ЛАБОРАТОРНА РОБОТА №3

Тема: Створення утіліти «DiskInfo»

Мета роботи: У даній лабораторній роботі, використовуючи виклики системних функцій, отримати інформацію про дискову підсистему.

Завдання 1

1. Список усіх логічних дисків в системі.
2. Отримати тип кожного диску присутнього в системі, та дати пояснення для кожного типу диску.
3. Отримати інформацію про диски в системі та про файлові системи Які Використовують на них.
4. Отримати інформацію про зайнятості та вільне місце на кожному з дисків.
5. Отримати інформацію про системну пам'ять.
6. Отримати інформацію про Назву комп'ютера
7. Отримати Назву поточного користувача
8. Отримати інформацію про поточний системний каталог, Тимчасовий каталог, поточний робочий каталог.
9. Для обраних каталогу на диску, Включити спостереження за змінами, продемонструвати відслідковування більше однієї зміни. Зміни записувати в лог файл.

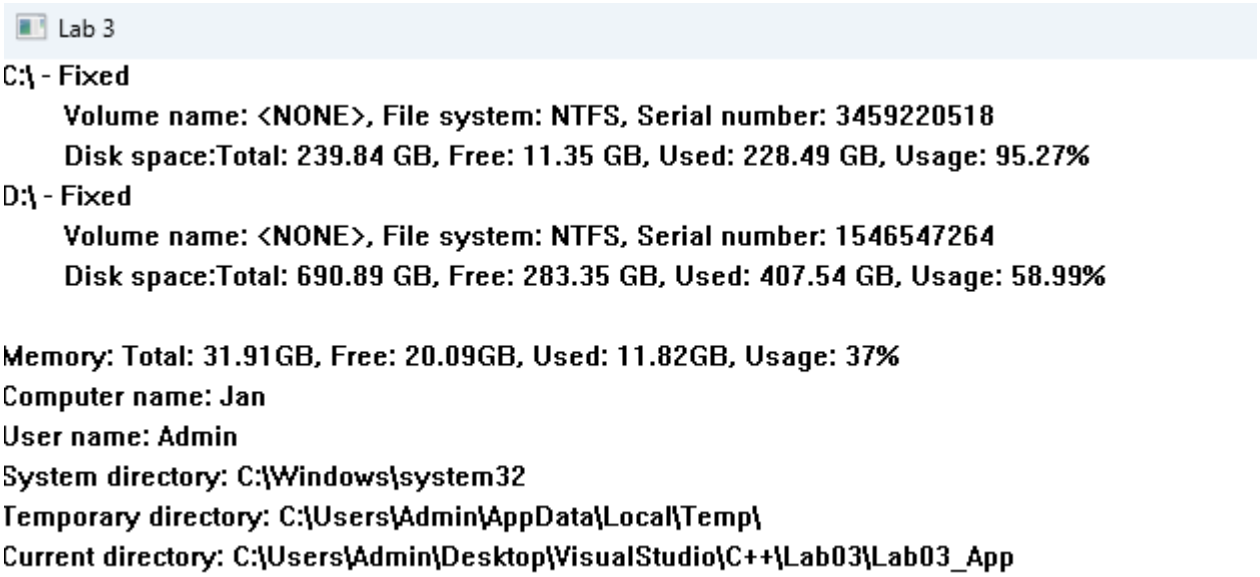

Рис. 1. Результат виконання.

```
[2025-05-21 23:09:26] Added - New Microsoft Word Document.docx
[2025-05-21 23:09:27] Renamed from - New Microsoft Word Document.docx
[2025-05-21 23:09:27] Renamed to - sdasda.docx
[2025-05-21 23:09:30] Added - New Bitmap image.bmp
[2025-05-21 23:09:32] Renamed from - New Bitmap image.bmp
[2025-05-21 23:09:32] Renamed to - sdasdad.bmp
[2025-05-21 23:09:35] Added - New Text Document.txt
[2025-05-21 23:09:36] Renamed from - New Text Document.txt
[2025-05-21 23:09:36] Renamed to - fasfasfa.txt
[2025-05-21 23:09:45] Added - New Text Document.txt
[2025-05-21 23:09:47] Renamed from - New Text Document.txt
[2025-05-21 23:09:47] Renamed to - sfasfasfasfhrh.txt
```

Рис. 2. Вміст лог-файлу.

