Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №3

на тему

Управление памятью и вводом-выводом, расширенные возможности ввода-вывода Windows. Функции API подсистемы памяти Win 32. Организация и контроль асинхронных операций ввода-вывода. Отображение файлов в память

Студент: гр.153502

Макаренко А.И.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495008)

[2 Теоретические сведения 4](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495009)

[3 Описание функций программы 5](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495010)

[Список использованных источников 7](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495011)

[Приложение А 8](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495012)

1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание приложения на *Win32 API*, которое управляет памятью и вводом-выводом, использует функции *API* подсистемы памяти *Win32*, организует и контролирует асинхронные операции ввода-вывода.

В качестве задачи необходимо реализовать приложение, которое визуализирует структуру файловой системы и отслеживает изменения в реальном времени.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

*Tree-View* – это окно, в котором отображается иерархический список элементов. Каждый элемент может иметь список связанных с ним дочерних элементов. Щёлкнув на элемент, пользователь может развернуть или свернуть связанный список таких элементов [1].

Данный элемент особенно удобен при визуализации структуры файловой системы, так как имеет флаги, позволяющие добавлять кнопки для раскрытия элементов и показывающие линии для более лёгкого ориентирования в потомках и родителях [2].

Функция *ReadDirectoryChanges* извлекает информацию, описывающую изменения в указанном каталоге, однако не отслеживает их. Функция содержит несколько параметров таких как:

1 *hDirectory* – дескриптор каталога, который необходимо отслеживать;

2 *lpBuffer* – указатель на буфер;

3 *nBufferLength* – размер буфера в байтах;

4 *bWatchSubtree* – параметр, указывающий, нужно ли отслеживать дерево каталогов, расположенное в указанном каталоге;

5 *dwNotifyFilter* – критерии фильтра для проверки завершения операции ожидания;

6 *lpBytesReturned* – количество байтов, переданных в буфер;

7 *lpOverlapped* – указатель на структуру *OVERLAPPED*, предоставляющая данные для использования во время асинхронной операции;

8 *lpCompletionRoutine* – указатель на процедуру завершения, которая будет вызываться, когда операция завершена или отменена.

Если функция завершается успешно, возвращаемое значение не равно нулю [3].

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– отображение структуры файловой системы;

– логирование изменений;

– реализация быстрого отклика на изменения.

При запуске программа отображает корневую папку (по умолчанию диск *C*). Чтобы просмотреть содержимое папки необходимо нажать на значок рядом с её названием (рисунок 1).

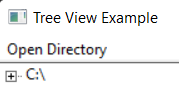


Рисунок 1 – Значок для того, чтобы открыть папку

После клика значок меняется. Чтобы закрыть папку необходимо нажать на изменившийся значок (рисунок 2). Папки отличаются от обычных файлов значками рядом с их названиями. Отличие показано на рисунке 3.

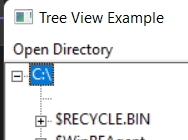


Рисунок 2 – Значок для того, чтобы закрыть папку



Рисунок 3 – Сравнение файла и папки

Для смены корневой папки необходимо нажать на кнопку *Open* *Directory* (рисунок 4), после чего откроется диалог для выбора папки (рисунок 5).



