      // описывающую параметры защиты потока. Если параметру присвоено
      // значение NULL, то устанавливаются атрибуты по умолчанию.
      DWORD dwStackSize, // устанавливает размер стека, который отводится
      // потоку. Если параметр равен нулю, то устанавливается стек, равный
      // стеку первичного потока
      LPTHREAD_START_ROUTINE lpStartAddress,// адрес входной потоковой
функции
      LPVOID lpParameter, // параметр, передаваемый в потоковую функцию
      DWORD dwCreationFlags, // флаг, который управляет созданием потока.
      // Если этот параметр равен CREATE_SUSPENDED, то поток после порожде-
ния   // не  запускается на исполнение.
      LPDWORD lpThreadId // адрес переменной типа DWORD, в которую функция
      // возвращает идентификатор нового потока
      );
```

Рис. 3.1. Функция создания вторичного потока

Поток можно приостановить на определенный период времени(указывается как параметр функции в милисекундах), вызвав функцию API **Sleep**. Также для приостановки («усыпления») потока служит функция API **SuspendThread**. Для возобновления работы потока служит функция API **ResumeThread**.

Поток может быть завершен в следующих случаях:

- функция потока возвращает управление (рекомендуемый способ);
- поток самоуничтожается вызовом функции **ExitThread**;
- один из потоков данного или другого процесса вызывает функцию **TerminateThread**;
- завершается процесс, содержащий данный поток.

Приоритеты потока

При создании потока он получает базовый приоритет в соответствии с классом своего процесса. После создания потока его относительный приоритет может изменяться как операционной системой, так и приложением с помощью функции API **SetThreadPriority**. Его параметрами являются: дескриптор потока и назначаемый приоритет. Создаваемым потокам чаще всего по умолчанию присваивается класс **Normal**. В табл. 3.1 представлены возможные приоритеты потоков.

Таблица 3.1

Приоритеты потоков

Приоритет потока	Идентификатор
Below normal	THREAD_PRIORITY_BELOW_NORMAL
Normal	THREAD_PRIORITY_NORMAL
Above normal	THREAD_PRIORITY_ABOVE_NORMAL
Highest	THREAD_PRIORITY_HIGHEST
Realtime	THREAD_PRIORITY_TIME_CRITICAL
Lowest	THREAD_PRIORITY_LOWEST
Idle	THREAD_PRIORITY_IDLE

Для получения относительного приоритета потока используется функция API **GetThreadPriority**.

Потоковые функции

Функция потока **ThreadProc()** принимает указатель на структуру с параметрами типа LPVOID и возвращает значение DWORD.

Функции потока могут выполнять любые поставленные задачи. Когда они закончат свою работу и вернут управление, поток остановится, память будет освобождена, счетчик пользователей его объекта ядра «поток» уменьшится на 1. Когда счетчик обнулиться, этот объект ядра будет разрушен.

Для получения времени создания и завершения потока используется функция **GetThreadTimes**. Ее параметры: дескриптор потока, время создания потока, время выхода потока, время работы в привилегированном режиме, время работы в пользовательском режиме. Если функция завершается успешно, величина возвращаемого значения – не ноль. Если функция завершается с ошибкой, величина возвращаемого значения – ноль.

Функция ожидания завершения потока – это **WaitForMultipleObjects**. Ее параметры: число элементов в массиве объектов (перехода в сигнальное состояние которых надо ждать); массив объектов; логический переключатель; максимальное время ожидания.

Для получения контекста потока служит функция **GetTheadContext**. Ее параметры: дескриптор потока и указатель на структура CONTEXT. Если функция завершается с ошибкой, величина возвращаемого значения – ноль.

Показатели времени работы потока определяются функцией **GetThreadTimes().** Параметры данной функции представлены в таблице 3.2.

Таблица 3.2

Время выполнения потока

Показатель времени	Описание параметра
Creation time	Абсолютная величина, выраженная в интервалах по 100 нс
Exit time	Абсолютная величина, выраженная в интервалах по 100 нс. Если поток все еще выполняется, этот показатель имеет неопределенное значение

Окончание таблицы 3.2

Показатель времени	Описание параметра
Kernel time	Относительная величина, выраженная в интервалах по 100 нс. Показывает время, затраченное потоком на выполнение кода операционной системой
User time	Относительная величина, выраженная в интервалах по 100 нс. Показывает время, затраченное потоком для выполнения кода приложения

На рис. 3.2 и 3.3 приведен фрагмент программы для создания потока, а также ее интерфейс.

```
HANDLE hThreadArray[MAX_THREAD];
	DWORD dwThreadIdArray[MAX_THREAD];

	int j = 0;
	int K = N;
	auto start = chrono::high_resolution_clock::now();
	for (int i = 0; i < MAX_THREAD; i++)
	{
		int p = 0;
		for (j; j < K; j++)
		{
			arr[p] = array[j];
			p++;
		}

		hThreadArray[i] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)SortArray, &arr, 0, &dwThreadIdArray[i]);
		SetThreadPriority(hThreadArray[i], THREAD_PRIORITY_HIGHEST);
		j += N - 1;
		K += N;

