**ЛАБОРАТОРНА РОБОТА № 3**

**Тема:** «Створення утіліти «DiskInfo»»

**Хід роботи:**

**Завдання:** використовуючи виклики системних функцій, отримати інформацію про дискову підсистему:

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.
10. Переліг деяких функцій, які можуть бути використанні при виконанні лабораторної роботи:

**Лістинг коду Program.cs:**

namespace Lab03App

{

class Program

{

static void Main(string[] args)

{

string watchedFolder = @"P:";

DirInfoLogger dirInfoLogger = new();

Console.WriteLine("Logical disks:");

DriveInfo[] allDrives = DriveInfo.GetDrives();

foreach (DriveInfo drive in allDrives)

{

var driveInfoLogger = new DriveInfoLogger(drive);

Console.WriteLine("\n");

Console.WriteLine(driveInfoLogger.GetLog());

}

Console.WriteLine(dirInfoLogger.GetLog());

Console.WriteLine($"Monitoring folder: {watchedFolder}");

using var watcher = new FileSystemWatcher(watchedFolder)

{

NotifyFilter = NotifyFilters.Attributes

| NotifyFilters.CreationTime

| NotifyFilters.DirectoryName

| NotifyFilters.FileName

| NotifyFilters.LastAccess

| NotifyFilters.LastWrite

| NotifyFilters.Security

| NotifyFilters.Size,

IncludeSubdirectories = true,

EnableRaisingEvents = true

};

watcher.Changed += WatcherEvents.OnChanged;

watcher.Created += WatcherEvents.OnCreated;

watcher.Deleted += WatcherEvents.OnDeleted;

watcher.Renamed += WatcherEvents.OnRenamed;

watcher.Error += WatcherEvents.OnError;

Console.WriteLine("Press enter to exit.");

Console.ReadLine();

}

}

}

**Лістинг моделі DiscInfoModel.cs:**

using System;

using System.IO;

namespace Lab03App.Models

{

public class DiskInfoModel

{

public string Name { get; set; }

public string Type { get; set; }

public string VolumeLabel { get; set; }

public string FileSystem { get; set; }

public string RootDirectory { get; set; }

public string AvailableFreeSpaceGB { get; set; }

public string TotalFreeSpaceGB { get; set; }

public string TotalSizeGB { get; set; }

public DiskInfoModel(DriveInfo drive)

{

Name = drive.Name;

Type = drive.DriveType.ToString();

VolumeLabel = drive.VolumeLabel;

FileSystem = drive.DriveFormat;

RootDirectory = drive.RootDirectory.ToString();

AvailableFreeSpaceGB = (drive.AvailableFreeSpace / 1024 / 1024 / 1024).ToString();

TotalFreeSpaceGB = (drive.TotalFreeSpace / 1024 / 1024 / 1024).ToString();

TotalSizeGB = (drive.TotalSize / 1024 / 1024 / 1024).ToString();

}

public override string ToString()

{

return $@"Disk Type: {Type}

Volume Label: {VolumeLabel}

File System: {FileSystem}

Root Directory: {RootDirectory}

Available Space to Current User: {AvailableFreeSpaceGB} GB

Total Available Space: {TotalFreeSpaceGB} GB

Total Size of Drive: {TotalSizeGB} GB";

}

}

}

**Лістинг класу DirInfoLogger.cs:**

using System;

using System.IO;

using System.Text;

namespace Lab03App.Tools

{

public class DirInfoLogger

{

private readonly StringBuilder \_logBuilder;

private readonly DirectoryInfo \_systemDirectoryInfo;

private readonly DirectoryInfo \_tempDirectoryInfo;

private readonly DirectoryInfo \_currentDirectoryInfo;

public DirInfoLogger()

{

\_logBuilder = new StringBuilder();

\_systemDirectoryInfo = new DirectoryInfo(Environment.SystemDirectory);

\_tempDirectoryInfo = new DirectoryInfo(Path.GetTempPath());

\_currentDirectoryInfo = new DirectoryInfo(Directory.GetCurrentDirectory());

LogDirectoryInformation();

}

private void LogDirectoryInformation()

{

LogDirectoryDetails("System directory", \_systemDirectoryInfo);

LogDirectoryDetails("Temporary directory", \_tempDirectoryInfo);

LogDirectoryDetails("Current directory", \_currentDirectoryInfo);

}

private void LogDirectoryDetails(string title, DirectoryInfo directoryInfo)

{

\_logBuilder.AppendLine($"\n{title}:");

\_logBuilder.AppendLine($"Name: {directoryInfo.Name}");

\_logBuilder.AppendLine($"Full name: {directoryInfo.FullName}");

\_logBuilder.AppendLine($"Creation time: {directoryInfo.CreationTime}");

\_logBuilder.AppendLine($"Root: {directoryInfo.Root}");

}

public void ClearLog()

{

\_logBuilder.Clear();

}

public string GetLog()

{

return \_logBuilder.ToString();

}

public override string ToString()

{

return \_logBuilder.ToString();

