Федеральное агентство по образованию

Пензенский государственный университет

Кафедра "Информационной безопасности систем и технологий"

Пояснительная записка к курсовой работе

по теме:

"Системное программирование в операционных системах"

Дисциплина: Безопасность операционных систем

Группа

Выполнил:.

Проект принят с оценкой

Руководитель проекта:

Пенза, 2006 г.

Реферат

Пояснительная записка содержит 38с., 7 рис., 2 источника, 2 прил.

VISUAL СИ++, ПРОГРАММА, РАСПИСАНИЕ, WINDOWSNT (98),ФАЙЛ*.*

Объектом исследования является системное программирование в операционных системах*.*

Целью работы является разработка программы, выполняющей запуск программ по расписанию: расписание хранится в файле.

В процессе работы были освоены навыки системного программирования в операционных системах*.*

В результате работы была разработана программа, выполняющая запуск программ по расписанию: расписание хранится в файле (её текст приведён в приложении).

Содержание

1 Интерфейс

1.1 Понятие интерфейса и его назначение

1.2 Выбор интерфейса

2. Разработка алгоритма программы

3. Создание программы на базе разработанного алгоритма

4. Руководство пользователя

Заключение

Список использованных источников

Алгоритм программы

## 1 Интерфейс

## 1.1 Понятие интерфейса и его назначение

Перед началом самого программирования и даже составления алгоритмов необходимо представить себе как будет работать программа и посредством этого выбрать интерфейс.

Под интерфейсом понимают совокупность схемотехнических средств и правил, обеспечивающих непосредственное взаимодействие составных элементов вычислительной системы и или программ. Интерфейс обеспечивает взаимосвязь между функциональными блоками или устройствами системы.

Основным назначением интерфейса является унификация внутрисистемных и межсистемных связей и устройств сопряжения с целью эффективной реализации прогрессивных методов проектирования функциональных элементов вычислительной системы.

Системный подход при проектировании представляет собой комплексное, взаимосвязанное, пропорциональное рассмотрение всех факторов, путей и методов решения сложной многофакторной и многовариантной задачи проектирования интерфейса взаимодействия. В отличие от классического инженерно - технического проектирования при использовании системного подхода учитываются все факторы проектируемой системы - функциональные, психологические, социальные и даже эстетические.

Автоматизация управления неизбежно влечёт за собой осуществление системного подхода, т.к. она предполагает наличие саморегулирующейся системы, обладающей входами, выходами и механизмом управления. Уже само понятие системы взаимодействия указывает на необходимость рассмотрения окружающей среды, в которой она должна функционировать.

Таким образом, система взаимодействия должна рассматриваться как часть более обширной системы - АСУ реального времени, тогда как последняя - системы управляемой среды.

В настоящее время можно считать доказанным, что главная задача проектирования интерфейса пользователя заключается не в том, чтобы рационально "вписать" человека в контур управления, а в том, чтобы, исходя из задач управления объектом, разработать систему взаимодействия двух равноправных партнёров (человек - оператор и аппаратно - программный комплекс АСУ), рационально управляющих объектом управления.

## 1.2 Выбор интерфейса

Интерфейс взаимодействия с пользователем является важным элементом любой программы. От его продуманности и удобства часто зависит коммерческий успех программного продукта.

Интерфейс пользователя можно разделить на пакетный и интерактивные. Пакетные характеризуются тем, что пользователь должен сформировать пакет с заданиями, затем программа эти задания выполняет и выдаёт результат. Интерактивные характеризуются тем, что пользователь в ходе работы программы постоянно с ней взаимодействует. Существует интерфейс на базе меню, псевдографический интерфейс, интерактивно - командный интерфейс, эти интерфейсы описывать не будем, так как программа, которую необходимо создать по заданию требует графический интерфейс. Графический интерфейс следует выбирать только для многомодульных программ, которые, предположительно будут иметь массовое применение.

В соответствии с заданием и для большего удобства и понятия принципа работы программы интерфейс должен содержать наименьшее количество кнопок. Вследствие чего выберем: 1 поле, в котором будет вводиться имя файла с расписанием, одну кнопку после нажатия, которой будет происходить запуск программ, кнопку для приостановления работы, копку обзора, а также кнопку выход. Данная разработка представлена на рисунке 1.

Файл с расписанием

Обзор

Установить расписание

Остановить работу

Выход

Рисунок 1. Интерфейс программы.

## 2. Разработка алгоритма программы

В данном разделе проводится разработка алгоритма работы программы, выполняющей запуск программ по расписанию, расписание которых храниться в файле.

Прежде чем начать разработку, вводим несколько ограничений:

а) Общие требования к программе:

исходные данные должны храниться в файле;

программа должна обеспечивать взаимодействие с пользователем посредством графического интерфейса;

процесс выполнения запуска программ должен быть реализован в интегрированной среде разработки Microsoft Visual C++ v.6.0.