Рисунок 4 – Кнопка *Open* *Directory*

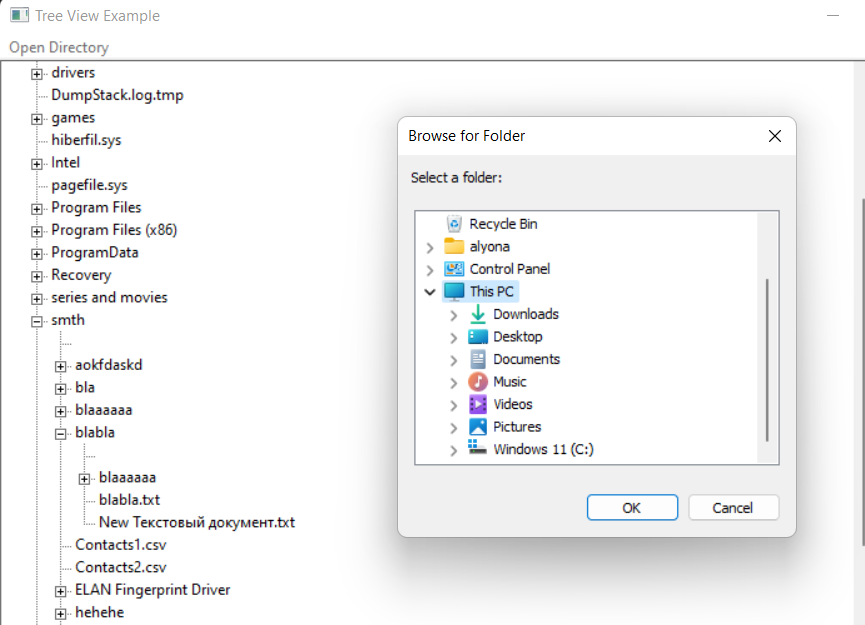


Рисунок 5 – Диалоговое окно выбора корневой директории

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] About Tree-View Controls [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/controls/tree-view-controls>.

[2] Using TreeControl (TreeView) under Win32 API [Электронный ресурс]. – Режим доступа: <https://www.codeproject.com/Articles/3448/Using-TreeControl-TreeView-under-Win32-API>.

[3] Функция ReadDirectoryChangesW [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-readdirectorychangesw>.

ПРИЛОЖЕНИЕ А

**Исходный код программы**

**Файл main.cpp**

#include <windows.h>

#include <CommCtrl.h>

#include <string>

#include <vector>

#include <shlobj.h> // for folder browsing

#pragma comment(lib, "comctl32.lib")

#define ID\_TREE\_VIEW 551

#define IDM\_OPEN\_FOLDER 554

HWND hTreeView;

HWND hMainWindow;

struct TreeViewItem {

std::wstring fullPath;

std::wstring shortPath;

HANDLE thread;

HTREEITEM currentItem;

};

HANDLE hMutex;

std::vector<std::wstring> fileChanges;

HTREEITEM AddItemToTreeView(HWND hTreeView, HTREEITEM hParent, LPCWSTR text)

{

TVINSERTSTRUCT tvInsert;

tvInsert.hParent = hParent;

tvInsert.hInsertAfter = TVI\_LAST;

tvInsert.item.mask = TVIF\_TEXT;

tvInsert.item.pszText = (LPWSTR)text;

return TreeView\_InsertItem(hTreeView, &tvInsert);

}

HTREEITEM FindChildItemByName(HTREEITEM hParent, LPCWSTR targetName) {

HTREEITEM hChild = TreeView\_GetChild(hTreeView, hParent);

while (hChild) {

TVITEM tvi;

ZeroMemory(&tvi, sizeof(TVITEM));

tvi.mask = TVIF\_TEXT;

tvi.hItem = hChild;

tvi.pszText = new WCHAR[MAX\_PATH]; // allocating memory for item text

tvi.cchTextMax = MAX\_PATH;

TreeView\_GetItem(hTreeView, &tvi); // getting the text of the current item

if (wcscmp(tvi.pszText, targetName) == 0) { // comparing item's text to the target name

delete[] tvi.pszText;

return hChild;

}

delete[] tvi.pszText;

hChild = TreeView\_GetNextSibling(hTreeView, hChild);

}

return NULL; // Item not found

}

DWORD WINAPI MonitorDirectoryChanges(LPVOID lparam)

{

auto item = (TreeViewItem\*)lparam;

HANDLE hDir = CreateFile( // open dir

item->fullPath.c\_str(),

FILE\_LIST\_DIRECTORY, // grants to read a list of content

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE | FILE\_SHARE\_DELETE, // file can be shared while program is using it

NULL, // security attributes

OPEN\_EXISTING, // creation options

FILE\_FLAG\_BACKUP\_SEMANTICS, // this flag is to obtain handle to a dir

NULL); // template

if (hDir == INVALID\_HANDLE\_VALUE) {

return 1;