Лістинг програми:

```
#include <Shlwapi.h>
#define _SILENCE_EXPERIMENTAL_FILESYSTEM_DEPRECATION_WARNING
#include <experimental/filesystem>
#include <lmcons.h>
#include <iterator>
#include <processenv.h>
#include <sysinfoapi.h>
#include <cstdint>
#include <errhandlingapi.h>
#include <fileapi.h>
#include <iomanip>
#include <ios>
#include <minwindef.h>
#include <ostream>
#include <sstream>
#include <stdexcept>
#include <string>
#include <thread>
#include <vector>
#include <windef.h>
#include <iostream>
#include <wingdi.h>
#include <winnt.h>
#include <winuser.h>
#include <windows.h>
#include <string>
#include <VersionHelpers.h>

#pragma comment(lib, "Shlwapi.lib")
#pragma comment(lib, "User32.lib")
#pragma comment(lib, "Gdi32.lib")

std::wstring space_indent = L"        ";
int timer_interval = 100;
int timer_id = 1;

std::wstring directory_path = L"C:\\Users\\Admin\\Desktop\\VisualStudio\\C++\\Lab03\\target_folder";
std::wstring log_file_path = L"./lab.log";

std::wstring get_time_wstring()
{
    SYSTEMTIME time;
    GetLocalTime(&time);

    std::wstringstream time_stream;

    time_stream << std::setw(4) << std::setfill(L'0') << time.wYear << L"-"
        << std::setw(2) << std::setfill(L'0') << time.wMonth << L"-"
        << std::setw(2) << std::setfill(L'0') << time.wDay << L" "
        << std::setw(2) << std::setfill(L'0') << time.wHour << L":"
        << std::setw(2) << std::setfill(L'0') << time.wMinute << L":"
        << std::setw(2) << std::setfill(L'0') << time.wSecond;

    return time_stream.str();
}

void monitor_directory_changes()
{
```

```
HANDLE log_file_handle = CreateFileW(
   log_file_path.c_str(),
   GENERIC_WRITE,
   FILE_SHARE_WRITE | FILE_SHARE_READ | FILE_SHARE_DELETE,
   NULL,
   OPEN_ALWAYS,
   FILE_ATTRIBUTE_NORMAL,
   NULL
);

if (!PathFileExistsW(directory_path.c_str()))
{
   CreateDirectoryW(directory_path.c_str(), NULL);
}

HANDLE directory_handle = CreateFileW(
   directory_path.c_str(),
   FILE_LIST_DIRECTORY,
   FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
   NULL,
   OPEN_EXISTING,
   FILE_FLAG_BACKUP_SEMANTICS,
   NULL
);

if (directory_handle == INVALID_HANDLE_VALUE)
{
   std::cerr << "Failed to open the directory for watching!" << std::endl;
   return;
}

char buffer[256];
DWORD bytes_returned;

while (true)
{
   if (ReadDirectoryChangesW(directory_handle,
      &buffer,
      sizeof(buffer),
      TRUE,
      FILE_NOTIFY_CHANGE_FILE_NAME |
      FILE_NOTIFY_CHANGE_DIR_NAME |
      FILE_NOTIFY_CHANGE_ATTRIBUTES |
      FILE_NOTIFY_CHANGE_SIZE |
      FILE_NOTIFY_CHANGE_LAST_WRITE |
      FILE_NOTIFY_CHANGE_LAST_WRITE |
      FILE_NOTIFY_CHANGE_LAST_ACCESS |
      FILE_NOTIFY_CHANGE_CREATION |
      FILE_NOTIFY_CHANGE_SECURITY,
      &bytes_returned,
      NULL,
      NULL))
   {
      FILE_NOTIFY_INFORMATION* information = reinterpret_cast<FILE_NOTIFY_INFORMATION*>(buffer);

      while (true)
      {
         std::wstring filename(information->FileName, information->FileNameLength / sizeof(WCHAR));
         std::wstring action;

         switch (information->Action)
         {
         case FILE_ACTION_ADDED: action = L"Added"; break;
         case FILE_ACTION_REMOVED: action = L"Removed"; break;
         case FILE_ACTION_MODIFIED: action = L"Modified"; break;
         case FILE_ACTION_RENAMED_OLD_NAME: action = L"Renamed from"; break;
         case FILE_ACTION_RENAMED_NEW_NAME: action = L"Renamed to"; break;