б) Требования к организации файловых операций:

обеспечить возможность задания любого имени для файла с расписанием;

обеспечить обработку ошибочных ситуаций при работе с файлом с выдачей соответствующих информационных сообщений пользователю.

программа должна быть организованна в виде приложения.

в) Дополнительное требование:

в файле с расписанием имя программы должно быть указано полностью и заключено в кавычки, а далее через пробел должно быть указано время формата 00: 00: 00;

расписание может быть организовано и не по возрастанию;

После запуска программы пользователь должен будет ввести имя файла с расписанием. После нажатия клавиши “Установить расписание” программа должна будет проверить существование данного файла, если произошла ошибка, будет выдаваться об этом сообщение пользователю. После открытия файла программа будет считывать строчку за строчкой, в которых будет храниться имена и время запускаемых программ и происходить проверка на их существование. Далее должна будет происходить проверка на окончание файла.

После завершения работы программы должен выдаваться отчёт о её состоянии.

Структурная схема алгоритма программы, выполняющей запуск программ по расписанию, расписание которых храниться в файле, приведена в приложении А.

## 3. Создание программы на базе разработанного алгоритма

В данном разделе приводится описание текста программы выполняющей генерацию паролей пользователей. Данная программа реализована на языке программирования С++ при помощи интегрированной среды обработки Visual С++.

Основные принципы работы графических приложений Windows**.**

В отличие от программ MS-DOS, графическая программа Windows после запуска находится в "пассивном" состоянии. Она постоянно ожидает от операционной системы управляющих данных, которые называются сообщениями. Сообщения могут быть разного типа, например сообщение о нажатии клавиши, сообщение о движении курсора "мыши", управляющее сообщение операционной системы и т.д. Таким образом, работа графического приложения заключается в следующем: вначале программа должна сформировать на экране главное окно программы. Весь вывод программа будет осуществлять в дальнейшем в это окно в графическом виде. Затем программа должна включиться в обработку сообщений, которые посылает ей операционная система. Результат обработки сообщений может отображаться в главном окне программы.

Любое графическое приложение Windows имеет типовую структуру, основой которой является каркасное приложение. Это приложение содержит минимально необходимый программный код для обеспечения функционирования полноценного Windows-приложения.

Каркасное графическое приложение Windows.

Минимальное каркасное приложение Windows состоит из трех частей:

главной функции;

цикла обработки сообщений;

оконной функции.

Выполнение любого графического приложения начинается с главной функции, имеющей обязательное имя WinMain (). Она содержит код, осуществляющий инициализацию приложения в среде Windows. Видимым результатом работы функции является появление на экране графического объекта в виде окна. Последним действием кода главной функции является создание цикла обработки сообщений. После его создания приложение становится пассивным и начинает взаимодействовать с операционной системой посредством сообщений. Обработка поступающих сообщений осуществляется специальной функцией, называемой *оконной*. Особенность оконной функции заключается в том, что она вызывается из операционной системы, а не из приложения, ее содержащего.

Для создания каркасного приложения в MS Visual C++ 6.0 необходимо создать новый проект, используя меню "File - >New‑>Projects‑>Win32 Application". Далее из предложенных вариантов необходимо выбрать "A simple Win32 application". В этом случае будет создано приложение с пустой главной функцией. Для создания каркасного приложения в эту функцию необходимо добавить код инициализации окна программы и цикл обработки сообщений. Кроме того, в приложение необходимо добавить код оконной функции.

Главная функция имеет четыре параметра. Первый параметр является идентификатором данной программы, второй параметр всегда имеет значение NULL, третий параметр содержит командную строку, с помощью которой программа была запущена, а четвертый параметр содержит информацию о том, как будет отображаться окно программы.

Оконная функция может иметь произвольное имя, но тип возвращаемого результата и количество и тип передаваемых параметров для нее строго определены. Прототип оконной функции должен выглядеть так:

LRESULT CALLBACK WindowProc (HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

Где первый параметр - идентификатор окна, которое обслуживается данной оконной функцией, второй параметр - идентификатор сообщения операционной системы, которое передается на обслуживание, третий и четвертый параметры - дополнительные данные, передаваемые вместе с сообщением. Обычно прототип оконной функции записывают в начале программы, до функции WinMain (), а сам код функции помещают в конец программы. Это делается потому, что функция должна быть определена до ее использования в WinMain ().

Таким образом, первым шагом, создания каркасного приложения будет определение прототипа оконной функции в начале программы (до функции WinMain).

