***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №2**

**По дисциплине ППвИС за IV семестр**

**Тема: «Вывод данных в оконных Windows-приложениях»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Муравьев Г.Л.

Брест 2019

Цель:

* Ознакомиться с особенностями организации вывода данных в оконных windows-приложениях
* Ознакомиться с особенностями управления сообщениями

Состав отчета:

1. Описание приложений
2. Листинги приложений
3. Диаграммы прецедентов, состояний

Ход работы:

1. Изучим теоретический материал.

Из параграфа §1.4.3 можно узнать о 3 функциях вывода: TextOut – вывод текста; MessageBox – окно сообщения (о котором знаем из предыдущей лабораторной работы); DrawText – вывод текста с усиленными возможностями редактирования.

1. Создадим приложение на базе ТКП, который мы изучили в лабораторной работе №1. Добавим строку приветствия с координатами 0, 0. Для этого вставим функцию TextOut в обработчик сообщения WM\_PAINT:

const int SIZE = 256;

char szGreeting[SIZE] = "Greeting!";

…

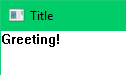
case WM\_PAINT:

…

TextOut(hdc, 0, 0, szGreeting, strlen(szGreeting));

…

Результат выполнения следующий:



Организуем вывод целого числа.

const int SIZE = 256;

char szInt[SIZE] = "";

…

case WM\_PAINT:

…

wsprintf(szInt, "Integer: %d", nValue);

TextOut(hdc, 0, 0, szInt, sizeof(szInt));

…



1. Модифицируем приложение, увеличив число обрабатываемых чисел до 75. Добавим вывод номера перерисовки (RedrowNumber).

static int RedrowNumber = 0;

const int SIZE = 100;

char szRedraw[SIZE] = "";

char szNumber[SIZE] = "";

char szSquere[SIZE] = "";

int i = 0;

…

RedrowNumber++;

wsprintf(szRedraw, "Redrawing # %d", RedrowNumber);

TextOut(hdc, 0, 0, szRedraw, sizeof(szRedraw));

for(i = 0; i < 75; i++)

{

wsprintf(szNumber, "Number - %d", i);

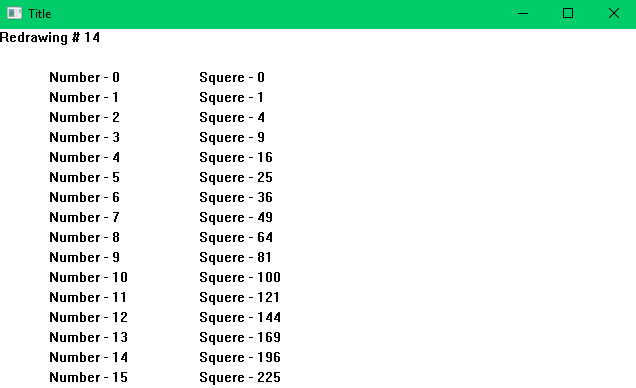
TextOut(hdc, 50, 40 + 20\*i, szNumber, sizeof(szNumber));

wsprintf(szSquere, "Squere - %d", i \* i);

TextOut(hdc, 200, 40 + 20\*i, szSquere, sizeof(szSquere));

}

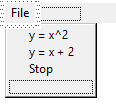
Выполняться приложение будет следующим образом:



Каждый раз, когда окно будет обновляться, число справа от “Redrawing #” будет инкрементироваться. Диаграммы изображены в приложении 1.

1. Создадим приложение для табулирования выбранной функции.

Добавим в наше приложение меню для выбора функции.



Как видно из иллюстрации, наше меню имеет 3 клавиши: 2 клавиши выбора функции (“y = x^2” – IDM\_X2; “y = x + 2” – IDM\_XP2) и клавишу, останавливающую выведение данной функции (“Stop” – IDM\_STOP).