	}
	WaitForMultipleObjects(MAX_THREAD, hThreadArray, TRUE,INFINITE);
	auto end = chrono::high_resolution_clock::now();
	chrono::duration<float> duration = end - start;
	cout << endl << "Time = " << duration.count() << "s" << endl;
	for (int i = 0; i < MAX_THREAD; i++)
	{
		CloseHandle(hThreadArray[i]);
	}
```

Рис. 3.1. Пример программы создания потока

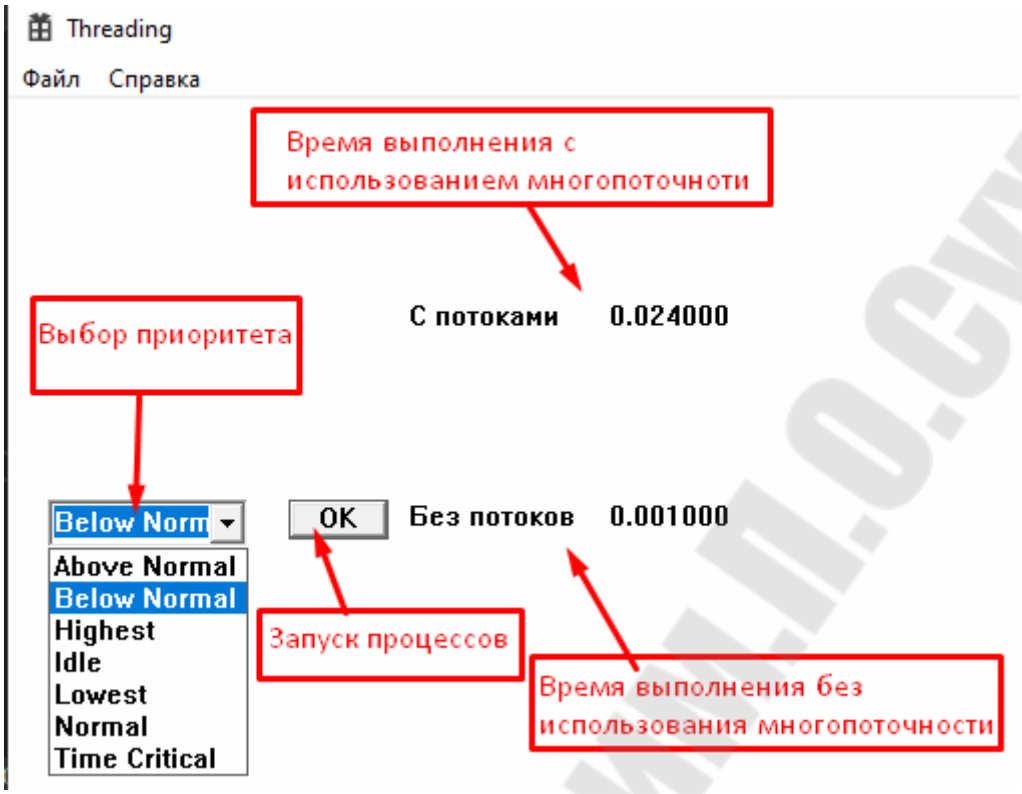

Рис.3.1. Пример интерфейса многопоточного приложения

Контрольные вопросы

1. Что такое поток? Какие бывают потоки?
2. Что такое потоковая функция?
3. Какая функция API позволяет создать вторичный поток?
4. Какие существуют способы завершения потока?
5. Какой способ завершения потока считается наиболее корректным?
6. Какая функция API позволяет приостановить поток на определенный период времени?
7. Посредством какой функции API можно «усыпить» поток?
8. Посредством какой функции API можно возобновить работу потока?
9. В чем состоит идея вытесняющего планирования на основе приоритетов?
10. Как определяется приоритет потока?
11. Какая функция API позволяет изменить относительный приоритет потока?
12. Какая функция API позволяет получить приоритет потока?

13. Какая функция API используется для получения дескриптора текущего потока?
14. Какая функция API используется для получения идентификатора текущего процесса?
15. Какая функция API используется для получения идентификатора текущего потока?

Индивидуальные задания

Создать потоки и процессы согласно индивидуальному заданию, приведенному в табл. 3.3. В основном окне приложения предоставить возможность выбора приоритета потока. Вывести время выполнения задания для каждого выбранного приоритета. Сравнить время выполнения задания без использования потоков и с использованием.

Таблица 3.3

Задания для выполнения лабораторной работы

Вариант	Задания
1	Сгенерировать массив на 10000 элементов. Разбить его на N частей. Создать N потоков, которые будут сортировать его части (метод пузырька). Организовать слияние этих частей в отсортированный массив
2	Сгенерировать массив на 10000 элементов. Создать N потоков, которые будут сортировать элементы, стоящие на местах, кратных N. Организовать слияние этих частей в отсортированный массив
3	Сгенерировать новый файл из нескольких других по принципу первая строка первого файла, вторая – второго и т. д., для чего создать N потоков (по числу файлов)
4	Сгенерировать N новых файлов из одного исходного файла по принципу первая строка в первый файл, вторая – во второй и т. д., для чего создать N потоков (по числу файлов)
5	Сгенерировать N новых файлов (по числу букв алфавита) из одного исходного файла по принципу: слова, начинающиеся на первую букву в первый файл, на вторую – во второй и т. д. Создать N потоков (по числу файлов)

Продолжение табл. 3.3

Вариант	Задания
6	Создать 4 потока, генерирующих координаты точек X,Y, цвет и толщину линии соответственно. Данные записать в общий файл, а в основном окне отрисовывать эти линии
7	Сгенерировать массив на 10000 элементов. Разбить его на N частей. Создать N потоков, которые будут сортировать его части (метод вставок). Организовать слияние этих частей в отсортированный массив
8	Сгенерировать массив на 10000 элементов. Разбить его на N частей. Создать N потоков, которые будут сортировать его части (метод выбора). Организовать слияние этих частей в отсортированный массив
9	Сгенерировать матрицу N на N элементов. Запустить N потоков, которые будут сортировать элементы в строках (быстрая сортировка). После организовать сортировку столбцов по значениям первых элементов строк
10	Сгенерировать матрицу N на N элементов. Запустить N потоков, которые будут сортировать элементы в столбцах (сортировка выбором). После организовать сортировку строк по значениям первых элементов столбцов
11	Создать 2 процесса, в которых создать по 2 потока. В каждом потоке необходимо считывать данные из своего файла по 100 байт и одновременно записывать в общий файл
12	При помощи трех потоков реализовать механические часы: первый поток должен двигать часовую стрелку, второй минутную и третий секундную. Записывать значения времени в общий файл каждые 10 с.
13	Реализовать N потоков по поиску седловой точки в матрице (считывается из файла). Каждый поток по строке ищет макс элемент, каждый поток по столбцу – мин элемент. Если полученный элемент находится на пересечении действия двух потоков, то его значение записывается в общий файл

Продолжение табл. 3.3

Вариант	Задания
14	Реализовать N потоков по инвертированию данных в матрице (считывается из файла). Каждый поток по строке инвертирует ее, каждый поток по столбцу – меняет элементы на четных и нечетных местах в столбце. Определить вероятность того, что строка N окажется в первоначальном состоянии. Данную строку записать в файл
15	Реализовать N потоков по инвертированию данных в матрице (считывается из файла). Каждый поток по столбцу инвертирует его, каждый поток по строке – инвертирует ее. Определить вероятность того, что столбец N окажется в первоначальном состоянии. Данный столбец записать в файл
16	Создать 2 потока по обмену элементов матрицы относительно главной и побочной диагонали (транспонирование). Матрицу считать из файла. Определить количество элементов, оставшихся на своих местах, в результате работы потоков за N секунд
17	Создать 4 потока по обмену элементов матрицы; 1 поток – обмен элементов, стоящих на четных и нечетных местах в столбце, 2 поток – обмен элементов, стоящих на четных и нечетных местах в строке, 3 поток – инвертирование элементов строки, 4 поток – инвертирование элементов столбца. Матрицу считать из файла. Определить количество элементов, оставшихся на своих местах, в результате работы потоков за N секунд
