Босько Виктор, 151004, Лабораторная работа №3

**Source.cpp(lab3)**

#include <windows.h>

#include "Constant.h"

// static library

#include "TableDrawer.h"

#include "TextFormatter.h"

typedef void drawEllipticalText(HDC hdc, const wchar\_t\* text, POINT center, int vDiameter, int hDiameter, int letterHeight, int ind);

wchar\_t msg[] = L"orig";

// static linking from CircleText.dll

\_\_declspec(dllimport) BOOL WINAPI MessageCircle(HWND hWnd);

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

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

WNDCLASSEX wcex;

HWND hWnd;

MSG msg;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_DBLCLKS;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

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

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

wcex.hbrBackground = HBRUSH(CreateSolidBrush(ProjConst::WND\_MAIN\_INI\_COLOR));

wcex.lpszMenuName = NULL;

wcex.lpszClassName = ProjConst::WND\_MAIN\_CLASS\_NAME;

wcex.hIconSm = wcex.hIcon;

RegisterClassEx(&wcex);

hWnd = CreateWindow(ProjConst::WND\_MAIN\_CLASS\_NAME, ProjConst::WND\_MAIN\_CAPTION, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, ProjConst::WND\_INI\_WIDTH, ProjConst::WND\_INI\_HEIGHT,

NULL, NULL, hInstance, NULL);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

int pid = GetCurrentProcessId();

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

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

{

HDC hdc;

PAINTSTRUCT ps;

RECT wndRect;

SIZE letterSize;

LOGFONT font{};

HFONT hFont{};

HMODULE hModule = LoadLibrary(L"CircleText\_Dynamic.dll");

drawEllipticalText\* pDraw;

if (hModule != NULL) {

pDraw = (drawEllipticalText\*)GetProcAddress(hModule, "drawEllipticalText");

}

else {

pDraw = NULL;

}

GetWindowRect(hWnd, &wndRect);

int width = wndRect.right - wndRect.left;

FLOAT fontSize = -((FLOAT)width / ProjConst::WND\_INI\_WIDTH) \* ProjConst::FONT\_DEFAULT\_SIZE;

TableDrawer table(ProjConst::ROWS, ProjConst::COLUMNS, width, ProjConst::TEXT);

switch (message)

{

case WM\_SIZE:

InvalidateRect(hWnd, NULL, true);

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

hFont = CreateFont(fontSize, 0, 0, 0, 0, 0, 0, 0,

ANSI\_CHARSET, OUT\_DEVICE\_PRECIS, CLIP\_DEFAULT\_PRECIS, DEFAULT\_QUALITY, DEFAULT\_PITCH,

// ProjConst::FONT\_TYPE\_TERMINAL);

L"Times New Roman");

table.setHDC(hdc);

table.draw(hFont);

GetTextExtentPoint(hdc, &TextFormatter::ELLIPTICAL\_TEXT[0], 1, &letterSize);

if (pDraw != NULL)

{

pDraw(hdc, TextFormatter::ELLIPTICAL\_TEXT, { TextFormatter::COORD\_X,

TextFormatter::COORD\_Y }, TextFormatter::RADIUS\_V, TextFormatter::RADIUS\_H, letterSize.cy + 4, 0);

}

if (hModule != NULL) {

FreeLibrary(hModule);

}

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

MessageCircle(hWnd);

LoadLibrary(L"StringReplacer.dll");

MessageBox(hWnd, msg, msg, MB\_OK);

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

**TableDrawer.h(lab3)**

#pragma once

#include <windows.h>

#include <vector>

// Used to draw text in a table

class TableDrawer

{

public:

TableDrawer(int rows, int columns, int width, const LPCWSTR text[]);

void draw();

void draw(HFONT hFont);

void draw(LOGFONT font);

void setHDC(HDC hdc);

private:

const int TEXT\_FORMAT\_DRAW = DT\_LEFT | DT\_WORDBREAK | DT\_WORD\_ELLIPSIS;

const int TEXT\_FORMAT\_COUNT = DT\_LEFT | DT\_WORDBREAK | DT\_CALCRECT | DT\_WORD\_ELLIPSIS;

const int LEFT\_OFFSET = 3;

const int LETTER\_SIZE = 20 + LEFT\_OFFSET;