Вторым шагом создания каркасного приложения будет создание и инициализация структуры с информацией класса окна. Для этого необходимо в функции WinMain объявить переменную данного типа и заполнить все десять полей структуры:

WNDCLASS wcl;

// объявление переменной типа структуры класса окна

wcl. style=CS\_HREDRAW|CS\_VREDRAW;

// стиль окна. В данном случае показывает, что окно // будет перерисовываться при изменении горизонтального // и вертикального размеров.

wcl. lpfnWndProc=WindowProc;

// указатель на оконную функцию, которая будет // вызываться операционной системой для данного окна.

wcl. cbClsExtra=0;

// количество дополнительных байт для следующей // структуры.

wcl. cbWndExtra=0;

// количество дополнительных байт для следующего окна.

wcl. hInstance=hInstance;

// идентификатор процедуры (программы), владеющей // данным окном.

wcl. hIcon=LoadIcon (NULL, IDI\_APPLICATION);

// идентификатор класса значка (иконки) для данного // окна. Значок включается в проект в виде ресурса. В // данном случае функцией LoadIcon () загружается // стандартный значок для приложений Windows.

wcl. hCursor=LoadCursor (NULL, IDC\_ARROW);

// идентификатор класса курсора для данного окна. Курсор // включается в проект в виде ресурса. В данном случае // функцией LoadCursor () загружается стандартный курсор // в виде стрелки.

wcl. hbrBackground= (HBRUSH) GetStockObject (WHITE\_BRUSH);

// идентификатор класса кисти для фона окна. В данном // случае используется белый цвет фона.

wcl. lpszMenuName=”MENU”;

// указатель на строку, идентифицирующую ресурс главного // меню окна.

wcl. lpszClassName="Pass";

// имя класса окна

Третьим шагом является создание окна. Он состоит из регистрации класса окна, создания окна, показа и перерисовки окна. Регистрация класса окна выполняется функцией RegisterClass ():

if (! RegisterClass (&wcl)) return 0;

В качестве параметра функции используется указатель на заполненную структуру класса окна. В случае успешной регистрации функция возвращает ненулевой результат. В противном случае функция возвращает ноль и приложение необходимо завершить.

Создание окна выполняется функцией CreateWindow (), имеющей прототип

HWND CreateWindow (

LPCTSTR lpClassName, // адрес имени зарегистрированного класса

LPCTSTR lpWindowName, // адрес имени окна

DWORD dwStyle, // стиль окна, константы WS\_xxxxxx

int x, // горизонтальная позиция окна

int y, // вертикальная позиция окна

int nWidth, // ширина окна

int nHeight, // высота окна

HWND hWndParent, // идентификатор "внешнего" окна

HMENU hMenu, // идентификатор меню окна

HANDLE hInstance, // идентификатор приложения

LPVOID lpParam // адрес дополнительных данных

);

Функция возвращает идентификатор открытого окна.

В данном случае в функцию WinMain можно добавить такую строку:

HWND hWnd = CreateWindow (wcl. lpszClassName, "Window", WS\_OVERLAPPEDWINDOW, // окно с заголовком и бордюром CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, // положение и размер "по умолчанию" NULL, NULL, hInstance, NULL);

// нет "внешнего" окна (это главное окно программы),

// используется меню класса окна,

// идентификатор программы - второй параметр функции WinMain ()

// нет дополнительных данных.

Далее, для показа окна на экране, используется функция ShowWindow ():

ShowWindow (hWnd, nCmdShow);

Где первый параметр - идентификатор окна (которое должно быть уже зарегистрировано и открыто), а второй параметр - способ отображения окна. В качестве второго параметра используется способ отображения, переданный операционной системой функции WinMain (четвертый параметр).

И, наконец, для перерисовки окна используется функция UpdateWindow ():

UpdateWindow (hWnd);

В которую в качестве параметра передается идентификатор окна.

Четвертым шагом создания каркасного приложения будет создание цикла обработки сообщений:

MSG lpMsg;

while (GetMessage (&lpMsg, NULL, 0,0))

{

TranslateMessage (&lpMsg);

DispatchMessage (&lpMsg);

}

return lpMsg. wParam;

Функция GetMessage () выбирает сообщение из очереди сообщений и помещает его в структуру, адрес которой указан в качестве первого параметра. Функция работает только с сообщениями, адресованными данному приложению или его дочерним процессам. Данная функция возвращает 0, если было выбрано сообщение WM\_QUIT и не ноль во всех остальных случаях. Таким образом, цикл обработки сообщений завершается после получения сообщения WM\_QUIT. А затем завершается и сама функция WinMain ().

