МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Програмної інженерії

Звіт
з лабораторної роботи № 4
з дисципліни: «Операційні системи»
з теми: «Керування зовнішніми пристроями. Нестандартні пристрої»

Виконали:
ст. гр. ПЗПІ-23-2
Ситник Є. С.
Малишкін А. С.

Перевірила:
доц. каф. ПІ
Мельнікова Р. В.

Харків – 2025

4 КЕРУВАННЯ ЗОВНІШНІМИ ПРИСТРОЯМИ. НЕСТАНДАРТНІ ПРИСТРОЇ

4.1 Мета роботи

Навчитися практичному використанню функцій WinAPI для роботи з файлами.

4.2 Хід роботи

4.2.1 Створення класу для роботи з пристроями, файлами та каталогами.

4.2.1.1 Вивчити усі функції, які використовуються для роботи з пристроями, файлами та каталогами в C#.

В C# класи для роботи з пристроями, файлами та каталогами знаходяться в просторі імен «System.IO». Такі класи як «File», «FileStream», «Directory» та «DriveInfo» надають основні засоби для взаємодії з файловою системою.

Клас «File» пропонує статичні методи для виконання базових операцій над файлами. До них належать: створення нового файлу («File.Create»), видалення існуючого файлу («File.Delete»), перевірка існування файлу («File.Exists»), копіювання файлу («File.Copy»), переміщення файлу («File.Move»), та інші.

Клас «FileStream» надає базову функціональність для роботи з файлами як з потоками даних. Його основні функції включають відкриття файлу для читання або запису (через конструктори або методи «Open», «OpenRead», «OpenWrite»), читання даних з файлу («Read»), запис даних у файл («Write»), закриття файлового потоку («Close» або використання блоку «using» для автоматичного закриття), а також керування поточною позицією у файлі («Seek»).

Клас «Directory» надає статичні методи для виконання базових операцій над директоріями. Серед них: створення нової директорії («Directory.CreateDirectory»), видалення існуючої директорії («Directory.Delete»), перевірка існування директорії («Directory.Exists»), переміщення директорії («Directory.Move»), а також отримання списку файлів у директорії («Directory.GetFiles») та списку піддиректорій («Directory.GetDirectories»).

Клас «DriveInfo» надає інформацію про логічні диски, підключені до комп’ютера, для отримання інформації про конкретний диск необхідно створити його екземпляр, передавши ім’я диску в конструктор, а серед основних

властивостей можна знайти ім’я диску («Name»), його тип («DriveType»), стан готовності («IsReady»), загальний розмір («TotalSize»), доступний вільний простір («AvailableFreeSpace»), формат файлової системи («DriveFormat»), мітку тому («VolumeLabel») та об’єкт кореневого каталогу («RootDirectory»), а для отримання інформації про всі диски в системі використовується статичний метод «GetDrives».

4.2.1.2 Визначити клас (класи) для мови C++, інтерфейс для яких подібний інтерфейсу C#.

Створимо на C++ класи «File», «FileStream», «Directory» та «DriveInfo». Для кожного з них створимо власний файл заголовків.

```cpp
#pragma once
#include <cstdint>
#include <string>

#include "IFileSystemEntity.h"

namespace System::IO {
class File : public IFileSystemEntity {
private:
  std::wstring path;

public:
  explicit File(const std::wstring &filePath);
  ~File() override = default;

  std::wstring GetPath() const override;
  DWORD GetAttributes() const override;
  bool SetAttributes(DWORD attributes) override;

  static bool Exists(const std::wstring &path);
  static bool Copy(const std::wstring &source, const std::wstring &destination, bool overwrite = false);
  static bool Delete(const std::wstring &path);
  static bool Move(const std::wstring &source, const std::wstring &destination);

  static uint64_t GetSize(const std::wstring &path);
};
} // namespace System::IO
```