18	Реализовать N потоков по поиску и обмену элементов в матрице (считывается из файла). Для каждой строки организовать 3 потока: 1 поток по строке ищет макс элемент, 2 поток – мин элемент, 3 поток – меняет местами найденные элементы. Записать результат в общий файл

Продолжение табл. 3.3

Вариант	Задания
19	Создать 2 процесса , в которых создать по 2 потока. В каждом потоке необходимо записывать сгенерированные случайно данные (по 1 числу) в каждый из 4 общих файлов. Сравнить содержимое итоговых файлов. Вывести статистику по числу совпадений элементов в этих файлах.
20	N потоков считывают целые значения из своих файлов, формируют матрицу, транспонируют ее и записывают полученную матрицу в те же файлы
21	Четыре потока генерируют случайные числа, записывают их в общий файл и строят прямоугольник по сгенерированным координатам
22	Один процесс кодирует файл, во втором процессе создать два потока, которые дублируют шифрованный файл и декодируют его
23	Два процесса по два потока, работа с общим файлом. Каждый поток генерирует свою матрицу и записывает в свой файл. Каждый следующий поток считывает данные из файла предыдущего потока и записывает в свой
24	Три потока параллельно считывают матрицы с целыми числами из трех файлов и одновременно записывают в три новых файла (каждый поток пытается записать свою матрицу в каждый из файлов)
25	Реализовать восемь потоков, каждый из которых рисует постепенно удлиняющийся луч. Все лучи должны исходить из одной точки и быть направлены под разными углами. Данные для данных потоков генерируются случайно (прирост длины каждого луча) и записываются в общий файл
26	Реализовать шесть потоков, каждый из которых двигает свой шарик по окружности. Данные для данных потоков генерируются случайно (радиус траектории, радиус шарика , его цвет) и записываются в общий файл

Продолжение табл. 3.3

Вариант	**Задания**
27	Реализовать N потоков, каждый из которых рисует окружность. При наложении (касании) последних отрисованных окружностей, два соответствующих потока прекращают свою работу. Данные об окружностях (координаты центра, радиус и цвет) сохранить в общий файл
28	Реализовать N потоков, каждый из которых рисует прямоугольник. При наложении (касании) последних отрисованных прямоугольников, два соответствующих потока прекращают свою работу. Данные о прямоугольниках (координаты верхнего левого угла, ширина и высота) сохранить в общий файл

Лабораторная работа № 4

Тема работы: синхронизация доступа потоков к общим ресурсам.

Цель работы: изучить особенности функционирования объектов синхронизации, научиться использовать их при создании многозадачных и многопоточных приложений.

Теоретические сведения по изучению темы

Все потоки, принадлежащие одному процессу, разделяют некоторые общие ресурсы – такие, как адресное пространство оперативной памяти или открытые файлы. Если один из потоков запросит монопольный доступ к какому-либо ресурсу, другим потокам, которым тоже нужен этот ресурс, не удастся выполнить свои задачи. Именно поэтому необходим механизм, позволяющий потокам согласовывать свою работу с общими ресурсами. Этот механизм получил название Thread Synchronization (механизм синхронизации потоков). Синхронизацию потоков обычно осуществляют с использованием примитивов синхронизации, таких как:

1) атомарные операции API-уровня;
2) критические секции;
3) события;
4) ожидаемые таймеры;
5) семафоры;
6) мьютексы.

Работа с объектами синхронизации

Чтобы создать тот или иной объект синхронизации, производится вызов специальной функции WinAPI типа Create... (напр. CreateMutex). Этот вызов возвращает дескриптор объекта (HANDLE), который может использоваться всеми потоками, принадлежащими данному процессу.

Объекту, если только он не предназначен для использования внутри одного процесса, обязательно присваивается имя. Имена всех объектов должны быть различны (даже если они разного типа). Нельзя, например, создать событие и семафор с одним и тем же именем.

По имеющемуся дескриптору объекта можно определить его текущее состояние. Это делается с помощью ожидающих функций(wait-функции):

Функция Sleep() приостанавливает работу потока на заданное число миллисекунд.

Функция WaitForSingleObject() приостанавливает выполнение потока до тех пор, пока не произойдет одно из двух событий:

– истечет таймаут ожидания;

– ожидаемый объект перейдет в сигнальное (signaled) состояние.

Функции WaitForMultipleObjects передается сразу массив объектов. Можно ожидать срабатывания сразу всех объектов или какого-то одного из них.

Очень важен тот факт, что обращение к ожидающей функции блокирует текущий поток, т. е. пока поток находится в состоянии ожидания, ему не выделяется процессорное время.

Критические секции

Критическая секция позволяет сделать так, чтобы единовременно только один поток получал доступ к определенному ресурсу. Если в определенный момент времени несколько потоков попытаются получить доступ к критической секции, то контроль над ним будет предоставлен только одному из потоков, а все остальные будут переведены в состояние ожидания до тех пор, пока секция не освободится. Перед использованием ресурса поток входит в критическую секцию (вызывает функцию **EnterCriticalSection**). Если после этого какой-либо другой поток попытается войти в ту же самую критическую секцию, его выполнение приостановится, пока первый поток не покинет секцию с помощью вызова **LeaveCriticalSection**. Используется только для потоков одного процесса. Для использования критической секции необходимо определить переменную типа **CRITICAL_SECTION**. Поскольку эта переменная должна находиться в области видимости для каждого использующего ее потока, обычно ее объявляют глобальной. Пример использования критической секции приведен на рис. 4.1