Минимальный цикл обработки сообщений должен состоять из вызова системной функции DispatchMessage (), которая передает сообщение оконной функции, связанной с данным окном. Перед вызовом функции DispatchMessage () сообщение может подвергаться обработке. Так, функция TranslateMessage преобразует сообщения от клавиатуры. В Windows от клавиатуры идут сообщения WM\_KEYDOUN и WM\_KEYUP, соответствующие нажатию и отпусканию клавиш. Эти сообщения содержат виртуальные коды клавиш. Функция TranslateMessage () преобразует эти сообщения в сообщения WM\_CHAR, содержащие Пятым, и последним, шагом создания каркасного приложения является написание оконной функции. Она выглядит так:

LRESULT CALLBACK WindowProc (HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam)

{

static HINSTANCE hInstance;

switch (message)

{

case WM\_CREATE:

{

hInstance = ( (LPCREATESTRUCT) lParam) - >hInstance;

DialogBox (hInstance, IDD\_DIAL,hWnd, MyDlgProc);

break;

}

case WM\_DESTROY:

{

PostQuitMessage (0);

break;

}

default:

return DefWindowProc (hWnd, message, wParam, lParam);

}

return 0;

}

Оконная процедура получает сообщение и анализирует его. После анализа сообщение либо обрабатывается, либо передает сообщение процедуре обработки сообщений по умолчанию. Например оконная функция обрабатывает сообщение - WM\_DESTROY. Это сообщение передается оконной функции при закрытии окна. Нормальной реакцией на такое сообщение является посылка сообщения WM\_QUIT для завершения приложения. Это сообщение посылается путем вызова функции PostQuitMessage (). Параметром этой функции является код завершения программы.

Таким образом получено минимальное каркасное приложение Windows.

**Создание меню.**

Для добавления меню к программе необходимо создать соответствующий ресурс. Для создания ресурса меню в проекте необходимо воспользоваться пунктом меню "Insert - > Resourse - > Resurs Type: Menu - > New". При этом в окне редактора появится возможность в интерактивном режиме создать систему меню. Для редактирования свойств элементов меню можно использовать "View‑>Properties". Каждому пункту меню необходимо присвоить свой идентификатор. После завершения создания ресурса необходимо его сохранить. При этом автоматически будут созданы два файла - resource. h и script. rc (где script - имя файла ресурсаd). После этого необходимо файл ресурса script. rc включить в проект в папку Source Files, а в файл StdAfh. h включить строку *#include "resource. h".* После того, как файл ресурса будет включен в проект, в окне проекта станет, доступна закладка ResourceView. Используя эту закладку можно просмотреть древовидную структуру ресурсов проекта. В папке menu будут показаны идентификаторы ресурсов меню.

Для того чтобы созданное меню отображалось в окне программы его надо указать при инициализации структуры класса окна:

wcl. lpszMenuName=“MENU”;

где “MENU" - выбранный мною идентификатор меню. После компиляции такого проекта в окне программы будет отображаться меню.

Для того, чтобы назначить пунктам меню конкретные действия, необходимо включить их в обработку оконной функцией. Выбор пунктов меню с помощью мыши отслеживается автоматически операционной системой и формируются сообщения WM\_COMMAND. В качестве параметра такого сообщения передается идентификатор выбранного пункта меню. Таким образом, в оконной функции необходимо перехватить сообщение типа WM\_COMMAND и проверить его параметр. А далее, в зависимости от параметра, выполнить то или иное действие.

Следует отметить, что используемые переменные следует объявить вне тела оператора switch.

**Вывод сообщений в отдельном окне.**

Самым простым способом вывода сообщений в отдельном окне является использование функции MessageBox (). Эта функция имеет следующий формат:

int MessageBox (

HWND hWnd,

LPCTSTR lpText,

LPCTSTR lpCaption,

UINT uType);

Где параметры имеют следующее значение:

hWnd - идентификатор окна-владельца создаваемого окна с сообщением;

lpText - указатель на строку с сообщением, строка должна заканчиваться символом с кодом 0;

lpCaption - указатель на строку с заголовком окна, строка должна заканчиваться символом с кодом 0;

uType - тип окна с сообщением, может быть комбинацией различных констант.д.опустимы следующие константы в этом параметре:

MB\_YESNO - окно будет иметь две кнопки: Yes и No.

MB\_YESNOCANCEL - окно будет иметь три кнопки: Yes, No, и Cancel.

Функция возвращает ноль в случае нехватки памяти для открытия окна. При успешном завершении функции она возвращает одно из следующих значений:

IDABORT - выбрана кнопка Abort.

IDCANCEL - выбрана кнопка Cancel.

IDIGNORE - выбрана кнопка Ignore.

IDNO - выбрана кнопка No.

IDOK - выбрана кнопка OK.

IDRETRY - выбрана кнопка Retry.

IDYES - выбрана кнопка Yes.

Если окно содержит кнопку Cancel, то функция возвращает значение IDCANCEL в случае нажатия клавиши ESC или выбора кнопки Cancel. Если кнопка Cancel отсутствует, то нажатие клавиши ESC не имеет эффекта.