Рисунок 4.1 – Вміст файлу «File.h»

```cpp
#pragma once
#include <string>
#include <windows.h>

namespace System::IO {
class FileStream {
private:
  HANDLE hFile;
  std::wstring path;

public:
  FileStream(const std::wstring &filePath, DWORD access,
             DWORD creationDisposition, DWORD shareMode = 0,
             DWORD flagsAndAttributes = FILE_ATTRIBUTE_NORMAL);
  ~FileStream();

  bool Read(void *buffer, DWORD numBytesToRead, DWORD &numBytesRead);
  bool Write(const void *buffer, DWORD numBytesToWrite, DWORD &numBytesWritten);
  bool Seek(LARGE_INTEGER offset, DWORD moveMethod, ULARGE_INTEGER &newPos);
  bool SetEnd();

  HANDLE GetHandle() const { return hFile; }
  std::wstring GetPath() const { return path; }
};
} // namespace System::IO
```

Рисунок 4.2 – Вміст файлу «FileStream.h»

```cpp
#pragma once
#include <string>
#include <vector>

#include "IFileSystemEntity.h"

namespace System::IO {
class Directory : public IFileSystemEntity {
private:
  std::wstring path;

public:
  explicit Directory(const std::wstring &directoryPath);
  ~Directory() override = default;

  std::wstring GetPath() const override;
  DWORD GetAttributes() const override;
  bool SetAttributes(DWORD attributes) override;

  static bool Exists(const std::wstring &path);
  static bool Create(const std::wstring &path);
  static bool Delete(const std::wstring &path);
  static bool Move(const std::wstring &source, const std::wstring &destination);

  static std::vector<std::wstring> GetFiles(const std::wstring &path);
  static std::vector<std::wstring> GetDirectories(const std::wstring &path);
};
} // namespace System::IO
```

Рисунок 4.3 – Вміст файлу «Directory.h»

```cpp
#pragma once
#include <string>
#include <vector>
#include <cstdint>

namespace System::IO {
    enum class DriveType : unsigned int { ↙ 8

    class DriveInfo {
    public:
        explicit DriveInfo(const std::wstring& path);

        std::wstring Name;
        DriveType DriveType;
        std::wstring DriveFormat;
        uint64_t TotalSize;
        uint64_t AvailableFreeSpace;
        bool IsReady;

        static std::vector<DriveInfo> GetDrives();

    private:
        std::wstring rootPath;
    };
}
```

Рисунок 4.4 – Вміст файлу «DriveInfo.h»

4.2.1.3 Реалізувати функції класу (класів) за допомогою функцій WinAPI.

Оскільки WinAPI надає функції, майже ідентичні необхідним, більшість з реалізацій є лише обгортками навколо відповідних функцій WinAPI.

Наприклад функція «SetAttributes» класу «File» має наступний вигляд:

```cpp
bool File::SetAttributes(DWORD attributes) {
  return SetFileAttributesW(lpFileName: path.c_str(), dwFileAttributes: attributes) ≠ 0;
}
```

Рисунок 4.5 – Реалізація функції «SetAttributes» класу «File»

Таким чином реалізовано більшість методів.

Із методів, що не мають точних аналогів у WinAPI можна виділити такі.

```cpp
uint64_t File::GetSize(const std::wstring &path) {
  HANDLE hFile = CreateFileW(
      path.c_str(),
      GENERIC_READ,
      FILE_SHARE_READ,
      nullptr,
      OPEN_EXISTING,
      FILE_ATTRIBUTE_NORMAL,
      nullptr
  );

  if (hFile == INVALID_HANDLE_VALUE) {
    throw std::runtime_error("Unable to open file to get its size.");
  }

  LARGE_INTEGER size;
  if (!GetFileSizeEx(hFile, &size)) {
    CloseHandle(hFile);
    throw std::runtime_error("Unable to get file size.");
  }

  CloseHandle(hFile);
  return static_cast<uint64_t>(size.QuadPart);
}
```

Рисунок 4.6 – Реалізація функції «GetSize» класу «File»

```cpp
std::vector<std::wstring> Directory::GetDirectories(const std::wstring &path) {
  std::vector<std::wstring> directories;

  std::wstring searchPath = path + L"\\*";
  WIN32_FIND_DATAW findData;
  HANDLE hFind = FindFirstFileW(searchPath.c_str(), &findData);

  if (hFind == INVALID_HANDLE_VALUE)
    return directories;

  do {
    if (findData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
      std::wstring dirName = findData.cFileName;
      if (dirName != L"." && dirName != L"..")
        directories.push_back(dirName);
    }
  } while (FindNextFileW(hFind, &findData) != 0);

  FindClose(hFind);
  return directories;
}
```