HDC hdc;

std::vector<RECT> cell;

std::vector<LPCWSTR> text;

int rows;

int columns;

int wndWidth;

int tableHeight;

int getCellInd(int row, int column);

int getColWidth();

int getRowHeight(int row);

void setRowHeight(int row, int rowHeight);

void drawCell(int row, int column);

void drawRow(int row, int rowHeight);

void drawBorders();

void createCells();

};

**TableDrawer.cpp(lab3)**

#include "TableDrawer.h"

TableDrawer::TableDrawer(int rows, int columns, int width, const LPCWSTR text[])

{

this->rows = rows;

this->columns = columns;

wndWidth = width - LETTER\_SIZE;

hdc = NULL;

for (int i = 0; i < rows \* columns; i++)

{

this->text.push\_back(text[i]);

}

tableHeight = 0;

}

// Draws table

void TableDrawer::draw()

{

createCells();

for (int row = 0; row < rows; row++)

{

int height = getRowHeight(row);

drawRow(row, height);

}

drawBorders();

}

// Draws table with given font

void TableDrawer::draw(HFONT hFont)

{

createCells();

SelectObject(hdc, hFont);

for (int row = 0; row < rows; row++)

{

int height = getRowHeight(row);

drawRow(row, height);

}

drawBorders();

}

// Draws table with given font

void TableDrawer::draw(LOGFONT font)

{

createCells();

HFONT hFont = CreateFontIndirect(&font);

SelectObject(hdc, hFont);

for (int row = 0; row < rows; row++)

{

int height = getRowHeight(row);

drawRow(row, height);

}

drawBorders();

}

// Fills cells with coordinates

void TableDrawer::createCells()

{

for (int i = 0; i < rows \* columns; i++)

{

int width = getColWidth();

RECT rect{ (i % columns) \* width, 0, (i % columns) \* width + width, 0 };

cell.push\_back(rect);

}

}

// Draws one row of the table

void TableDrawer::drawRow(int row, int rowHeight)

{

setRowHeight(row, rowHeight);

for (int column = 0; column < columns; column++)

{

drawCell(row, column);

}

for (int currCell = (row + 1) \* columns; currCell < rows \* columns; currCell++)

{

cell[currCell].top += rowHeight;

}

}

// Draws selected cell

void TableDrawer::drawCell(int row, int column)

{

int ind = getCellInd(row, column);

cell[ind].left += LEFT\_OFFSET;

cell[ind].right += LEFT\_OFFSET;

DrawText(hdc, text[ind], -1, &cell[ind], TEXT\_FORMAT\_DRAW);

}

// Draws border of the table

void TableDrawer::drawBorders()

{

HPEN pen = CreatePen(PS\_SOLID, 1, RGB(0, 0, 0));

int currRow = 0;

for (int column = 1; column < columns; column++)

{

MoveToEx(hdc, column \* getColWidth(), 0, NULL);

LineTo(hdc, column \* getColWidth(), tableHeight);

}

tableHeight = 0;

for (int row = 0; row < rows; row++)

{

currRow += getRowHeight(row);

MoveToEx(hdc, 0, currRow, NULL);

LineTo(hdc, wndWidth + LETTER\_SIZE, currRow);

}

}

// Sets the height of all cells in the row

void TableDrawer::setRowHeight(int row, int rowHeight)

{

for (int column = 0; column < columns; column++)

{

int ind = getCellInd(row, column);

cell[ind].bottom += rowHeight;

}

}

// Returns column width

int TableDrawer::getColWidth()

{

return wndWidth / columns;

}

// Returns row height

int TableDrawer::getRowHeight(int row)

{

int rowHeight = 0;

int currCellHeight = 0;

for (int column = 0; column < columns; column++)

{

int ind = getCellInd(row, column);

currCellHeight = DrawText(hdc, text[ind], -1, &cell[ind], TEXT\_FORMAT\_COUNT);

if (currCellHeight > rowHeight)

{

rowHeight = currCellHeight;

}

}

tableHeight += rowHeight;

return rowHeight;

}

// Returns cell as array index

int TableDrawer::getCellInd(int row, int column)