На этом создание программы на языке Си закончилось.

**Создание процесса.**

**"**Процесс - это выполняющееся приложение, которое состоит из личного виртуального адресного пространства, кода, данных и других ресурсов операционной системы, таких как файлы, пайпы и синхронизационные объекты, видимые для процесса."

У процесса есть несколько объектов: адресное пространство, выполняемый модуль (модули) и все, что эти модули создают или открывают. Как минимум, процесс должен состоять из выполняющегося модуля, личного адресного пространства и ветви. У каждого процесса по крайней мере одна ветвь. Что такое ветвь? Фактически, ветвь - это выполняющаяся очередь. Когда Windows впервые создает процесс, она делает только одну ветвь на процесс. Эта ветвь обычно начинает выполнение с первой инструкции в модуле. Если в дальнейшем понадобится больше ветвей, он может сам создать их.

Когда Windows получает команду для создания процесса, она создает личное адресное пространство для процесса, а затем она загружает исполняемый файл в пространство. После этого она создает основную ветвь для процесса.

Под Win32 можно создать процессы из своих программ с помощью функции Createprocess. Она имеет следующих синтаксис:

BOOL CreateProcess (

LPCTSTR lpszApplicationName.

LPCTSTR IpszCommandLine,

LPSECURITYJUTRIBUTES ipsaProcess.

LPSECURITY\_ATTRIBUTES lpsaThread,

BOOL flnhentHandles,

DWORD fdwCreate,

LPVOID ipvEnvironment.

LPTSTR lpszCurDir,

LPSTARTUPINFO ipsiStartlnfo,

LPPROCESS\_INFORMATION lppiProcInfo):

Когда поток в приложении вызывает CreateProcess, система создает объект ядра "процесс" с начальным значением счетчика числа его пользователей, равным 1. Этот объект - не сам процесс, а компактная структура данных, через которую операционная система управляет процессом. (Объект ядра "процесс" следует рассматривать как структуру данных со статистической информацией о процессе) Затем система создает для нового процесса виртуальное адресное пространство размером 4 Гб и загружает в него код и данные как для исполняемого файла, так и для любых DLL (если таковые требуются).

Далее система формирует объект ядра "поток" (со счетчиком, равным 1) для первичного потока нового процесса. Как и в первом случае, объект ядра "поток" - это компактная структура данных, через которую система управляет потоком. Первичный поток начнет с исполнения стартового кода из стандартной библиотеки С, который - как всегда - вызовет функцию WinMain. Если системе удастся создать новый процесс и его первичный поток, CreateProcess вернет TRUE.

На этом программирование закончилось.

## 4. Руководство пользователя

Для начала работы с программой необходимо записать файл с расписанием следующего формата "C: /totalcmd/TOTALCMD. EXE" 00: 00: 00

Далее необходимо запустить файл с названием OSkurs. После запуска появится окно которое представлено на рис.10.



Рисунок 2.

Далее необходимо ввести в окошко полное имя файла в котором находится расписание и нажать на клавишу “Установить расписание”. После чего будет происходить запуск программ по расписанию. Если имя не будет указано, то появится окошко:



Рисунок 3.

Если данного файла не существует или имя указано не верно то выдаётся сообщение.



Рисунок 4

Если файл с расписанием записан не верно, то выдаётся сообщение:



Рисунок 5

Если имя программы указано с ошибкой или его не существует, то выдаётся сообщение и два варианта продолжения работы программы:



Рисунок 6.

После успешного окончания работы программы выдаётся сообщение:



Рисунок 7.

Для завершения расписания необходимо нажать на клавишу “Остановить работу". Для выхода из программы необходимо нажать на клавишу “Выход” или нажав на “крестик".

## Заключение

В процессе выполнения курсового проекта была разработана программа, выполняющая запуск программ по расписанию, расписание хранится в файле. Программа была написана при помощи интегрированной среды разработки Visual C++. Программа соответствует всем функциональным требованиям, заданным на курсовое проектирование. Взаимодействие с пользователем осуществляется с помощью графического интерфейса.

Таким образом, задание на курсовое проектирование было полностью выполнено.

## Список использованных источников

1. Келецкий Я. Энциклопедия языка Си/ Пер. с англ. - М.: Мир, 1992. - 687с.

2. Керниган Б., Ричи Д., Фьюер А. Язык программирования Си. Задачи по языку С / Пер. с англ. - М.: Финансы и статистика, 1985. - 279с.