         default: action = L"Unknown action"; break;
         }

         std::wstring text = L"[" + get_time_wstring() + L"] " + std::wstring(action) + L" - " +
            std::wstring(filename) + L"\n";

         DWORD bytes_written;

         SetFilePointer(log_file_handle, 0, NULL, FILE_END);

         WriteFile(
            log_file_handle,
            text.c_str(),
            static_cast<DWORD>(text.length() * sizeof(wchar_t)),
            &bytes_written,
            NULL
```

```
            );

            if (information->NextEntryOffset == 0) break;

            information = reinterpret_cast<FILE_NOTIFY_INFORMATION*>(
              reinterpret_cast<BYTE*>(information) + information->NextEntryOffset
              );
          }
        }
    }

    CloseHandle(log_file_handle);
    CloseHandle(directory_handle);
}

std::wstring percentage_formatter(double value)
{
    double percent = value * 100;

    std::wostringstream out;

    out << std::fixed << std::setprecision(2) << percent << L"%";

    return out.str();
}

std::wstring format_memory(ULONGLONG bytes)
{
    const double GB = 1024.0 * 1024.0 * 1024.0;
    const double MB = 1024.0 * 1024.0;

    std::wostringstream out;

    out << std::fixed << std::setprecision(2);

    if (bytes >= GB) out << (bytes / GB) << L"GB";
    else out << (bytes / MB) << L"MB";

    return out.str();
}

void get_system_directory_path(std::vector<std::wstring>& lines)
{
    wchar_t system_directory[MAX_PATH + 1];
    DWORD system_directory_size = sizeof(system_directory) / sizeof(system_directory[0]);

    GetSystemDirectoryW(system_directory, system_directory_size);

    lines.push_back(L"System directory: " + std::wstring(system_directory));
}

void get_temporary_directory_path(std::vector<std::wstring>& lines)
{
    wchar_t temporary_directory[MAX_PATH + 1];
    DWORD temporary_directory_size = sizeof(temporary_directory) / sizeof(temporary_directory[0]);

    GetTempPathW(temporary_directory_size, temporary_directory);

    lines.push_back(L"Temporary directory: " + std::wstring(temporary_directory));
}

void get_current_directory_path(std::vector<std::wstring>& lines)
{
    wchar_t current_directory[MAX_PATH + 1];
    DWORD current_directory_size = sizeof(current_directory) / sizeof(current_directory[0]);

    GetCurrentDirectoryW(current_directory_size, current_directory);

    lines.push_back(L"Current directory: " + std::wstring(current_directory));
}

void get_directory_paths(std::vector<std::wstring>& lines)
{
    get_system_directory_path(lines);
    get_temporary_directory_path(lines);
    get_current_directory_path(lines);
}

void get_computer_name(std::vector<std::wstring>& lines)
{
    wchar_t computer_name[MAX_COMPUTERNAME_LENGTH + 1];
    DWORD computer_name_size = sizeof(computer_name) / sizeof(computer_name[0]);
```

```
    GetComputerNameExW(ComputerNameDnsHostname, computer_name, &computer_name_size);

    lines.push_back(L"Computer name: " + std::wstring(computer_name));
}

void get_user_name(std::vector<std::wstring>& lines)
{
    wchar_t user_name[UNLEN + 1];
    DWORD user_name_size = sizeof(user_name) / sizeof(user_name[0]);

    GetUserNameW(user_name, &user_name_size);

    lines.push_back(L"User name: " + std::wstring(user_name));
}

std::wstring get_drive_type_wstring(std::wstring volume_root)
{
    int drive_type = GetDriveTypeW(volume_root.c_str());

    switch (drive_type)
    {
    case 0: return L"Unknown";