Приложение будет работать в 2 режимах: режим выбора функции и режим вывода выбранной функции.

enum Mode

{

FUNC\_CHOOSING,

X2,

XP2

};

Для вывода табуляции функции будем использовать следующую функцию:

void OutputFunction(HDC hdc, Mode mode)

{

char szX[10] = "";

char szY[10] = "";

TextOut(hdc, 10, 10, "X", 1);

TextOut(hdc, 30, 10, "Y", 1);

for(int i = 0; i < 10; i++)

{

wsprintf(szX, "%d", i);

switch(mode)

{

case X2:

wsprintf(szY, "%d", i \* i);

break;

case XP2:

wsprintf(szY, "%d", 2 + i);

break;

default:

wsprintf(szY, "%d", i);

}

TextOut(hdc, 10, 30+20\*i, szX, 10);

TextOut(hdc, 30, 30+20\*i, szY, 10);

}

}

Функция обработчик сообщений окна примет вид:

LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

HDC hdc;

PAINTSTRUCT ps;

char szChoosingMsg[] = "Please choose function in File pop-menu";

static Mode mode = FUNC\_CHOOSING;

switch(msg)

{

case WM\_COMMAND:

switch(LOWORD(wParam))

{

case IDM\_X2:

if(mode==X2)

{

MessageBox(hWnd, "Function is already choosen", "Warning!", MB\_OK);

break;

}

mode = X2;

InvalidateRect(hWnd, NULL, TRUE);

break;

case IDM\_XP2:

if(mode==XP2)

{

MessageBox(hWnd, "Function is already choosen", "Warning!", MB\_OK);

break;

}

mode = XP2;

InvalidateRect(hWnd, NULL, TRUE);

break;

case IDM\_STOP:

if(mode==FUNC\_CHOOSING)

{

MessageBox(hWnd, "There are now function choosen", "Warning!",

MB\_OK);

break;

}

mode = FUNC\_CHOOSING;

InvalidateRect(hWnd, NULL, TRUE);

break;

}

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

if(mode == FUNC\_CHOOSING)

{

TextOut(hdc, 5, 5, szChoosingMsg, sizeof(szChoosingMsg));

}

else

{

OutputFunction(hdc, mode);

}

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

Диаграмму состояний данного приложения можно найти в Приложении 2.

Результаты выполнения приложения можно найти на иллюстрациях, приведенных ниже.

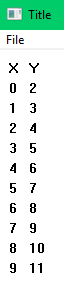
1. Режим выбора функции



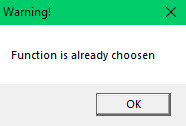
1. Режим функции y = x^2



1. Режим функции y = x + 2



b) При попытке переключиться на уже работающую функцию



1. Создадим приложение на базе ТКП. Включим чувствительность к нажатию левой клавиши «мыши». Для обработки нажатия клавиши мыши будем использовать обработку сообщения WM\_LBUTTONDOWN. Действия будут аналогичными с п. 4.

static int RedrowNumber = 0;

const int SIZE = 100;

char szRedraw[SIZE] = "";

char szNumber[SIZE] = "";

char szSquere[SIZE] = "";

int i = 0;

…

case WM\_LBUTTONDOWN:

hdc = BeginPaint(hWnd, &ps);

RedrowNumber++;

wsprintf(szRedraw, "Redrawing # %d", RedrowNumber);

TextOut(hdc, 0, 0, szRedraw, sizeof(szRedraw));

for(i = 0; i < 75; i++)

{

wsprintf(szNumber, "Number - %d", i);

TextOut(hdc, 50, 40 + 20\*i, szNumber, sizeof(szNumber));

wsprintf(szSquere, "Squere - %d", i \* i);

TextOut(hdc, 200, 40 + 20\*i, szSquere, sizeof(szSquere));

}

EndPaint(hWnd, &ps);

break;

Секцию же WM\_PAINT оставим пустой.

case WM\_PAINT:

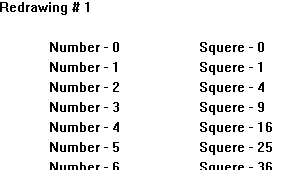
break;

Проведем анализ результатов выполнения приложения.

В начале выполнения приложения видим пустое окно. Это вызвано тем, что обработка вывода содержимого происходит в секции WM\_LBUTTONDOWN. Нажмем левую клавишу мыши.

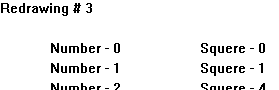


Результат, как видим, повторяет таковой в пункте 4. Попробуем нажать на клавишу ещё раз.