## Алгоритм программы

Приложение А

Ввод пользователем имени файла с расписанием

Файл существует

Открытие файла

Считывание имени программы

программа существует

Считывание времени запуска

Файл закончился

Считывание следующего имени

Начало

А

Ошибка

---------------------- [на лист 2

Рисунок А.1. Алгоритм программы

Запуск программ по расписанию

Конец

Отчёт о завершении программы

А

------------------------ [с листа1

Рисунок А.1. Лист №2

Приложение Б

Текст программы

#include <windows. h>

#include "resource. h"

#include <io. h>

#define WM\_SHELLNOTIFY WM\_USER+5

#define IDC\_RESTORE 1000

#define TM 1001

typedef struct \_SHEDULEDATA {

char fname [120] ;

char tname [6] ;

int h;

int m;

int s;

int priority;

int delay;

} SHEDULEDATA;

char \* FileOpen (HWND hWnd);

int FindNext (SHEDULEDATA sd [30], int count);

void cstrcpy (char \*a,char \*b, int from, int count);

void MakeItStandart (SHEDULEDATA sd [30], int count);

void SubtDelay (SHEDULEDATA sd [30], int count, int delay);

int ReadShedule (HANDLE hFile,SHEDULEDATA sd [30]);

LRESULT CALLBACK WindowProc (HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam); // дескриптор окна

INT CALLBACK MyDlgProc (HWND hDlg, // обращается к dialog box

UINT uMsg, // сообщение

WPARAM wParam, // первый параметр сообщения

LPARAM lParam // второй параметр сообщения);

int APIENTRY WinMain (HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{

WNDCLASS wcl;

HWND hWnd;

MSG lpMsg;

// заполнение структуры с параметрами окна программы

wcl. style=CS\_HREDRAW|CS\_VREDRAW; // стиль окна

wcl. lpfnWndProc=WindowProc; // ф-ция обработки событий окна // оконная процедура

wcl. cbClsExtra=0; // дополнительные байты класса

wcl. cbWndExtra=0; // дополнительные байты окна

wcl. hInstance=hInstance; // дескриптор экземпляра окна // дескриптор пиктограммы

wcl. hIcon=LoadIcon (NULL, IDI\_APPLICATION);

wcl. hCursor=LoadCursor (NULL, IDC\_ARROW); // дескриптор курсора

wcl. hbrBackground= (HBRUSH) GetStockObject (WHITE\_BRUSH);); // цвет фона

wcl. lpszMenuName=NULL; // имя ресурса с главным меню программы // ресурс меню, связанный с окном

wcl. lpszClassName="Pass"; // имя класса окна

if (! RegisterClass (&wcl))

return 0;

hWnd=CreateWindow (wcl. lpszClassName,"window",WS\_OVERLAPPEDWINDOW,CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, NULL, NULL, hInstance, NULL);

ShowWindow (hWnd,SW\_HIDE);

UpdateWindow (hWnd);

while (GetMessage (&lpMsg, NULL, 0,0))

{

TranslateMessage (&lpMsg);

DispatchMessage (&lpMsg);

}

return lpMsg. wParam;

}

LRESULT CALLBACK WindowProc (HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam) // дескриптор окна

{

static HINSTANCE hInstance;

switch (message)

{

case WM\_CREATE:

{

hInstance = ( (LPCREATESTRUCT) lParam) - >hInstance;

DialogBox (hInstance, IDD\_DIAL,hWnd, MyDlgProc);

break;

}

case WM\_DESTROY:

{

PostQuitMessage (0);

break;

}

default:

return DefWindowProc (hWnd, message, wParam, lParam);

}

return 0;

}

INT CALLBACK MyDlgProc (HWND hDlg,UINT uMsg,WPARAM wParam,LPARAM lParam)

{

static SHEDULEDATA sd [30] ;

static int count,nextexe,delayexe;

static char name [300] ;

char \* n;

static BOOL frst, fileopen = FALSE,filewasopened=FALSE;

BOOL cp;

char tip [] ="запуск по расписанию\0";

UINT uId=111;

static NOTIFYICONDATA nfd;

int i;

static STARTUPINFO si;

static HANDLE hMenu,hTimer,hFile;

static PROCESS\_INFORMATION pi;

switch (uMsg)

{

case WM\_INITDIALOG:

{

si. cb=sizeof (STARTUPINFO);

GetStartupInfo (&si);

nfd. cbSize=sizeof (NOTIFYICONDATA);

nfd. hWnd=hDlg;

nfd. uID=uId;

nfd. uFlags=NIF\_MESSAGE|NIF\_TIP|NIF\_ICON;

nfd. uCallbackMessage=WM\_SHELLNOTIFY;

lstrcpy (nfd. szTip,tip);

nfd. hIcon=LoadIcon (NULL, IDI\_WINLOGO);

hMenu = CreatePopupMenu ();

AppendMenu (hMenu,MF\_ENABLED|MF\_STRING, IDC\_RESTORE,"восстановить");