    case 1: return L"Invalid";
    case 2: return L"Removable";
    case 3: return L"Fixed";
    case 4: return L"Remote";
    case 5: return L"CD-ROM";
    }

    throw std::invalid_argument("Invalid drive type!");
}

std::wstring human_readble_bytes(uint64_t bytes)
{
    const wchar_t* suffixes[] = { L"B", L"KB", L"MB", L"GB", L"TB", L"PB" };
    int i = 0;
    double count = static_cast<double>(bytes);

    while (count >= 1024 && i < 5)
    {
        count /= 1024.0;
        ++i;
    }

    std::wostringstream out;
    out << std::fixed << std::setprecision(2) << count << L" " << suffixes[i];
    return out.str();
}

void get_volume_information(std::wstring volume_root, std::vector<std::wstring>& lines, std::wstring space_indent = L"")
{
    wchar_t volume_name_buffer[MAX_PATH + 1] = { };
    DWORD volume_serial_number;
    wchar_t file_system_name_buffer[MAX_PATH + 1] = { };
    DWORD max_component_length = 0;
    DWORD file_system_flags = 0;

    if (GetVolumeInformationW(volume_root.c_str(), volume_name_buffer, static_cast<DWORD>(MAX_PATH + 1),
        &volume_serial_number, &max_component_length, &file_system_flags,
        file_system_name_buffer, static_cast<DWORD>(MAX_PATH + 1)))
    {
        lines.push_back(space_indent +
            L"Volume name: " + std::wstring((volume_name_buffer[0] ? volume_name_buffer : L"<NONE>")) +
            L", File system: " + std::wstring((file_system_name_buffer[0] ? file_system_name_buffer : L"<NONE>")) +
            L", Serial number: " + std::to_wstring(volume_serial_number));
    }
}

void get_storage_information(std::wstring volume_root, std::vector<std::wstring>& lines, std::wstring space_indent = L"")
{
    DWORD sectors_per_cluster;
    DWORD bytes_per_sector;
    DWORD number_of_free_clusters;
    DWORD total_number_of_clusters;

    if (GetDiskFreeSpaceW(volume_root.c_str(), &sectors_per_cluster,
        &bytes_per_sector, &number_of_free_clusters, &total_number_of_clusters))
    {
        uint64_t total_bytes = static_cast<uint64_t>(total_number_of_clusters) * sectors_per_cluster * bytes_per_sector;
        uint64_t free_bytes = static_cast<uint64_t>(number_of_free_clusters) * sectors_per_cluster * bytes_per_sector;
        uint64_t used_bytes = total_bytes - free_bytes;

        lines.push_back(space_indent + L"Disk space:" +
```

```
            L"Total: " + human_readble_bytes(total_bytes) + L", " +
            L"Free: " + human_readble_bytes(free_bytes) + L", " +
            L"Used: " + human_readble_bytes(used_bytes) + L", " +
            L"Usage: " + percentage_formatter(static_cast<double>(used_bytes) / total_bytes));
    }
}

void get_drive_information(std::vector<std::wstring>& lines)
{
    DWORD drives = GetLogicalDrives();

    for (int i = 0; i < 26; i++)
    {
        if (drives & (1 << i))
        {
            char disk_letter = char('A' + i);

            std::wstring volume_root = std::wstring(1, disk_letter) + L":\\";

            lines.push_back(std::wstring(1, disk_letter) + L":\\ - " + get_drive_type_wstring(volume_root));

            get_volume_information(volume_root, lines, space_indent);
            get_storage_information(volume_root, lines, space_indent);
        }
    }
}

void get_memory_information(std::vector<std::wstring>& lines)
{
    MEMORYSTATUSEX memory_struct;
    memory_struct.dwLength = sizeof(memory_struct);

    if (GlobalMemoryStatusEx(&memory_struct))
    {