Рисунок 4.7 – Реалізація функції «GetDirectories» класу «Directory»

```cpp
std::vector<std::wstring> Directory::GetFiles(const std::wstring &path) {
  std::vector<std::wstring> files;

  std::wstring searchPath = path + L"\\*";
  WIN32_FIND_DATAW findData;
  HANDLE hFind = FindFirstFileW(searchPath.c_str(), &findData);

  if (hFind == INVALID_HANDLE_VALUE)
    return files;

  do {
    if (!(findData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
      files.push_back(findData.cFileName);
  } while (FindNextFileW(hFind, &findData) != 0);

  FindClose(hFind);
  return files;
}
```

Рисунок 4.8 – Реалізація функції «GetFiles» класу «Directory»

```cpp
std::vector<DriveInfo> DriveInfo::GetDrives() {
  std::vector<DriveInfo> drives;

  WCHAR buffer[256] = {[0]=0};
  GetLogicalDriveStringsW(nBufferLength: 256, lpBuffer: buffer);

  WCHAR *drive = buffer;
  while (*drive) {
    drives.emplace_back(std::wstring(s: drive));
    drive += wcslen(Str: drive) + 1;
  }

  return drives;
}
```

Рисунок 4.9 – Реалізація функції «GetDrives» класу «DriveInfo»

4.2.1.4 Порівняти швидкодію функцій при їх використанні для C#, C в разі використання великих файлів.

Створимо прості програми для порівняння швидкодії функцій стандартної бібліотеки C# та функцій реалізованих на C++.

Для замірів часу на C++ використаємо модуль «chrono» із стандартної бібліотеки.

Для зручності тестів заздалегідь підготуємо всі змінні із потрібними шляхами:

```cpp
#include <iostream>
#include <chrono>
#include <vector>
#include <cstring>

#include "include/System/IO/File.h"
#include "include/System/IO/FileStream.h"
#include "include/System/IO/Directory.h"
#include "include/System/IO/DriveInfo.h"

using namespace System::IO;

int main()
{
    const std::wstring filesDir = L"../files";
    const std::wstring filePath = filesDir + L"/bigfile.dat";
    const std::wstring copyPath = filesDir + L"/bigfile_copy.dat";
    const std::wstring movedCopyPath = filesDir + L"/moved_bigfile_copy.dat";
    const std::wstring movedDirPath = L"../files_moved";
    const size_t blockSize = 1024 * 1024; // 1 MB
    const int32_t iterations = 100; // 100 MB file

    if (!Directory::Exists(filesDir))
        Directory::Create(filesDir);

    std::vector<char> buffer(blockSize, 'A');
    DWORD bytesWritten = 0;
```

Рисунок 4.10 – Підготовка до тестів

Додамо код для тестів:

```
auto startWrite = std::chrono::high_resolution_clock::now();
{
    FileStream outStream(filePath, GENERIC_WRITE, CREATE_ALWAYS);
    for (int32_t i = 0; i < iterations; i++)
        outStream.Write(buffer.data(), static_cast<DWORD>(buffer.size()), bytesWritten);
}
auto endWrite = std::chrono::high_resolution_clock::now();
std::wcout << L"C++ Write time: "
           << std::chrono::duration_cast<std::chrono::milliseconds>(endWrite - startWrite).count()
           << L" ms" << std::endl;
```

Рисунок 4.11 – Тест запису в файл

```
auto startCopy = std::chrono::high_resolution_clock::now();
File::Copy(filePath, copyPath, true);
auto endCopy = std::chrono::high_resolution_clock::now();
std::wcout << L"C++ Copy time: "
           << std::chrono::duration_cast<std::chrono::milliseconds>(endCopy - startCopy).count()
           << L" ms" << std::endl;
```

Рисунок 4.12 – Тест копіювання файлу

```
auto startRead = std::chrono::high_resolution_clock::now();
{
    FileStream inStream(filePath, GENERIC_READ, OPEN_EXISTING);
    std::vector<char> readBuffer(blockSize);
    DWORD bytesRead = 0;
    while (inStream.Read(readBuffer.data(), static_cast<DWORD>(readBuffer.size()), bytesRead) &&
           bytesRead > 0) {}
}
auto endRead = std::chrono::high_resolution_clock::now();
std::wcout << L"C++ Read time: "
           << std::chrono::duration_cast<std::chrono::milliseconds>(endRead - startRead).count()
           << L" ms" << std::endl;
```