{

return row \* columns + column;

}

// Provides device context handle to draw

void TableDrawer::setHDC(HDC hdc)

{

this->hdc = hdc;

}

**CircleText.cpp**

#include "pch.h"

\_\_declspec(dllexport) BOOL WINAPI MessageCircle(HWND hWnd)

{

MessageBox(hWnd, L"Hello, World!", L"Message", MB\_OK);

return true;

}

**dllmain.cpp(CircleText\_dynamic)**

#include "pch.h"

const wchar\_t\* ELLIPTICAL\_TEXT = L"Test multiline text, many lines here!!! So many!!! SOOOOOO MANYYY";

const int COORD\_X = 500;

const int COORD\_Y = 500;

const int RADIUS\_V = 200;

const int RADIUS\_H = 300;

const double PI = 3.14;

// Counts max length of text that can be written in given perimeter

int getTextMaxLength(HDC hdc, int perimeter, const wchar\_t\* text, size\_t textLength, int ind)

{

SIZE letterSize;

int i;

for (i = ind; perimeter > 0 && i < textLength; i++)

{

GetTextExtentPoint(hdc, &text[i], 1, &letterSize);

perimeter -= letterSize.cx;

}

return i - ind;

}

// Counts ellipse perimeter as S = 2 \* PI sqrt((a^2 + b^2) / 8)

int getEllipsePerimeter(int a, int b)

{

return 2 \* PI \* sqrt((a \* a + b \* b) / 8);

}

// Draws formatted text. Elliptic text

extern "C" \_declspec(dllexport) void \_\_stdcall drawEllipticalText(HDC hdc, const wchar\_t\* text, POINT center, int vDiameter, int hDiameter, int letterHeight, int ind = 0)

{

XFORM xForm;

size\_t textLength = wcslen(text);

int perimeter = getEllipsePerimeter(vDiameter, hDiameter);

int maxLetters = getTextMaxLength(hdc, perimeter, text, textLength, ind);

double angleStep = 2 \* PI / (maxLetters);

SetGraphicsMode(hdc, GM\_ADVANCED);

for (int i = ind; i < maxLetters + ind && i < textLength; i++)

{

int x = static\_cast<int>(center.x + hDiameter \* cos(i \* angleStep - PI / 2));

int y = static\_cast<int>(center.y + vDiameter \* sin(i \* angleStep - PI / 2));

double rotationAngle = -(i \* angleStep);

xForm.eM11 = (FLOAT)cos(rotationAngle);

xForm.eM12 = (FLOAT)-sin(rotationAngle);

xForm.eM21 = (FLOAT)sin(rotationAngle);

xForm.eM22 = (FLOAT)cos(rotationAngle);

xForm.eDx = (FLOAT)x;

xForm.eDy = (FLOAT)y;

SetWorldTransform(hdc, &xForm);

TextOut(hdc, 0, 0, &text[i], 1);

ModifyWorldTransform(hdc, NULL, MWT\_IDENTITY);

}

if (ind < textLength)

{

drawEllipticalText(hdc, text, center, vDiameter - letterHeight, hDiameter - letterHeight, letterHeight, ind + maxLetters);

}

}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved)

{

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

**dllmain.cpp(StringReplacer)**

#include "pch.h"

#include "StringReplacer.h"

wchar\_t origin[] = L"orig";

wchar\_t dest[] = L"dest";

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved)

{

// Perform actions based on the reason for calling.

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

changeTextInProcces(origin, ARRAYSIZE(origin) - 1, dest);

// Initialize once for each new process.

// Return FALSE to fail DLL load.

break;

case DLL\_THREAD\_ATTACH:

// Do thread-specific initialization.

break;

case DLL\_THREAD\_DETACH:

// Do thread-specific cleanup.

break;

case DLL\_PROCESS\_DETACH:

// Perform any necessary cleanup.

break;

}

return TRUE; // Successful DLL\_PROCESS\_ATTACH.