AppendMenu (hMenu,MF\_ENABLED|MF\_SEPARATOR,0,0);

AppendMenu (hMenu,MF\_ENABLED|MF\_STRING, IDC\_EXIT,"закрыть");

frst=TRUE;

SetDlgItemText (hDlg, IDC\_TEXT,"");

return (TRUE);

}

case WM\_CLOSE:

{

if (filewasopened)

CloseHandle (hFile);

Shell\_NotifyIcon (NIM\_DELETE,&nfd);

DestroyMenu (hMenu);

SendMessage (GetParent (hDlg),WM\_DESTROY,0,0);

EndDialog (hDlg, 0);

return (TRUE);

}

case WM\_SIZE:

{

if (wParam==SIZE\_MINIMIZED)

{

ShowWindow (hDlg,SW\_HIDE);

Shell\_NotifyIcon (NIM\_ADD,&nfd);

}

return TRUE;

}

case WM\_TIMER:

{

cp=CreateProcess (sd [nextexe]. fname,NULL,NULL,FALSE,CREATE\_DEFAULT\_ERROR\_MODE,NULL,NULL,NULL,&si,&pi);

KillTimer (hDlg,TM);

if (! cp)

{

if (MessageBox (hDlg,"Приложение не удалось запустить. Продолжить работу?",sd [nextexe]. fname,MB\_YESNO|MB\_ICONERROR) ==IDNO)

SendMessage (hDlg,WM\_COMMAND, IDC\_STOP,0);

else

SendMessage (hDlg,WM\_COMMAND, IDC\_START,0);

}

else

{

nextexe=FindNext (sd,count);

if (nextexe<0)

{

SendMessage (hDlg,WM\_COMMAND, IDC\_STOP,0);

return TRUE;

}

delayexe=sd [nextexe]. delay;

sd [nextexe]. delay=-1;

SubtDelay (sd,count,delayexe);

if (! SetTimer (hDlg,TM,1000\*delayexe,NULL))

{

frst=TRUE;

if (MessageBox (hDlg,"В системе нет свободных таймеров. ",0,MB\_RETRYCANCEL| MB\_ICONERROR) ==IDRETRY)

SendMessage (hDlg,WM\_COMMAND, IDC\_START,0);

else

SendMessage (hDlg,WM\_COMMAND, IDC\_STOP,0);

}

}

return TRUE;

}

case WM\_SHELLNOTIFY:

{

POINT pn;

if (wParam==uId)

{

if (lParam==WM\_RBUTTONDOWN)

{

GetCursorPos (&pn);

SetForegroundWindow (hDlg);

TrackPopupMenu (hMenu,TPM\_RIGHTALIGN,pn. x,pn. y,0,hDlg,NULL);

PostMessage (hDlg,WM\_NULL,0,0);

}

else if (lParam==WM\_LBUTTONDOWN)

SendMessage (hDlg,WM\_COMMAND, IDC\_RESTORE,0);

}

return TRUE;

}

case WM\_COMMAND:

switch (LOWORD (wParam))

{

case IDC\_EXIT:

{

if (filewasopened)

CloseHandle (hFile);

Shell\_NotifyIcon (NIM\_DELETE,&nfd);

DestroyMenu (hMenu);

SendMessage (GetParent (hDlg),WM\_DESTROY,0,0);

EndDialog (hDlg,0);

return TRUE;

}

case IDC\_OPEN:

{

n=FileOpen (hDlg);

if (n! =0)

{

lstrcpy (name,n);

fileopen=TRUE;

SetDlgItemText (hDlg, IDC\_TEXT,name);

}

return TRUE;

}

case IDC\_START:

{

if (frst)

{

if (! fileopen)

{

i=GetDlgItemText (hDlg, IDC\_TEXT,name,120);

if (i==0)

{

MessageBox (hDlg,"введите имя файла с расписанием","ожидается ввод",MB\_OK);

return TRUE;

}

}

if ( (hFile=CreateFile (name,GENERIC\_READ,0,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL)) ==INVALID\_HANDLE\_VALUE)

MessageBox (0,"Не удается открыть файл с расписанием","error",MB\_APPLMODAL|MB\_OK|MB\_ICONSTOP);

filewasopened=FALSE;

return TRUE;

}

filewasopened=TRUE;

count=ReadShedule (hFile,sd);

if (count <=0)

{

MessageBox (hDlg,"Файл не соответствует формату",NULL,MB\_OK|MB\_ICONERROR);

return TRUE;

}

CloseHandle (hFile);

MakeItStandart (sd,count);

filewasopened=frst=FALSE;

}

nextexe=FindNext (sd,count);

if (nextexe<0)

{

SendMessage (hDlg,WM\_COMMAND, IDC\_STOP,0);

return TRUE;