```
DWORD WINAPI Write(LPVOID lp)
{
	srand(time(0));
	EnterCriticalSection(&cs);
	ofstream out(TEXT("array.txt"));
	int Arr[50];
	for(int i = 0; i < 50; i++)
	{
		Arr[i] = rand()%10;
		out << Arr[i] << ' ';
	}
	out.close();
	LeaveCriticalSection(&cs);
	MessageBox(0, TEXT("Поток завершен"),
			TEXT("Критическая секция"), MB_OK);
	return 0;
}
```

Рис. 4.1. Пример использования критической секции

Семейство Interlocked-функций

Win32 API предоставляет семейство **Interlocked-функций** для реализации взаимно блокированных операций. Данные функции работают на уровне атомарного доступа, т. е. без прерывания другими потоками. При этом все Interlocked-функции работают корректно только при условии, что их аргументы выровнены по границе двойного слова (DWORD) (рис. 4.2).

```
LONG g = 0; // определяем глобальную переменную


DWORD WINAPI Thread1( LPVOID lp )
{
      InterlockedIncrement(&g);
      return 0;
}


DWORD WINAPI Thread2( LPVOID lp )
{
      InterlockedIncrement(&g);
      return 0;
}
```

Рис. 4.2. Пример использования Interlocked-функций

Мьютексы

Объекты-взаимоисключения (мьютексы, mutex – от MUTual EXclusion) позволяют координировать взаимное исключение доступа к разделяемому ресурсу. Сигнальное состояние объекта (т. е. состояние «установлен») соответствует моменту времени, когда объект не принадлежит ни одному потоку и его можно «захватить». И наоборот, состояние «сброшен» (не сигнальное) соответствует моменту, когда какой-либо поток уже владеет этим объектом. Доступ к объекту разрешается, когда поток, владеющий объектом, освободит его.

Два (или более) потока могут создать мьютекс с одним и тем же именем, вызвав функцию **CreateMutex**. Первый поток действительно создает мьютекс, а следующие - получают дескриптор уже существующего объекта. Это дает возможность нескольким потокам получить дескриптор одного и того же мьютекса, освобождая программиста от необходимости заботиться о том, кто в действительности создает мьютекс. Если используется такой подход, желательно установить флаг bInitialOwner в FALSE, иначе возникнут определенные трудности при определении действительного создателя мьютекса.

Параметры данной функции: HANDLE CreateMutex(LPSECURITY_ATTRIBUTES lpMutexAttributes, // атрибуты доступа BOOL bInitialOwner, // флаг наличия потока-владельца LPCTSTR pszName // имя объекта);

Для получения доступа к разделяемому ресурсу поток обычно вызывает функцию **WaitForSingleObject** и передает ей дескриптор мьютекса. Параметры данной функции: DWORD WaitForSingleObject(HANDLE hHandle, // дескриптор объекта ядра «мьютекс» DWORD dwMilliseconds // время ожидания);. Возвращаемым значением функции WaitForSingleObject чаще всего является одна из следующих констант, приведенных в табл. 4.2.

Таблица 4.2

Результат работы функции WaitForSingleObject

Константа	Интерпретация
WAIT_OBJECT_0	Контролируемый объект ядра перешел в свободное (сигнальное) состояние, т. е. произошел захват мьютекса
WAIT_TIMEOUT	Истек интервал тайм-аута, а контролируемый объект ядра остался в занятом (несигнальном) состоянии
WAIT_FAILED	Функция завершилась с ошибкой
WAIT_ABANDONED	Объект мьютекса не был освобожден тем потоком, который им владел, по причине того, что поток был некорректно завершен. В этом случае мьютекс переводится в сигнальное состояние и им овладевает тот поток, который его ожидал

Пример использования мьютекса приведен на рис. 4.3

```
DWORD WINAPI ThreadNew(LPVOID lp)
{
	HANDLE hMutex = OpenMutex(MUTEX_ALL_ACCESS, false,
TEXT("{B8A2C543-22FE-423d-A123-865B2AF872E0}"));
	DWORD dwAnswer = WaitForSingleObject(hMutex, INFINITE);
	if(dwAnswer == WAIT_OBJECT_0)
	{
		…….
```

```
            ReleaseMutex(hMutex);
        }
        return 0;
}
```

Рис. 4.3. Пример использования мьютекса

События

Event-Объекты используются для уведомления ожидающих потоков о наступлении какого-либо события. Различают два вида событий – с ручным и автоматическим сбросом. Сброс события осуществляется функцией **ResetEvent**. События с ручным сбросом используются для уведомления сразу нескольких потоков. При использовании события с автоматическим сбросом уведомление получит и продолжит свое выполнение только один ожидающий поток, остальные будут ожидать дальше.

Функция **CreateEvent** создает объект-событие, **SetEvent** – устанавливает событие в сигнальное состояние, **ResetEvent** – сбрасывает событие. Функция **PulseEvent** устанавливает событие, а после возобновления ожидающих это событие потоков (всех при ручном сбросе и только одной при автоматическом), сбрасывает его. Если ожидающих потоков нет, PulseEvent просто сбрасывает событие. Функция создания события имеет следующий прототип:

```
HANDLE CreateEvent(
  LPSECURITY_ATTRIBUTES eventAttributes, // атрибуты доступа
  BOOL bManualReset, // тип сброса
  BOOL bInitialState, // начальное состояние
  LPCTSTR pszName // имя объекта
);
```

Пример использования объекта-события приведен на рис. 4.4

```
DWORD WINAPI ThreadNew(LPVOID lp)
```

```
{
	HWND hWnd = HWND (lp);
	// получим дескриптор существующего события
	HANDLE hEvent = OpenEvent(EVENT_ALL_ACCESS, 0,
			TEXT("{9BA6C76B-D9F8-1111-BB6F-E12358765139}"));
	//поток ожидает переход события в сигнальное состояние
	if(WaitForSingleObject(hEvent, INFINITE) == WAIT_OBJECT_0)
		{
			……….
		}
		ResetEvent(hEvent); // перевод события в несигнальное состояние
	}
	return 0;
}
```

Рис. 4.4. Пример использования объекта-события

Семафоры

Объект-семафор – это фактически объект-взаимоисключение со счетчиком. Данный объект позволяет «захватить» себя определенному количеству потоков. После этого «захват» будет невозможен, пока один из ранее «захвативших» семафор потоков не освободит его. Семафоры применяются для ограничения количества потоков, одновременно работающих с ресурсом. Объекту при инициализации передается максимальное число потоков, после каждого «захвата» счетчик семафора уменьшается. Сигнальному состоянию соответствует значение счетчика больше нуля. Когда счетчик равен нулю, семафор считается не установленным (сброшенным).

Функция **CreateSemaphore** создает объект-семафор с указанием и максимально возможного начального его значения, **OpenSemaphore** – возвращает дескриптор существующего семафора, захват семафора производится с помощью ожидающих функций, при этом значение семафора уменьшается на единицу, **ReleaseSemaphore** – освобождение семафора с увеличением значения семафора на указанное в пара-

метре число. Пример использования объекта-семафора приведен на рис. 4.5.