Рисунок 4.13 – Тест читання з файлу

```
auto startAttr = std::chrono::high_resolution_clock::now();
DWORD attrs = File(filePath).GetAttributes();
File(filePath).SetAttributes(attrs | FILE_ATTRIBUTE_HIDDEN);
File(filePath).SetAttributes(attrs);
auto endAttr = std::chrono::high_resolution_clock::now();
std::wcout << L"File attribute toggle time: "
           << std::chrono::duration_cast<std::chrono::milliseconds>(endAttr - startAttr).count()
           << L" ms" << std::endl;
```

Рисунок 4.14 – Тест отримання та встановлення атрибутів файлу

```
auto startMoveFile = std::chrono::high_resolution_clock::now();
File::Move(copyPath, movedCopyPath);
auto endMoveFile = std::chrono::high_resolution_clock::now();
std::wcout << L"Move file time: "
           << std::chrono::duration_cast<std::chrono::milliseconds>(endMoveFile - startMoveFile).count()
           << L" ms" << std::endl;
```

Рисунок 4.15 – Тест переміщення файлу

```
if (!Directory::Exists(movedDirPath))
    Directory::Create(movedDirPath);

auto startMoveDir = std::chrono::high_resolution_clock::now();
Directory::Move(filesDir, movedDirPath + L"/files");
auto endMoveDir = std::chrono::high_resolution_clock::now();
std::wcout << L"Move directory time: "
           << std::chrono::duration_cast<std::chrono::milliseconds>(endMoveDir - startMoveDir).count()
           << L" ms" << std::endl;
```

Рисунок 4.16 – Тест переміщення директорії

```
std::wcout << L"Listing drives:\n";
auto fixedDrives = DriveInfo::GetDrives();
for (const auto &d : fixedDrives) {
  if (d.IsReady && d.DriveType == DriveType::Fixed &&
      d.DriveFormat ≠ L"squashfs") {
    std::wcout << L"Drive: " << d.Name << L" | Type: "
               << static_cast<UINT>(d.DriveType) << L" | Format: "
               << d.DriveFormat << L" | Total: "
               << d.TotalSize / (1024 * 1024) << L" MB" << L" | Free: "
               << d.AvailableFreeSpace / (1024 * 1024) << L" MB\n";
  }
}
```

Рисунок 4.17 – Тест отримання інформації про носії даних

Для замірів часу на C# скористаємося класом «StopWatch» стандартної бібліотеки.

Для зручності тестів заздалегідь підготуємо всі змінні із потрібними шляхами:

```
string filesDir = "../files";
string filePath = Path.Combine(filesDir, "bigfile.dat");
string copyPath = Path.Combine(filesDir, "bigfile_copy.dat");
string movedCopyPath = Path.Combine(filesDir, "moved_bigfile_copy.dat");
string movedDirPath = "../files_moved";
int blockSize = 1024 * 1024; // 1 MB
int iterations = 100; // 100 MB

if (!Directory.Exists(filesDir))
    Directory.CreateDirectory(filesDir);

byte[] buffer = new byte[blockSize];
for (int i = 0; i < buffer.Length; i++)
    buffer[i] = (byte)'A';
```

Рисунок 4.18 – Підготовка до тестів

Додамо код для тестів:

```
var swWrite = Stopwatch.StartNew();
using (var fs = new FileStream(filePath, FileMode.Create, FileAccess.Write))
{
    for (int i = 0; i < iterations; i++)
        fs.Write(buffer, 0, buffer.Length);
}
swWrite.Stop();
Console.WriteLine($"C# Write time: {swWrite.ElapsedMilliseconds} ms");
```

Рисунок 4.19 – Тест запису в файл

```
var swCopy = Stopwatch.StartNew();
File.Copy(filePath, copyPath, true);
swCopy.Stop();
Console.WriteLine($"C# Copy time: {swCopy.ElapsedMilliseconds} ms");
```