Нажатие клавиши не дало изменений. Попробуем вызвать функцию WM\_PAINT. Для этого изменим размер окна. Окно опустело. Нажмем ещё раз на клавишу мыши.

Как видно, результат действительно обновился до 3, именно столько раз мы нажимали на рабочую область окна, но перерисовка стала возможной только после вызова функции WM\_PAINT.



1. Создадим приложение для подсчета событий: нажатие левой клавиши мыши. Количество нажатий будем выводить в секции WM\_PAINT при нажатии правой клавиши мыши. (Приложение 3)

static int cLClks = 0;

char szCount[100] = "";

…

case WM\_LBUTTONDOWN:

cLClks++;

break;

case WM\_RBUTTONDOWN:

InvalidateRect(hWnd, NULL, TRUE);

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

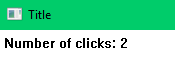
wsprintf(szCount, "Number of clicks: %d", cLClks);

TextOut(hdc, 5, 5, szCount, sizeof(szCount));

EndPaint(hWnd, &ps);

break;

Как и предполагалось, программа считает количество кликов левой клавишей мыши и выводит результат при нажатии правой.



Модифицируем приложение: результат будем выводить сразу в секции WM\_PAINT.

case WM\_LBUTTONDOWN:

cLClks++;

InvalidateRect(hWnd, NULL, TRUE);

break;

Функция InvalidateRect указывает системе, что некоторая область нуждается в обновлении, и система обновляет её когда в очереди сообщений не остается ни одного сообщения.

1. Создать приложение для вывода текста «Работает ТКП», начиная с позиций X, Y (сначала 0, 0). При нажатии левой кнопки мыши будем увеличивать координаты на 50 и инициировать перерисовку. При нажатии правой кнопки мыши будем уменьшать координаты на 50 и инициировать перерисовку (Приложение 4).

char szMsg[] = "Работает ТКП";

static int x = 0, y = 0

…

case WM\_LBUTTONDOWN:

x+=50; y+=50;

InvalidateRect(hWnd, NULL, TRUE);

break;

case WM\_RBUTTONDOWN:

x-=50; y-=50;

InvalidateRect(hWnd, NULL, TRUE);

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

TextOut(hdc, x, y, szMsg, sizeof(szMsg));

EndPaint(hWnd, &ps);

break;

Как и ожидалось, при нажатии кнопок мыши можно двигать содержимое пользовательской области.



1. Создадим приложение для вывода координат указателя «мыши». По нажатию левой кнопки мыши будем считывать текущее положение курсора мыши, а по нажатию правой – обновлять данные на рабочей области. Будем блокировать обновление вывода по другим причинам перерисовки (Приложение 5).

char szX[10] = ""; // Сообщение об X координате

char szY[10] = ""; // Сообщение об Y координате

static bool bRedraw = true; // Указывает, нужно ли производить обновление координат

static int x = 0, y = 0; // Координаты для вывода

static POINTS p; // Скрытые до нажатия правой кнопки мыши координаты

…

case WM\_LBUTTONDOWN:

p = MAKEPOINTS(lParam);

break;

case WM\_RBUTTONDOWN:

bRedraw = true;

InvalidateRect(hWnd, NULL, TRUE);

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

if(bRedraw) // Обновление координат только по нажатию правой кнопки мыши

{

x = p.x;

y = p.y;

bRedraw = false;

}

wsprintf(szX, "X=%d", x);

wsprintf(szY, "Y=%d", y);

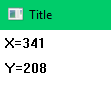
TextOut(hdc, 5, 5, szX, sizeof(szX));

TextOut(hdc, 5, 30, szY, sizeof(szY));

EndPaint(hWnd, &ps);

break;

Выполнение производится без ошибок:

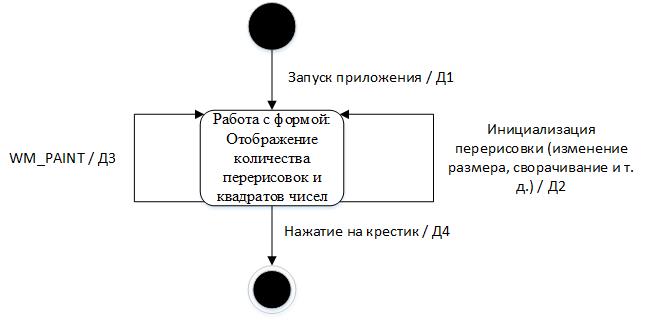


Вывод:

Принцип работы Windows-приложения целиком основан на обработке сообщений. «Общение» человека и компьютера производится при помощи средств ввода, в основном посредством мыши и клавиатуры. Производя манипуляции с назваными контроллерами, на компьютер поступает определенная информация. Эта информация обрабатывается Windows (если на компьютере установлена данная операционная система). Windows выделяет ту информацию, которая относится к нашей программе и отправляет сообщение, которое характеризует тот рад манипуляции, который пользователь сообщил нашей программе. Структура обработки сообщений нашей программы составляется таким образом: сообщения принимаются и отдаются соответствующему обработчику. Обработчик в свою очередь идентифицирует сообщения и производит оперирование информацией в соответствии и с типом сообщения. То, что именно за операции будут производиться по получении определенного сообщения, определяется самим разработчиком. Пример: выведение текста, сохранение данных, пришедших с сообщением, изменение данных в соответствии с сообщением и т. д.

Приложение 1

Диаграмма состояний и прецедентов (п. 3)

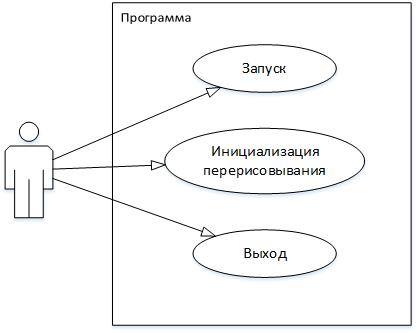


Д1: выделение ресурсов, создание окна, настройка обработчиков сообщений, инициализация счетчиков перерисовки (RedrawNumber = 0).

Д2: инкрементирование счетчика перерисовок, InvalidateRect(…)

Д3: выведение количества перерисовок и квадратов чисел

Д4: удаление окна, высвобождение ресурсов.



Прецедент: «Запуск».

Поток событий: выделение ресурсов, создание окна, настройка обработчика сообщений.

Прецедент: «Инициализация перерисовывания».

Поток событий: инкрементирование счетчика перерисовывания, отправка WM\_PAINT, в WM\_PAINT выведение количества перерисовывания.

Прецедент: «Выход».

Поток событий: удаление окна, высвобождение ресурсов.

Приложение 2

Диаграмма состояний (п. 4)



Д1: изменение мода на XP2, отправка сообщения WM\_PAINT, в обработчике WM\_PAINT выведение табуляции функции y = x + 2.

Д2: изменение мода на X2, отправка сообщения WM\_PAINT.

Д3: изменение мода на FUNC\_CHOOSING, отправка сообщения WM\_PAINT, в обработчике WM\_PAINT выведение сообщение с просьбой выбрать функцию.

Д4: закрытие окна – высвобождение ресурсов.

Приложение 3

Диаграмма состояний и прецедентов (п. 6)



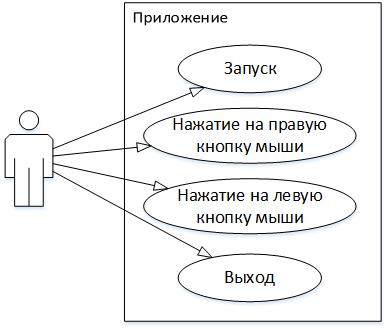
Д1: выделение ресурсов, создание окна, настройка обработчиков сообщений, инициализация счетчика нажатия левой клавиши мыши.

Д2: инкрементирование счетчика нажатия левой клавиши мыши.

Д3: отправка сообщения WM\_PAINT (InvalidateRect)

Д4 (обработчик WM\_PAINT): выводится значение счетчика нажатий левой клавиши мыши.

Д5: удаление окна, высвобождение ресурсов.



Прецедент: «Запуск».

Поток событий: выделение ресурсов, создание окна, настройка обработчика сообщений.

Прецедент: «Нажатие правой кнопки мыши».

Поток событий: отправка сообщения WM\_PAINT, в обработчике сообщения WM\_PAINT выводится значение счетчика нажатий левой клавиши мыши.