```
DWORD WINAPI Thread(LPVOID lp)
{
        HANDLE h = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE,
        TEXT("{22B4DB34-123A-4567-B456-1450D12654B3}"));
    if (WaitForSingleObject(h, INFINITE) == WAIT_OBJECT_0)
    {
    …….
        ReleaseSemaphore(h, 1, NULL);
    }
    CloseHandle(h);
    return 1;
}
```

Рис. 4.5. Пример использования объекта-семафора

Таймер синхронизации

Таймер ожидания переходит в сигнальное состояние по завершении заданного интервала времени. Для его создания используется функция CreateWaitableTimer. Для доступа к объекту используются следующие значения:

TIMER_ALL_ACCESS	**Разрешает полный доступ к объекту**
TIMER_MODIFY_STATE	Разрешает изменять состояние таймера функциями SetWaitableTimer и CancelWaitableTimer
SYNCHRONIZE	Только для Windows NT – разрешает использовать таймер в функциях ожидания

Задание:

Создать родительское приложение, в котором создается объект синхронизации и происходит запуск дочерних процессов. Объекты синхронизации выбираются самостоятельно. Графически отобразить процесс, представленный в задании. Задания по вариантам представлены в табл. 4.2.

Таблица 4.3

Индивидуальные задания

№	Задание
1	В пансионе отдыхают и предаются размышлениям 5 философов, пронумерованные от 1 до 5. В столовой расположен круглый стол, вокруг которого расставлены 5 стульев, также пронумерованные от 1 до 5. На столе находится одна большая тарелка со спагетти, которая пополняется бесконечно, также там расставлены 5 тарелок, в которые накладывается спагетти, и 5 вилок, назначение которых очевидно.Для того чтобы пообедать, философ входит в столовую и садится на стул со своим номером. При этом есть философ сможет только в том случае, если свободны две вилки – справа и слева от его тарелки. При выполнении этого условия философ поднимает одновременно обе вилки и может поглощать пищу в течение какого-то заданного времени. В противном случае, философу приходится ждать освобождения обеих вилок. Пообедав, философ кладет обе вилки на стол одновременно и уходит. Описанный процесс происходит бесконечно
2	В парикмахерской расположено единственное кресло, на котором спит парикмахер, и несколько стульев для клиентов. Когда клиент приходит в парикмахерскую, он будит парикмахера, садится в кресло. Стрижка производится в течение заданного времени. Если же кресло занято другим клиентом, то вновь прибывший клиент занимает любой свободный стул и ожидает своей очереди (клиенты обслуживаются в порядке очередности, например, времени прибытия). Если все стулья заняты, то клиент поворачивается и уходит. Когда обслужены все клиенты, парикмахер садится в кресло и снова засыпает. Описанный процесс происходит бесконечно

Продолжение табл. 4.3

№	Задание
3	В магазин непрерывно входят покупатели. Вместимость магазина — 10 человек. Каждый покупатель выбирает товары некоторое время и следует к одной из трех касс. После обслуживания покупатель покидает магазин
4	К заправке непрерывно подъезжают автомобили. Водители подходят к кассе (она одна) и оплачивают бензин. Затем подходят к одной из двух колонок для заправки (длится некоторое время). После покидают заправку
5	В регистратуру непрерывно подходят пациенты и получают талон к врачу(всего талонов 20). Тем, кому не хватило таланов, разворачиваются и уходят. После получения талона направляются к врачу (он один). Обслуживание длится некоторое время. После обслуживания покидают поликлинику
6	В университет постоянно подходят преподаватели и студенты. В университете находятся 3 аудитории по 10 мест для студентов и 1 месту для преподавателей. Занятие длится некоторое время, затем преподаватели и студенты покидают университет, а аудитории занимают новые студенты и преподаватели
7	Посетители постоянно подходят к ресторану, в котором находится 10 мест. Если все места заняты, то посетитель уходит. Затем к посетителю подходит официант (он один) для заказа и относит заказ к повару. Поле приготовления пищи, официант относит ее клиенту и тот покидает ресторан
8	При сдаче экзамена в ГИБДД постоянно приходят клиенты. Вместимость аудитории для сдачи теории составляет 15 мест. Время сдачи теоретической части – случайное число до 10 секунд. После поочередно все отправляются сдавать вождение (у нас один инструктор и один автомобиль)
9	В городском автобусе один кондуктор. Вместимость автобуса – 5 человек. Пассажиры постоянно прибывают, оплачивают поездку кондуктору и некоторое время едут. Затем выходят, новые пассажиры садятся в автобус

Продолжение табл. 4.3

№	Задание
10	В железнодорожный вокзал приходят пассажиры и становятся в очередь к кассе (их две). После получения билета они подходят и садятся в автобус (в нем 5 мест). Затем он отъезжает и приезжает новый автобус
11	Студенческая столовая имеет одну кассу. Студенты постоянно подходят и становятся в очередь в кассу. После оплаты они кушают некоторое время и покидают столовую
12	К магазину постоянно приходят клиенты. Условием работы магазина является наличие 4 клиентов, также одновременно в помещении может находиться не более 5 человек (покупателей). Если в магазине становиться меньше 4 покупателей магазин закрывается на перерыв(некоторое время). Каждый покупатель находится в магазине в промежутке от 1 до 8 с
13	В пансионе отдыхают и предаются размышлениям 5 философов, пронумерованные от 1 до 5. В столовой расположен круглый стол, вокруг которого расставлены 5 стульев, также пронумерованные от 1 до 5. На столе находится одна большая тарелка со спагетти, которая пополняется бесконечно, также там расставлены 5 тарелок, в которые накладывается спагетти, и 5 вилок, назначение которых очевидно. Для того чтобы пообедать, философ входит в столовую и садится на стул со своим номером. При этом есть философ сможет только в том случае, если свободны две вилки – справа и слева от его тарелки. При выполнении этого условия философ поднимает одновременно обе вилки и может поглощать пищу в течение какого-то заданного времени. В противном случае, философу приходится ждать освобождения обеих вилок. Пообедав, философ кладет обе вилки на стол одновременно и уходит. Описанный процесс происходит бесконечно

Продолжение табл. 4.3

№	Задание
14	В парикмахерской расположено единственное кресло, на котором спит парикмахер, и несколько стульев для клиентов. Когда клиент приходит в парикмахерскую, он будит парикмахера, садится в кресло. Стрижка производится в течение заданного времени. Если же кресло занято другим клиентом, то вновь прибывший клиент занимает любой свободный стул и ожидает своей очереди (клиенты обслуживаются в порядке очередности, например, времени прибытия). Если все стулья заняты, то клиент поворачивается и уходит. Когда обслужены все клиенты, парикмахер садится в кресло и снова засыпает. Описанный процесс происходит бесконечно
15	В магазин непрерывно входят покупатели. Вместимость магазина — 10 человек. Каждый покупатель выбирает товары некоторое время и следует к одной из трех касс. После обслуживания покупатель покидает магазин
16	К заправке непрерывно подъезжают автомобили. Водители подходят к кассе (она одна) и оплачивают бензин. Затем подходят к одной из двух колонок для заправки (длится некоторое время). После покидают заправку
17	В регистратуру непрерывно подходят пациенты и получают талон к врачу (всего талонов 20). Тем, кому не хватило таланов, разворачиваются и уходят. После получения талона направляются к врачу (он один). Обслуживание длится некоторое время. После обслуживания покидают поликлинику
18	В университет постоянно подходят преподаватели и студенты. В университете находятся 3 аудитории по 10 мест для студентов и 1 месту для преподавателей. Занятие длится некоторое время, затем преподаватели и студенты покидают университет, а аудитории занимают новые студенты и преподаватели

Продолжение табл. 4.3

№	Задание
19	Посетители постоянно подходят к ресторану, в котором находится 10 мест. Если все места заняты, то посетитель уходит. Затем к посетителю подходит официант (он один) для заказа и относит заказ к повару. Поле приготовления пищи, официант относит ее клиенту и тот покидает ресторан
20	При сдаче экзамена в ГИБДД постоянно приходят клиенты. Вместимость аудитории для сдачи теории составляет 15 мест. Время сдачи теоретической части — случайное число до 10 с. После поочередно все отправляются сдавать вождение (у нас один инструктор и один автомобиль)
21	В городском автобусе один кондуктор. Вместимость автобуса – 5 человек. Пассажиры постоянно прибывают, оплачивают поездку кондуктору и некоторое время едут. Затем выходят, новые пассажиры садятся в автобус
22	В железнодорожный вокзал приходят пассажиры и становятся в очередь к кассе (их две). После получения билета они подходят и садятся в автобус (в нем 5 мест). Затем он отъезжает и приезжает новый автобус
23	Студенческая столовая имеет одну кассу. Студенты постоянно подходят и становятся в очередь в кассу. После оплаты они кушают некоторое время и покидают столовую
24	К магазину постоянно приходят клиенты. Условием работы магазина является наличие 4 клиентов, также одновременно в помещении может находиться не более 5 человек (покупателей). Если в магазине становиться меньше 4 покупателей магазин закрывается на перерыв(некоторое время). Каждый покупатель находится в магазине в промежутке от 1 до 8 с

Продолжение табл. 4.3

№	Задание
25	В магазин непрерывно входят покупатели. Вместимость магазина – 10 человек. Каждый покупатель выбирает товары некоторое время и следует к одной из трех касс. После обслуживания покупатель покидает магазин