}

BYTE buffer[4096]; // regular size

DWORD bytesReturned;

while (true) {

if (ReadDirectoryChangesW(

hDir,

buffer,

sizeof(buffer),

FALSE, // false means function monitors only changes within this dir

FILE\_NOTIFY\_CHANGE\_FILE\_NAME | FILE\_NOTIFY\_CHANGE\_DIR\_NAME,

&bytesReturned, // amount of bytes transferred to buffer

NULL, // indicates, that obtained data is not going to be used asynchronousely

NULL)) { // pointer to completion routine

FILE\_NOTIFY\_INFORMATION\* fni = (FILE\_NOTIFY\_INFORMATION\*)buffer; // fni is a change

int offset = -1; // initialization of variables before the cycle

std::wstring prevName = L"";

std::wstring log = L"error";

do {

std::wstring filename = std::wstring(fni->FileName, fni->FileNameLength / 2);

offset = fni->NextEntryOffset; // the number of bytes that must be skipped to get to the next record

if (fni->Action == FILE\_ACTION\_ADDED) {

WIN32\_FIND\_DATA findFileData;

HANDLE hFind = FindFirstFile((item->fullPath + L"\\" + filename).c\_str(), &findFileData);

if (hFind != INVALID\_HANDLE\_VALUE) {

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY) { // for dirs

HTREEITEM hItem = AddItemToTreeView(hTreeView, item->currentItem, findFileData.cFileName);

// associate the full path with the tree item as item data

TreeViewItem\* lparam = new TreeViewItem{ item->fullPath + L"\\" + findFileData.cFileName, std::wstring(findFileData.cFileName), NULL, hItem };

TVITEMEX tvItem;

tvItem.hItem = hItem;

tvItem.mask = TVIF\_PARAM;

tvItem.lParam = (LPARAM)lparam;

TreeView\_SetItem(hTreeView, &tvItem);

AddItemToTreeView(hTreeView, hItem, L""); // dummy item

TreeView\_Expand(hTreeView, hItem, TVE\_EXPAND);

}

else { // for regular files

AddItemToTreeView(hTreeView, item->currentItem, findFileData.cFileName);

}

}

log = L"new object created, filename: " + filename;

}

if (fni->Action == FILE\_ACTION\_REMOVED) {

auto fullname = item->fullPath + L"\\" + filename;

auto itemToDelete = FindChildItemByName(item->currentItem, filename.c\_str());

if (itemToDelete) {

TreeView\_DeleteItem(hTreeView, itemToDelete);

}

log = L"object removed, filename: " + filename;

}

if (fni->Action == FILE\_ACTION\_RENAMED\_OLD\_NAME) {

prevName = filename; // saving old name to change it in program

}

if (fni->Action == FILE\_ACTION\_RENAMED\_NEW\_NAME) {

auto prevFullname = item->fullPath + L"\\" + prevName;

auto fullname = item->fullPath + L"\\" + filename;

auto itemToRename = FindChildItemByName(item->currentItem, prevName.c\_str());

if (itemToRename) {

TVITEMEX tvItem;

tvItem.hItem = itemToRename;

tvItem.mask = TVIF\_PARAM;

if (TreeView\_GetItem(hTreeView, &tvItem)) { // for dirs

TreeViewItem\* lparam = (TreeViewItem\*)tvItem.lParam;

if (lparam) {

lparam->fullPath = fullname;

lparam->shortPath = filename;

}

}

{

TVITEMEX tvItem;

tvItem.hItem = itemToRename;

tvItem.mask = TVIF\_TEXT;

tvItem.pszText = (LPWSTR)filename.c\_str();

TreeView\_SetItem(hTreeView, &tvItem);

}

}

log = L"object renamed, old filename: " + prevName + L", new filename: " + filename;

}

fni = (FILE\_NOTIFY\_INFORMATION\*)((char\*)fni + offset);