Прецедент: «Нажатие левой кнопки мыши»

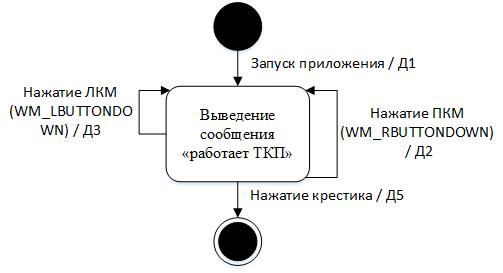
Поток событий: инкрементирование счетчика нажатия левой клавиши мыши

Прецедент: «Выход».

Поток событий: удаление окна, высвобождение ресурсов.

Приложение 4

Диаграмма состояний и прецедентов (п. 7)



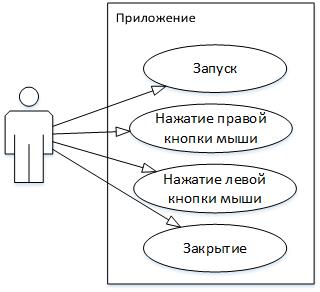
Д1: выделение ресурсов, создание окна, настройка обработчиков сообщений, задание начальных координат.

Д2: уменьшение координат текста на 50, InvalidateRect(…)

Д3: увеличение координат текста на 50; InvalidateRect(…)

Д4: выведение сообщения на соответствующих координатах

Д5: удаление окна, высвобождение ресурсов.



Прецедент: «Запуск».

Поток событий: выделение ресурсов, создание окна, настройка обработчика сообщений, задание начальных координат, выведение текста с начальными координатами.

Прецедент: «Нажатие правой кнопки мыши».

Поток событий: уменьшение координат текста на 50, отправка сообщения WM\_PAINT, в обработчике сообщения WM\_PAINT выводится сообщение на соответствующей позиции.

Прецедент: «Нажатие левой кнопки мыши»

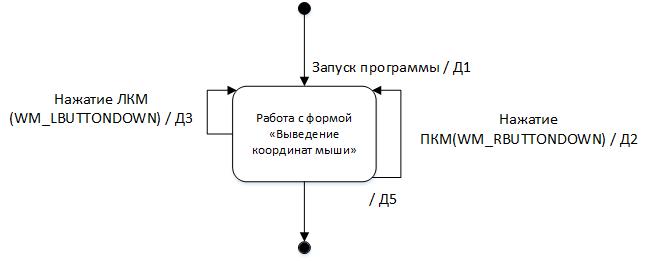
Поток событий: увеличение координат текста на 50отправка сообщения WM\_PAINT, в обработчике сообщения WM\_PAINT выводится сообщение на соответствующей позиции.

Прецедент: «Выход».

Поток событий: удаление окна, высвобождение ресурсов.

Приложение 5

Диаграмма состояний и прецедентов (п. 8)



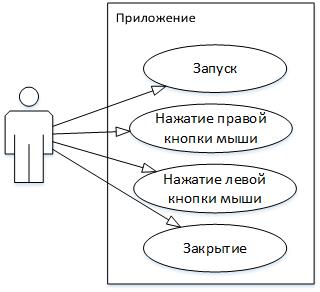
Д1: выделение ресурсов, создание окна, настройка обработчиков сообщений, инициализация значений координат (X = 0, Y = 0).

Д2 (обработчик WM\_RBUTTONDOWN): InvalidateRect(…)

Д3: считываются и сохраняются координаты стрелки мыши

Д4: выведение координат в клиентскую область TextOut()

Д5: удаление окна, высвобождение ресурсов.



Прецедент: «Запуск».

Поток событий: выделение ресурсов, создание окна, настройка обработчиков сообщений, инициализация координат начальными значениями.

Прецедент: «Нажатие правой кнопки мыши».

Поток событий: отправка сообщения WM\_PAINT, в обработчике сообщения WM\_PAINT выводится сообщение на соответствующей позиции.

Прецедент: «Нажатие левой кнопки мыши»

Поток событий: считываются и сохраняются координаты стрелки мыши

Прецедент: «Выход».

Поток событий: удаление окна, высвобождение ресурсов.