}

delayexe=sd [nextexe]. delay;

sd [nextexe]. delay=-1;

SubtDelay (sd,count,delayexe);

SendMessage (hDlg,WM\_SIZE,SIZE\_MINIMIZED,0);

if (! SetTimer (hDlg,TM,1000\*delayexe,NULL))

{

frst=TRUE;

if (MessageBox (hDlg,"В системе нет свободных таймеров. ",0,MB\_RETRYCANCEL| MB\_ICONERROR) ==IDRETRY)

SendMessage (hDlg,WM\_COMMAND, IDC\_START,0);

else

SendMessage (hDlg,WM\_COMMAND, IDC\_STOP,0);

}

return TRUE;

}

case IDC\_STOP:

{

if (! frst)

{

KillTimer (hDlg,TM);

count =0;

nextexe=-1;

frst=TRUE;

MessageBox (hDlg,"в расписании больше нет задач","Завершение",MB\_OK|MB\_ICONINFORMATION);

}

return TRUE;

}

case IDC\_RESTORE:

{

Shell\_NotifyIcon (NIM\_DELETE,&nfd);

ShowWindow (hDlg,SW\_RESTORE);

SetForegroundWindow (hDlg);

return TRUE;

}

}

return TRUE;

}

return FALSE;

}

char \* FileOpen (HWND hWnd)

{

OPENFILENAME filename;

char name [512] = "";

ZeroMemory (&filename, sizeof (filename));

filename. lStructSize = sizeof (filename);

filename. hwndOwner = hWnd;

filename. lpstrFile = name;

filename. lpstrFile [0] = '\0';

filename. nMaxFile = sizeof (name);

filename. lpstrFilter = "Text files\0\*. txt\0";

filename. nFilterIndex = 1;

filename. lpstrFileTitle = "Выберите имя файла для сравнения... ";

filename. nMaxFileTitle = 0;

filename. lpstrInitialDir = NULL;

filename. Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

if (GetOpenFileName (&filename) ! = NULL)

return (filename. lpstrFile);

else

return 0;

}

int ReadShedule (HANDLE hFile,SHEDULEDATA sd [30])

{

char str [200] ="";

char symb;

char buff [2] ="";

int w;

BOOL IsComm;

int numb;

int i;

int j;

for (i=0; i<30; i++)

{

j=-1;

w=0;

while (1)

{

ReadFile (hFile,&symb,1,&numb,NULL);

if (numb==0)

return i+1;

if (symb=='\\')

symb='/';

if ( (j==-1) && (symb! ='"'))

return - 1;

else

if (j==-1)

{

IsComm=TRUE;

j++;

continue;

}

else

if (IsComm)

if (symb=='"')

{

IsComm=FALSE;

sd [i]. fname [j] =0;

j=0;

continue;

}

else

{

sd [i]. fname [j] =symb;

j++;

continue;

}

else

if ( (symb==0xd) || (symb==' ') || (symb==': '))

continue;

if ( (symb! =0xa))

{

if (j>5)

return - 1;

sd [i]. tname [j] =symb;

j++;

continue;

}

else

break;

}

}

return i+1;

}

void cstrcpy (char \*a,char \*b, int from, int count)

{

int i;

int j=0;

for (i=from; i<from+count; i++)

{

b [j] =a [i] ;

j++;

}

}

void MakeItStandart (SHEDULEDATA sd [30], int count)

{

int i;

SYSTEMTIME st;

char buffer [2] ;

int uis;

int uil;

int ui;

GetLocalTime (&st);

for (i=0; i<count; i++)

{

cstrcpy (sd [i]. tname,buffer,0,2);

sd [i]. h=atoi (buffer);

cstrcpy (sd [i]. tname,buffer,2,2);

sd [i]. m=atoi (buffer);

cstrcpy (sd [i]. tname,buffer,4,2);

sd [i]. s=atoi (buffer);

uil=sd [i]. h\*3600+sd [i]. m\*60+sd [i]. s;

uis=st. wHour\*3600+st. wMinute\*60+st. wSecond;

ui=uil-uis;

if (ui<0)

sd [i]. delay=ui+24\*3600;

else

sd [i]. delay=ui;

}

}

int FindNext (SHEDULEDATA sd [30], int count)

{

int i,min=0xFFFFFFF;

int minnumber=-1;

for (i=0; i<count; i++)

{

if (sd [i]. delay<0)

continue;

else

if (sd [i]. delay<min)

{

min=sd [i]. delay;

minnumber=i;

}

}

return minnumber;

}

void SubtDelay (SHEDULEDATA sd [30], int count, int delay)

{

int i;

for (i=0; i<count; i++)

if (sd [i]. delay>=0)

sd [i]. delay-=delay;

}