26	К заправке непрерывно подъезжают автомобили. Водители подходят к кассе (она одна) и оплачивают бензин. Затем подходят к одной из двух колонок для заправки (длится некоторое время). После покидают заправку
27	В регистратуру непрерывно подходят пациенты и получают талон к врачу(всего талонов 20). Тем, кому не хватило таланов, разворачиваются и уходят. После получения талона направляются к врачу (он один). Обслуживание длится некоторое время. После обслуживания покидают поликлинику
28	В университет постоянно подходят преподаватели и студенты. В университете находятся 3 аудитории по 10 мест для студентов и 1 месту для преподавателей. Занятие длится некоторое время, затем преподаватели и студенты покидают университет, а аудитории занимают новые студенты и преподаватели
29	Посетители постоянно подходят к ресторану, в котором находится 10 мест. Если все места заняты, то посетитель уходит. Затем к посетителю подходит официант (он один) для заказа и относит заказ к повару. Поле приготовления пищи, официант относит ее клиенту и тот покидает ресторан
30	При сдаче экзамена в ГИБДД постоянно приходят клиенты. Вместимость аудитории для сдачи теории составляет 15 мест. Время сдачи теоретической части — случайное число до 10 секунд. После поочередно все отправляются сдавать вождение (у нас один инструктор и один автомобиль)

Контрольные вопросы

1. В каких случаях возникает необходимость синхронизации потоков?
2. Какие существуют примитивы синхронизации?
3. Что такое атомарный доступ?
4. Какие преимущества имеют **Interlocked**-функции перед операторами C++? В каких случаях желательно их применять?
5. Что такое критическая секция?
6. Какая функция позволяет потоку завладеть критической секцией?
7. Посредством какой функции поток освобождает критическую секцию?
8. Каким преимуществом обладает критическая секция?
9. Что такое мьютекс?
10. В чем заключается принцип синхронизации потоков с помощью мьютексов?
11. В чем состоит основное отличие мьютекса от критической секции?
12. Какая функция WinAPI предусмотрена для создания мьютекса?
13. Посредством какой функции WinAPI поток освобождает захваченный мьютекс?
14. Какая функция WinAPI позволяет ждать освобождения сразу нескольких объектов синхронизации?
15. Что такое семафор? Чем он отличается от мьютекса?
16. В чем состоит принцип синхронизации работы потоков с использованием семафоров?
17. Какая функция WinAPI предусмотрена для создания семафора?
18. Какая функция WinAPI позволяет получить дескриптор существующего семафора?

Лабораторная работа № 5

Тема работы: менеджеры памяти, библиотеки динамической компоновки DLL, буфер обмена.

Цель работы: научиться выделять и использовать различные виды памяти, использовать библиотеки динамической компоновки при разработке приложений, а также использовать буфер обмена информации.

Теоретические сведения по изучению темы

Библиотеки динамической компоновки (dynamiclinklibraries – DLL) являются исполняемыми файлами особого формата, которые содержат функции, данные или ресурсы, доступные для других приложений.

Особый формат модулей DLL предполагает наличие в них разделов импорта и экспорта. Раздел экспорта указывает те идентификаторы объектов (функций, классов, переменных), доступ к которым разрешен для клиентов.

Подавляющее большинство DLL (за исключением DLL, содержащих только ресурсы) импортирует функции из системных DLL – kernel32.dll, user32.dll, gdi32.dll и других библиотек.

Применение DLL может дать ряд преимуществ:

– расширение функциональности приложения. DLL можно загружать в адресное пространство процесса на этапе выполнения, что позволит программе, определив, какие действия от нее требуются, подгружать нужный код;

– более простое управление проектом. Использование DLL упрощает отладку, тестирование и сопровождение проекта;

– экономия памяти. Если одну и ту же DLL используют несколько приложений, то в оперативной памяти хранится только один ее экземпляр, доступный этим приложениям;

– разделение ресурсов. DLL могут содержать такие ресурсы, как строки, растровые изображения, шаблоны диалоговых окон. Этими ресурсами может воспользоваться любое приложение;

– возможность использования разных языков программирования.

Существует три способа загрузки DLL:

– неявная;

– явная;
– отложенная.

Управление памятью

С помощью функции **VirtualAlloc()** есть возможность выделить или зарезервировать страницы виртуальной памяти. При выделении физически выделяется память и, естественно, увеличивается файл подкачки. Формат функции: LPVOID VirtualAlloc(LPVOID lpAddress, // базовый адрес SIZE_T dwSize, // размер DWORD flAllocationType, // способ получения DWORD flProtect// тип доступа).

Функции **HeapCreate()** и **HeapDestroy()** позволяют создавать и разрушать частную кучу. Формат функции: HANDLE HeapCreate (DWORD flOptions, // атрибуты SIZE_T dwInitialSize, // начальный размер SIZE_T dwMaximumSize // конечный размер). Создав кучу, есть возможность выделить в ней память. LPVOID **HeapAlloc** (HANDLE hHeap, // указатель на кучу где можно выделить память DWORD dwFlags, // флаги SIZE_T dwBytes // обьем выделяемой памяти). Для освобождения памяти используется функция **HeapFree**.

Получение глобального блока памяти происходит с помощью функции **GlobalAlloc().** Формат функции HGLOBAL WINAPI GlobalAlloc(UINT fuAlloc, DWORD cbAlloc). Параметр **fuAlloc** определяет тип выделяемой памяти. Размер блока памяти в байтах должен передаваться через параметр **cbAlloc**, причем можно заказать блок памяти размером больше, чем 64 Кбайт. Для стандартного режима работы Windows можно заказать блок памяти размером до 1 Мбайт без 80 байт, для расширенного – до 16 Мбайт без 64 Кбайт. Функция возвращает идентификатор глобального блока памяти или **NULL**, если Windows не может выделить память указанного объема. Функция **GlobalLock** фиксирует блок памяти, идентификатор которого передается ей через параметр **hglb** и возвращает логический адрес зафиксированного блока или **NULL**, если указанный блок удален или произошла ошибка. Для разблокирования области памяти используется функция **GlobalUnlock**.

Для получения локального блока памяти вы должны использовать функцию **LocalAlloc** Формат команды: HLOCAL WINAPI LocalAlloc(UINT fuAlloc, UINT cbAlloc). Параметр **fuAlloc** определяет тип выделяемой памяти. Размер блока памяти в байтах

должен передаваться через параметр **cbAlloc**. Функция возвращает идентификатор локального блока памяти или **NULL**, если Windows не может выделить память указанного объема. Для получения доступа к полученному блоку памяти его необходимо зафиксировать, вызвав функцию **LocalLock** . Формат функции: void NEAR* WINAPI Local-Lock(HLOCAL hloc). Функция **LocalLock** фиксирует блок памяти, идентификатор которого передается ей через параметр **hloc** и возвращает логический адрес зафиксированного блока или **NULL**, если указанный блок удален или произошла ошибка. Для освобождения локального блока памяти, полученного от функции **LocalAlloc,** необходимо использовать функцию **LocalFree.** Формат данной функции: HLOCAL WINAPI LocalFree(HLOCAL hloc). Идентификатор освобождаемого блока передается функции в качестве ее единственного параметра.

При копировании блоков памяти используется функция **memcpy().** Формат функции : void *memcpy(void *to, const void *from, size_t count). Она копирует count символов из массива, адресуемого параметром from, в массив, адресуемый параметром to. Если заданные массивы перекрываются, поведение функции memcpy() не определено. В версии C99 к параметрам to к from применен квалификатор **restrict.** Функция **memcpy()** возвращает значение указателя to**.**

Организация буфера обмена

Буфер обмена – набор функций и сообщений, который делает возможным передачу данных в программы. Все прикладные программы имеют доступ к буферу обмена, данные могут быть легко перемещены между приложениями. Форматы буфера обмена приведены в таблице 5.1

Таблица 5.1

Форматы данных буфера обмена

Формат	Тип данных
CF_BITMAP	Растр(bitmap) в чистом виде
CF_DIB	Растр(bitmap) с заголовком BITMAPINFO
CF_DIF	Универсальный формат обмена
CF_DSPBITMAP	Пользовательское растровое изображение
CF_DSPMETAFILEPICT	Пользовательский расширенный метафайл

Окончание табл.5.1

Формат	Тип данных
CF_DSPTEXT	Пользовательский текст
CF_ENHMETAFILE	Расширенный метафайл
CF_METAFILEPICT	Метафайл в стиле METAFILEPICT
CF_OEMTEXT	Текст в кодировке OEM
CF_OWNERDISPLAY	Пользовательский формат данных
CF_PALETTE	Цветовая палитра
CF_RIFF	Файл ресурсов
CF_PENDATA	Формат данных, связанных с электронным пером
CF_SYLK	Символическая ссылка
CF_TEXT	Текст
CF_WAVE	Звук в формате WAVE
CF_TIFF	Графика в формате TIFF
CF_UNICODETEXT	Текст в кодировке UNICODE

Перед тем чтобы записать в буфер обмена какую-либо информацию приложение должно его открыть, для этого следует использовать функцию ***OpenClipboard.*** После того как приложение открыло буфер обмена, она должна очистить его содержимое, которое было записано в него до его открытия, и вызвать функцию ***EmptyClipboard.*** Затем необходимо выделить блок глобальной памяти с помощью функции **GlobalAlloc()**, достаточный для того, чтобы хранить в нем данные. Затем необходимо получить указатель на полученную область памяти (функция **GlobalLock()).** Она фиксирует в памяти блок, дескриптор которого передается в параметре полученном при использовании **GlobalAlloc().** Когда работа с блоком памяти завершиться, его необходимо разблокировать, вызвав **GlobalUnlock().** После этого можно перемещать в него свои данные в различных форматах, используя функцию **SetClipboardData()**.

Обобщенный пример работы с буфером обмена представлен ниже:

```
Public void Copy(wchar_t* msg, HWND hWnd)

{
```

HGLOBAL imgBuf = GlobalAlloc(GHND,100); // Выделение блока памяти

Wchar_t* pasteBuff = (wchar_t*)GlobalLock(imgBuf); // Фиксируем блок в памяти

Wcscpy(pasteBuff, msg); // Копируем в блок памяти данные

GlobalUnlock(imgBuf); // Разблокировать блок

OpenClipboard(hWnd) // Открываем буфер обмена

EmptyClipboard(); // Очищаем буфер обмена

SetClipboardData(CF_BITMAP, imgBuf); // Помещаем данные в буфер обмена

CloseClipboard(); // Закрываем буфер обмена

}

Чтение из буфера обмена выполняется командой **GetClipboardData().** Если в буфер обмена данные поместила другая программа, то можно проверить доступные форматы перед их извлечением с помощью **IsClipboardFormatAvailable()**.

Для примера функционирования графического буфера обмена представлен рис. 5.2.

```
void Copy(HANDLE local, int size, int index, HWND hWnd)
{
	wchar_t* mapView = (wchar_t*)local;
	HBITMAP hBMP = (HBITMAP) LoadImage( NULL, mapView, IMAGE_BITMAP, 0, 0,
LR_LOADFROMFILE);

	OpenClipboard(hWnd);
	EmptyClipboard();
	SetClipboardData(CF_BITMAP, hBMP);
	CloseClipboard();
}

HBITMAP Paste(HWND hWnd) {
	OpenClipboard(hWnd);
	HBITMAP getBuf = (HBITMAP)GetClipboardData(CF_BITMAP);
	CloseClipboard();
	return getBuf;
}
```

Рис. 5.1. Пример копирования и вставки информации в буфер обмена

Пример выделения памяти для информации из буфера обмена представлен на рис. 5.2.