} while (offset != 0); // cycle ends when no other changes were found

DWORD dwWaitResult = WaitForSingleObject(hMutex, INFINITE); // waiting for signal

if (dwWaitResult == WAIT\_OBJECT\_0) { // if signaled

fileChanges.push\_back(log);

ReleaseMutex(hMutex);

}

}

else {

break;

}

Sleep(50);

}

CloseHandle(hDir);

return 0;

}

// populating the tree-view with directories and files

void PopulateTreeView(HTREEITEM hParent, const std::wstring& parentPath)

{

WIN32\_FIND\_DATA findFileData;

HANDLE hFind = FindFirstFile((parentPath + L"\\\*").c\_str(), &findFileData);

if (hFind != INVALID\_HANDLE\_VALUE) {

do {

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0) {

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY) { // for dirs

HTREEITEM hItem = AddItemToTreeView(hTreeView, hParent, findFileData.cFileName);

// associate the full path with the tree item as item data

TreeViewItem\* lparam = new TreeViewItem{ parentPath + L"\\" + findFileData.cFileName, std::wstring(findFileData.cFileName), NULL, hItem };

TVITEMEX tvItem;

tvItem.hItem = hItem;

tvItem.mask = TVIF\_PARAM;

tvItem.lParam = (LPARAM)lparam;

TreeView\_SetItem(hTreeView, &tvItem);

AddItemToTreeView(hTreeView, hItem, L""); // dummy item

}

else { // for other files

AddItemToTreeView(hTreeView, hParent, findFileData.cFileName);

}

}

} while (FindNextFile(hFind, &findFileData) != 0);

FindClose(hFind);

}

}

void InitTreeView(const std::wstring& dir)

{

auto root = AddItemToTreeView(hTreeView, NULL, dir.c\_str());

// associate the full path with the tree item as item data

TreeViewItem\* lparam = new TreeViewItem{ dir, dir, NULL, root };

TVITEMEX tvItem;

tvItem.hItem = root;

tvItem.mask = TVIF\_PARAM;

tvItem.lParam = (LPARAM)lparam;

TreeView\_SetItem(hTreeView, &tvItem);

AddItemToTreeView(hTreeView, root, L""); // add a dummy item

}

void OnCreate(HWND hWnd)

{

hTreeView = CreateWindowEx(0, WC\_TREEVIEW, L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | TVS\_HASLINES | TVS\_LINESATROOT | TVS\_HASBUTTONS | TVS\_INFOTIP, 0, 0, 700, 500, hWnd, (HMENU)ID\_TREE\_VIEW, NULL, NULL);

InitTreeView(L"C:\\");

// infotip - for notifications

}

void OnNotify(LPARAM lParam)

{

LPNMHDR pnmh = (LPNMHDR)lParam; //handle for tree-view notifications

if (pnmh->idFrom == ID\_TREE\_VIEW && pnmh->code == TVN\_ITEMEXPANDED) { //itemexpanded notifies when a parent item's list of child items

//is about to expand or collapse

LPNMTREEVIEW pNMTV = (LPNMTREEVIEW)lParam;

HTREEITEM hItem = pNMTV->itemNew.hItem;

TVITEMEX tvItem;

tvItem.hItem = hItem;

tvItem.mask = TVIF\_PARAM | TVIF\_STATE;

tvItem.stateMask = TVIS\_EXPANDED;

if (TreeView\_GetItem(hTreeView, &tvItem)) { // retrieves all of a tree-view item's attributes

// checks if the item data contains a full path (which means it's a directory)

if (tvItem.state & TVIS\_EXPANDED) {

TreeViewItem\* lparam = (TreeViewItem\*)tvItem.lParam;

if (lparam != nullptr) {

std::wstring fullPath = lparam->fullPath;

/\*HTREEITEM hChild = TreeView\_GetChild(hTreeView, hItem);

if (hChild != NULL) {

TreeView\_DeleteItem(hTreeView, hChild);

}\*/

lparam->thread = CreateThread(NULL, NULL, MonitorDirectoryChanges, lparam, NULL, NULL);

// pointer to security attributes (whether to allow to inherit handle or not), stack size (null means default), pointer to executed func,