        lines.push_back(L"Memory: Total: " + format_memory(memory_struct.ullTotalPhys) +
            L", Free: " + format_memory(memory_struct.ullAvailPhys) +
            L", Used: " + format_memory(memory_struct.ullTotalPhys - memory_struct.ullAvailPhys) +
            L", Usage: " + std::to_wstring(memory_struct.dwMemoryLoad) + L"%");
    }
}

void render(HDC painting_handle, std::vector<std::wstring> lines)
{
    int gap_indent = 20;

    for (size_t i = 0; i < lines.size(); i++)
    {
        TextOutW(painting_handle, 0, static_cast<int>(gap_indent * i), lines[i].c_str(), static_cast<int>(lines[i].length()));
    }
}

LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
    case WM_PAINT:
    {
        PAINTSTRUCT paint_struct;

        HDC painting_handle = BeginPaint(hwnd, &paint_struct);

        std::vector<std::wstring> lines = std::vector<std::wstring>();

        get_drive_information(lines);

        lines.push_back(L"");

        get_memory_information(lines);
        get_computer_name(lines);
        get_user_name(lines);
        get_directory_paths(lines);

        render(painting_handle, lines);

        EndPaint(hwnd, &paint_struct);

        SetTimer(hwnd, timer_id, timer_interval, NULL);

        return 0;
    }

    case WM_TIMER:
        if (wParam == timer_id) InvalidateRect(hwnd, NULL, true);
```

```
        return 0;

    case WM_DESTROY:
    {
        PostQuitMessage(0);
        return 0;
    }
    }

    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    std::thread monitor_thread(monitor_directory_changes);

    std::wstring class_name = L"MainWindow";

    HINSTANCE handle = GetModuleHandle(NULL);

    WNDCLASSW window_class = { };

    window_class.lpfnWndProc = WindowProcedure;
    window_class.hInstance = handle;
    window_class.lpszClassName = class_name.c_str();
    window_class.hCursor = LoadCursor(handle, IDC_ARROW);
    window_class.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);

    if (!RegisterClassW(&window_class))
    {
        std::cerr << "Error registering the \"MainWindow\" window class" << std::endl;
        return 1;
    }

    HWND window_handle = CreateWindowExW(0,
        class_name.c_str(),
        L"Lab 3",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, handle, NULL);

    if (window_handle == NULL)
    {
        std::cerr << "Error registering the \"MainWindow\" window class" << std::endl;
        return 1;
    }

    ShowWindow(window_handle, SW_SHOW);
    UpdateWindow(window_handle);

    MSG message;

    while (GetMessageW(&message, NULL, 0, 0))
    {
        TranslateMessage(&message);
        DispatchMessage(&message);
    }

    UnregisterClassW(class_name.c_str(), handle);

    monitor_thread.join();

    return static_cast<int>(message.wParam);
}
```

Посилання на репозиторій: https://github.com/JanRizhenko/C-development/tree/master/Lab03

Висновок: У цій лабораторній роботі була створена утиліта «DiskInfo», яка за допомогою системних викликів надає розширену інформацію про дискову підсистему комп'ютера. Зокрема, програма виводить перелік логічних дисків, визначає їх типи з поясненням, отримує відомості про файлові системи, обсяг використаного та вільного простору, а також системну пам'ять, назву комп'ютера й поточного користувача. Крім того, реалізовано функціонал для визначення основних системних каталогів і спостереження за змінами у вибраній теці з фіксацією подій у лог-файл.