Рисунок 4.20 – Тест копіювання файлу

```
var swRead = Stopwatch.StartNew();
using (var fs = new FileStream(filePath, FileMode.Open, FileAccess.Read))
{
    byte[] readBuffer = new byte[blockSize];
    while (fs.Read(readBuffer, 0, readBuffer.Length) > 0) { }
}
swRead.Stop();
Console.WriteLine($"C# Read time: {swRead.ElapsedMilliseconds} ms");
```

Рисунок 4.21 – Тест читання з файлу

```
var swAttr = Stopwatch.StartNew();
FileAttributes originalAttrs = File.GetAttributes(filePath);
File.SetAttributes(filePath, originalAttrs | FileAttributes.Hidden);
File.SetAttributes(filePath, originalAttrs);
swAttr.Stop();
Console.WriteLine($"File attribute toggle time: {swAttr.ElapsedMilliseconds} ms");
```

Рисунок 4.22 – Тест отримання та встановлення атрибутів файлу

```
var swMoveFile = Stopwatch.StartNew();
File.Move(copyPath, movedCopyPath, true);
swMoveFile.Stop();
Console.WriteLine($"Move file time: {swMoveFile.ElapsedMilliseconds} ms");
```

Рисунок 4.23 – Тест переміщення файлу

```
var swMoveDir = Stopwatch.StartNew();
Directory.Move(filesDir, Path.Combine(movedDirPath, "files"));
swMoveDir.Stop();
Console.WriteLine($"Move directory time: {swMoveDir.ElapsedMilliseconds} ms");
```

Рисунок 4.24 – Тест переміщення директорії

```
Console.WriteLine("Listing fixed drives:");
var fixedDrives = DriveInfo.GetDrives()
                           .Where(d => d.IsReady &&
                                       d.DriveType == DriveType.Fixed &&
                                       d.DriveFormat != "squashfs" &&
                                       d.DriveFormat != "tracefs" &&
                                       d.DriveFormat != "pstorefs" &&
                                       d.DriveFormat != "bpf_fs" &&
                                       d.DriveFormat != "rpc_pipefs");

foreach (var drive in fixedDrives)
{
    Console.WriteLine($"Drive: {drive.Name} | " +
                      $"Type: {drive.DriveType} | Format: {drive.DriveFormat} | " +
                      $"Total: {drive.TotalSize / (1024 * 1024)} MB | " +
                      $"Free: {drive.AvailableFreeSpace / (1024 * 1024)} MB");
}
```

Рисунок 4.25 – Тест отримання інформації про носії даних

Запустимо обидві програми, та порівняємо результати.

```
CHAT   TERMINAL                                          zsh - cs

➜  cpp make
C++ Write time: 79 ms
C++ Copy time: 190 ms
C++ Read time: 14 ms
File attribute toggle time: 0 ms
Move file time: 0 ms
Move directory time: 0 ms
Listing drives:
Drive: C:\ | Type: 3 | Format: NTFS | Total: 477606 MB | Free: 404831 MB
Drive: Z:\ | Type: 3 | Format: NTFS | Total: 477606 MB | Free: 404831 MB
➜  cpp cd ../cs
➜  cs dotnet build && dotnet run
Restore complete (0.5s)
  cs succeeded (0.2s) → bin/Debug/net9.0/cs.dll

Build succeeded in 1.3s
C# Write time: 79 ms
C# Copy time: 142 ms
C# Read time: 19 ms
File attribute toggle time: 0 ms
Move file time: 0 ms
Move directory time: 0 ms
Listing fixed drives:
Drive: / | Type: Fixed | Format: ext3 | Total: 477606 MB | Free: 404622 MB
Drive: /boot/efi | Type: Fixed | Format: msdos | Total: 2043 MB | Free: 2043 MB
➜  cs
```

Рисунок 4.26 – Результат тестування програм

Варто зазначити, що оскільки лабораторна робота виконувалась на операційній системі Linux, програми показують дещо різні результати. Для запуску C++ версії було використано шар трансляції Wine, в той час коли C# версія працює нативно, через це інформація про диски в C++ та C# версії не співпадають.

Швидкодія обох версій відрізняється незначно, це можна пояснити тим, що обидві програми використовують системну API (WinAPI в випадку із Windows) для взаємодії із файловою системою.

4.3 Висновки

Під час виконання даної лабораторної роботи ми навчилися практичному використанню функцій WinAPI для роботи з файлами.