//pointer to a variable passed to thread, flags (null - runs immideately after creation), pointer to a variable that receives the thread id (null means it's not returned)

PopulateTreeView(hItem, fullPath);

}

}

else { // not expanded (changed)

HTREEITEM hParent = TreeView\_GetParent(hTreeView, hItem);

LPARAM lparam = tvItem.lParam;

TreeView\_DeleteItem(hTreeView, hItem);

HTREEITEM hNewItem = AddItemToTreeView(hTreeView, hParent, ((TreeViewItem\*)lparam)->shortPath.c\_str());

TVITEMEX tvNewItem;

tvNewItem.hItem = hNewItem;

tvNewItem.mask = TVIF\_PARAM;

tvNewItem.lParam = lparam;

TreeView\_SetItem(hTreeView, &tvNewItem); // sets attributes from tvNewItem

AddItemToTreeView(hTreeView, hNewItem, L"");

TreeView\_SortChildren(hTreeView, hParent, FALSE); // false - for recursion

}

}

}

if (pnmh->idFrom == ID\_TREE\_VIEW && pnmh->code == TVN\_DELETEITEM) {

LPNMTREEVIEW pNMTV = (LPNMTREEVIEW)lParam;

HTREEITEM hItem = pNMTV->itemOld.hItem;

TVITEMEX tvItem;

tvItem.hItem = hItem;

tvItem.mask = TVIF\_PARAM;

if (TreeView\_GetItem(hTreeView, &tvItem)) {

auto data = (TreeViewItem\*)tvItem.lParam;

if (data != nullptr && data->thread != nullptr) {

TerminateThread(data->thread, 0);

}

}

}

}

void OpenFolderDialog()

{

BROWSEINFO bi = { 0 };

bi.lpszTitle = L"Select a folder:";

LPITEMIDLIST pidl = SHBrowseForFolder(&bi);

if (pidl != NULL) {

wchar\_t selectedPath[MAX\_PATH];

SHGetPathFromIDList(pidl, selectedPath);

TreeView\_DeleteAllItems(hTreeView);

InitTreeView(selectedPath);

CoTaskMemFree(pidl); // free mem

}

}

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

{

switch (msg) {

case WM\_CREATE:

OnCreate(hWnd);

break;

case WM\_NOTIFY: {

OnNotify(lParam);

break;

}

case WM\_COMMAND:

if (LOWORD(wParam) == IDM\_OPEN\_FOLDER) {

OpenFolderDialog();

}

break;

case WM\_DESTROY:

if (fileChanges.size() > 0) {

std::wstring fullLog;

for (auto& log : fileChanges) {

fullLog += log + L"\n";

}

MessageBox(hMainWindow, fullLog.c\_str(), L"file changes", MB\_OK);

}

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

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

{

hMutex = CreateMutex(NULL, FALSE, NULL);

if (hMutex == NULL) {

return 1;

}

WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS\_CLASSDC, WndProc, 0L, 0L, GetModuleHandle(NULL), NULL, NULL, NULL, NULL, L"TreeView", NULL };

//size, style, pointer to window procedure, extra mem for struct, extra mem for instance, handle to instance that contains window procedure, icon, cursor,

//brush, pointer to class menu, pointer to str, small icon

RegisterClassEx(&wc);

hMainWindow = CreateWindow(wc.lpszClassName, L"TreeView", WS\_OVERLAPPED | WS\_MINIMIZEBOX | WS\_SYSMENU, 100, 100, 800, 600, GetDesktopWindow(), NULL, wc.hInstance, NULL);

ShowWindow(hMainWindow, nCmdShow);

HMENU hMenu = CreateMenu();

AppendMenu(hMenu, MF\_STRING, IDM\_OPEN\_FOLDER, L"Open Directory");

SetMenu(hMainWindow, hMenu);

MSG msg;

ZeroMemory(&msg, sizeof(msg));

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnregisterClass(wc.lpszClassName, wc.hInstance);

CloseHandle(hMutex);

return 0;

}