}

**StringReplacer.cpp(StringReplacer)**

#include "framework.h"

#include "pch.h"

#include "StringReplacer.h"

#define MAX\_READ 128

int fMatchCheck(wchar\_t\* mainstr, int mainstrLen, wchar\_t\* checkstr, int checkstrLen) {

BOOL fmcret = TRUE;

int x, y;

for (x = 0; x < mainstrLen; x++) {

fmcret = TRUE;

for (y = 0; y < checkstrLen; y++) {

if (checkstr[y] != mainstr[x + y]) {

fmcret = FALSE;

break;

}

}

if (fmcret)

return x + checkstrLen;

}

return -1;

}

void changeTextInProcces(wchar\_t\* lpOrig, int origSize, wchar\_t\* lpReplacement) {

HANDLE hProc = GetCurrentProcess();

DWORD pid = \_getpid();

std::wostringstream str;

str << pid;

MessageBox(0, str.str().c\_str(), L"Process", MB\_OK);

MEMORY\_BASIC\_INFORMATION mbi;

SYSTEM\_INFO msi;

ZeroMemory(&mbi, sizeof(mbi));

GetSystemInfo(&msi);

SIZE\_T dwRead = 0;

wchar\_t\* lpData = (wchar\_t\*)GlobalAlloc(GMEM\_FIXED, MAX\_READ);

for (LPBYTE lpAddress = (LPBYTE)msi.lpMinimumApplicationAddress; lpAddress <= (LPBYTE)msi.lpMaximumApplicationAddress; lpAddress += mbi.RegionSize) {

if (VirtualQuery(lpAddress, &mbi, sizeof(mbi))) {

if ((mbi.Protect & PAGE\_READWRITE) || (mbi.Protect & PAGE\_WRITECOPY)) {

for (LPBYTE tempLpAddress = lpAddress; tempLpAddress < (lpAddress + mbi.RegionSize); tempLpAddress += (MAX\_READ - origSize \* sizeof(wchar\_t))) {

if (ReadProcessMemory(hProc,

(LPCVOID)tempLpAddress,

lpData,

MAX\_READ, &

dwRead) == TRUE) {

LPBYTE sometemp = tempLpAddress;

if (fMatchCheck(lpData, dwRead / sizeof(wchar\_t), lpOrig, origSize) != -1) {

wchar\_t tempChar = lpOrig[0];

wchar\_t cmpChar = sometemp[0];

while ((sometemp < (tempLpAddress + dwRead)) && (cmpChar != tempChar)) {

sometemp++;

cmpChar = sometemp[0];

}

if ((LPCVOID)sometemp == lpOrig) {

continue;

}

int test = WriteProcessMemory(hProc,

(LPVOID)sometemp,

lpReplacement,

origSize \* sizeof(wchar\_t), &

dwRead);

test = dwRead;

}

}

}

}

}

}

if (lpData)

GlobalFree(lpData);

MessageBox(0, L"Replaced", L"DLL", MB\_OK);

}

**Injector.cpp**

#include <Windows.h>

#include <iostream>

#define DLL\_NAME "D:\BSUIR\SystemProgramming\lab3\lab3\x64\Debug\StringReplacer.dll"

BOOL InjectDll(int pid);

int main()

{

int pid;

printf("Enter the process id:\n");

scanf\_s("%d", &pid);

InjectDll(pid);

getchar();

getchar();

return 0;

}

BOOL InjectDll(int pid)

{

HANDLE process = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, pid);

if (process == NULL)

{

std::cout << "Error: Cannot open the process.\n";

return false;

}

LPVOID funcPointer = (LPVOID)GetProcAddress(GetModuleHandle(TEXT("kernel32.dll")), "LoadLibraryA");

if (funcPointer == NULL)

{

return false;

}

LPVOID alloc = VirtualAllocEx(process, 0, strlen(DLL\_NAME), MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

if (alloc == NULL)

{

return false;

}

if (!WriteProcessMemory(process, alloc, DLL\_NAME, strlen(DLL\_NAME), NULL))

{

return false;

}

HANDLE remoteThread = CreateRemoteThread(process, 0, 0, (LPTHREAD\_START\_ROUTINE)funcPointer, (LPVOID)alloc, 0, 0);

if (remoteThread == NULL)

{

return false;

}

std::cout << "Injected\n";

CloseHandle(remoteThread);

return true;

}