```
LPWSTR buff = new wchar_t[1024];
GetDlgItemText(hWnd, 3, buff, 1024);
size = _wtoi(buff);
localFirst = LocalAlloc(NULL,size);
if (isInitiate)
{
	localSecond = LocalAlloc(NULL,size);
	isInitiate = !isInitiate;
}
wchar_t* mapView = (wchar_t*)localFirst;
for (int i = 0; i < size / 2; i++)
	mapView[i] = L'\0';
UpdateListBox(localFirst, size, hWnd, 1);
```

Рис. 5.2. Пример выделения памяти под блок данных

Интерфейс программы для выполнения лабораторной работы может выглядеть так:

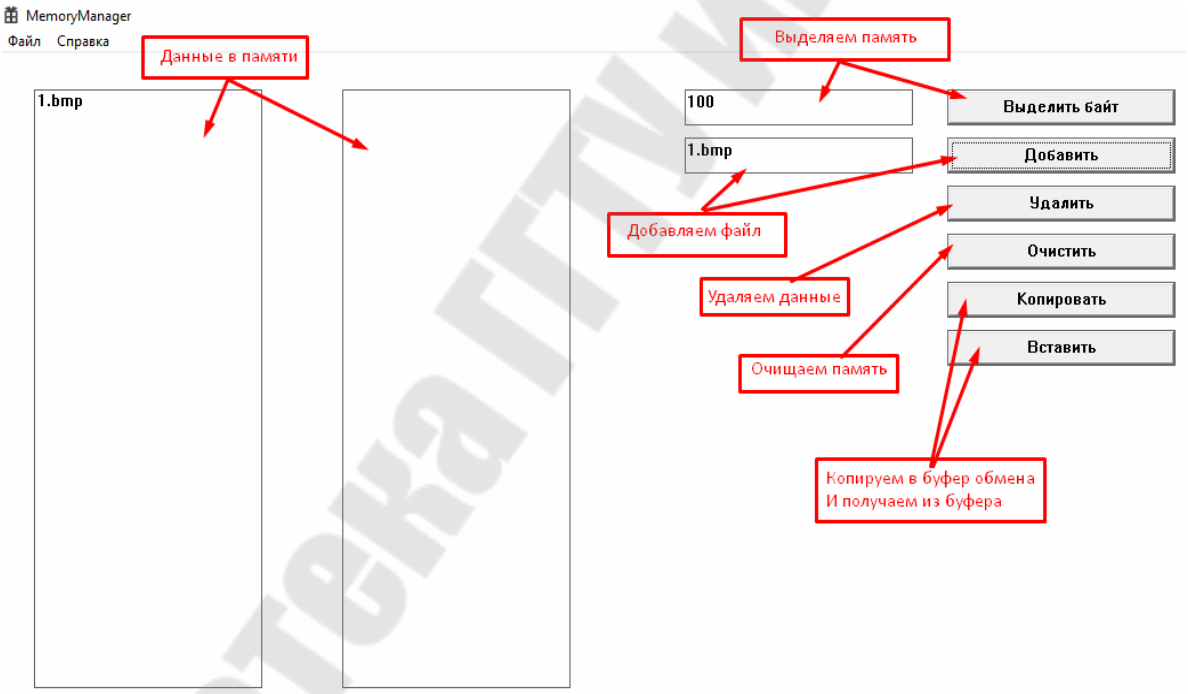

Рис. 5.3. Пример интерфейса программы

Маппинг файлов

Используя файлы, отображаемые в памяти, можно передавать информацию между разными процессами. Функция **CreateFile()** позволяет процессу открыть файл, который в свою очередь проецируется в память другими процессами. В свою очередь функция

CreateFileMapping() создает другие структуры и связывает сущность файла на диске с его сущностью в адресном пространстве. Далее необходимо использовать функцию **OpenFileMapping()** для открытия объекта проекции. И после того как отображение уже не нужно, его необходимо закрыть, используя **UnmapViewOfFile()**.

Контрольные вопросы

1. Что такое библиотека DLL?
2. Как происходит явное связывание?
3. Как происходит неявное связывание?
4. Что такое маппинг файлов и для чего он используется?
5. Что такое буфер обмена и для чего он нужен?
6. Можно ли создать пользовательский формат данных для буфера обмена? Если да, то как?
7. Как получить данные из буфера обмена?
8. Как записать данные в буфер обмена?
9. Как происходит копирование данных из одной области данных в другую?
10. Как происходит работа с глобальной памятью?
11. Как происходит работа с виртуальной памятью?
12. Как происходит работа с локальной памятью?
13. Как происходит работа с кучей?
14. Опишите назначение и особенности использования функции **CreateFileMapping**.
15. Какие форматы поддерживает буфер обмена?

Индивидуальные задания

В каждой программе должна присутствовать разработанная DLL – библиотека. Менеджер памяти должен выделять память, блокировать и разблокировать ее, записывать и считывать данные из нее, освобождать ее, копировать данные в другой блок. Индивидуальные задания представлены в табл. 5.2

Таблица 5.2

Варианты заданий для выполнения лабораторной работы

Вариант	Вид памяти (API)	DLL связывание	Буфер обмена (ClipBoard)
1	Виртуальная (VirtualAlloc)	Явное	Буфер обмена текстовый
2	Локальная (LocalAlloc)	Неявное	Буфер обмена графический
3	Глобальная (GlobalAlloc)	Явное	Буфер обмена текстовый
4	Куча (HeapAlloc)	Неявное	Буфер обмена графический
5	Проецирование файлов (map-ping)	Явное	Буфер обмена текстовый
6	Виртуальная (VirtualAlloc)	Неявное	Буфер обмена графический
7	Локальная (LocalAlloc)	Явное	Буфер обмена текстовый
8	Глобальная (GlobalAlloc)	Неявное	Буфер обмена графический
9	Куча (HeapAlloc)	Явное	Буфер обмена текстовый
10	Проецирование файлов (map-ping)	Неявное	Буфер обмена графический
11	Виртуальная (VirtualAlloc)	Явное	Буфер обмена графический
12	Локальная (LocalAlloc)	Неявное	Буфер обмена текстовый
13	Глобальная (GlobalAlloc)	Явное	Буфер обмена графический
14	Куча (HeapAlloc)	Неявное	Буфер обмена текстовый
15	Проецирование файлов (map-ping)	Явное	Буфер обмена графический

ЛИТЕРАТУРА

1. Вильямс, А. Системное программирование в Windows 2000 для профессионалов / А. Вильямс ; пер. с англ. – СПб. : Питер, 2001. – 624 с.

2. Гордеев, А. В. Системное программное обеспечение / А. В. Гордеев, А. Ю. Молчанов – СПб. : Питер, 2001. – 736 с.

3. Рихтер, Дж. Программирование на платформе Microsoft. NET Framework / Дж. Рихтер. – 3-е изд. – СПб. : Питер, Русская Редакция, 2005, – 486 с.

4. Рихтер, Дж. Windows для профессионалов / Дж. Рихтер. – СПб. : Питер, 2000. – 752 с.

5. Руссинович, М. Внутреннее устройство Microsoft Windows / М. Руссинович, Д. Соломон. – 4-е изд. – СПб. : Питер, Рус. ред., 2005. – 992 с.

6. Сорокина, С. И. Прогнраммирование драйверов и систем безопасности : учеб. пособие. / С. И. Сорокина, А. Ю. Тихонов, А. Ю. Щербаков. – СПб. : БХВ-Петербург, М. : издатель С. В. Молчанов – 2002. – 256 с.

7. Стивенс, У. Р. UNIX. Разработка сетевых приложений / У. Р. Стивенс, Б. Феннер, Э. М. Рудофф. – 3-е изд. – СПб. : Питер, 2007. – 1040 с.

8. Танненбаум, Э. Современные операционные системы / Э. Танненбаум. – 2-е изд. – СПб. : Питер, 2002. – 1040 с.

9. Танненбаум, Э. Операционные системы. Разработка и реализация / Э. Танненбаум, А. Вудхалл. – 3-е изд. – СПб.: Питер, 2007. – 704 с.

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Практикум
по выполнению лабораторных работ
по одноименной дисциплине
для студентов специальности 1-40 04 01
«Информатика и технологии программирования»
дневной формы обучения

Составитель **Косинов** Геннадий Петрович

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 17.05.21.
Рег. № 22Е.
http://www.gstu.by