}

}

}

**Лістинг класу DirWatcher.cs:**

namespace Lab03App.Tools

{

public class DirWatcher

{

private readonly DriveInfoLogger \_driveInfoLogger;

private readonly DirInfoLogger \_dirInfoLogger;

private readonly CancellationTokenSource \_cancellationTokenSource;

private const string LogFilePath = @"P:\labs\log.txt";

public DirWatcher(DriveInfoLogger driveLogger, DirInfoLogger dirLogger)

{

\_driveInfoLogger = driveLogger;

\_dirInfoLogger = dirLogger;

\_cancellationTokenSource = new CancellationTokenSource();

}

[PermissionSet(SecurityAction.Demand, Name = "FullTrust")]

public void MonitorDirAndFile(string path)

{

Task.Run(() => StartMonitoring(path, \_cancellationTokenSource.Token));

}

private void StartMonitoring(string path, CancellationToken cancellationToken)

{

using (FileSystemWatcher watcher = new FileSystemWatcher())

{

watcher.Path = path;

watcher.NotifyFilter = NotifyFilters.LastAccess

| NotifyFilters.LastWrite

| NotifyFilters.FileName

| NotifyFilters.DirectoryName;

watcher.Filter = "";

watcher.Changed += OnChanged;

watcher.EnableRaisingEvents = true;

try

{

while (!cancellationToken.IsCancellationRequested)

{

Thread.Sleep(1000);

}

}

catch (Exception ex)

{

Console.WriteLine($"Exception: {ex.Message}");

}

}

}

private void OnChanged(object sender, FileSystemEventArgs e)

{

try

{

using (StreamWriter sw = File.AppendText(LogFilePath))

{

sw.WriteLine($"File: {e.FullPath} {e.ChangeType}");

}

ShowDisksInfo();

}

catch (Exception ex)

{

Console.WriteLine($"Failed to log: {ex.Message}");

}

}

public void Close()

{

\_cancellationTokenSource.Cancel();

}

public void ShowDisksInfo()

{

\_driveInfoLogger.ClearLog();

\_dirInfoLogger.ClearLog();

DriveInfo[] allDrives = DriveInfo.GetDrives();

foreach (DriveInfo drive in allDrives)

{

var driveLogger = new DriveInfoLogger(drive);

string logData = driveLogger.GetLog();

Console.WriteLine("\n");

Console.WriteLine(logData);

}

Console.WriteLine(\_dirInfoLogger.GetLog());

}

}

}

**Лістинг класу DriveInfoLogger.cs:**

using System;

using System.IO;

using System.Text;

namespace Lab03App.Tools

{

public class DriveInfoLogger

{

private readonly StringBuilder \_logBuilder;

public DriveInfoLogger(DriveInfo drive)

{

\_logBuilder = new StringBuilder();

LogDriveInformation(drive);

}

private void LogDriveInformation(DriveInfo drive)

{

\_logBuilder.AppendLine($"Drive type: {drive.DriveType}");

\_logBuilder.AppendLine($"Volume label: {drive.VolumeLabel}");

\_logBuilder.AppendLine($"File system: {drive.DriveFormat}");

\_logBuilder.AppendLine($"Root directory: {drive.RootDirectory}");

\_logBuilder.AppendLine($"Available space to current user: {drive.AvailableFreeSpace / 1024 / 1024 / 1024} GB");

\_logBuilder.AppendLine($"Total available space: {drive.TotalFreeSpace / 1024 / 1024 / 1024} GB");

\_logBuilder.AppendLine($"Total size of drive: {drive.TotalSize / 1024 / 1024 / 1024} GB");

}

public void ClearLog()

{

\_logBuilder.Clear();

}

public string GetLog()

{

return \_logBuilder.ToString();

}

public override string ToString()

{

return \_logBuilder.ToString();

}

}

}

**Лістинг класу WatcherEvents.cs:**

namespace Lab3Programs.Tools

{

public static class WatcherEvents

{

public static void OnChanged(object sender, FileSystemEventArgs e)

{

if (e.ChangeType != WatcherChangeTypes.Changed)

{

return;

}

using (StreamWriter sw = File.AppendText("log.txt"))

sw.WriteLine($"Changed: {e.FullPath}");

Console.WriteLine($"Changed: {e.FullPath}");

}

public static void OnCreated(object sender, FileSystemEventArgs e)

{

string value = $"Created: {e.FullPath}";

using (StreamWriter sw = File.AppendText("log.txt"))

sw.WriteLine(value);

Console.WriteLine(value);

}

public static void OnDeleted(object sender, FileSystemEventArgs e)

{

string value = $"Deleted: {e.FullPath}";

using (StreamWriter sw = File.AppendText("log.txt"))

sw.WriteLine(value);

Console.WriteLine(value);

}

public static void OnRenamed(object sender, RenamedEventArgs e)

{

string value = $"Renamed:\n\tOld: {e.OldFullPath}\n\tNew: {e.FullPath}";

using (StreamWriter sw = File.AppendText("log.txt"))

sw.WriteLine(value);

}

public static void OnError(object sender, ErrorEventArgs e) =>

PrintException(e.GetException());

public static void PrintException(Exception? ex)

{

if (ex != null)

{

Console.WriteLine($"Message: {ex.Message}");

Console.WriteLine("Stacktrace:");

Console.WriteLine(ex.StackTrace);

Console.WriteLine();

PrintException(ex.InnerException);

}

}

}

}

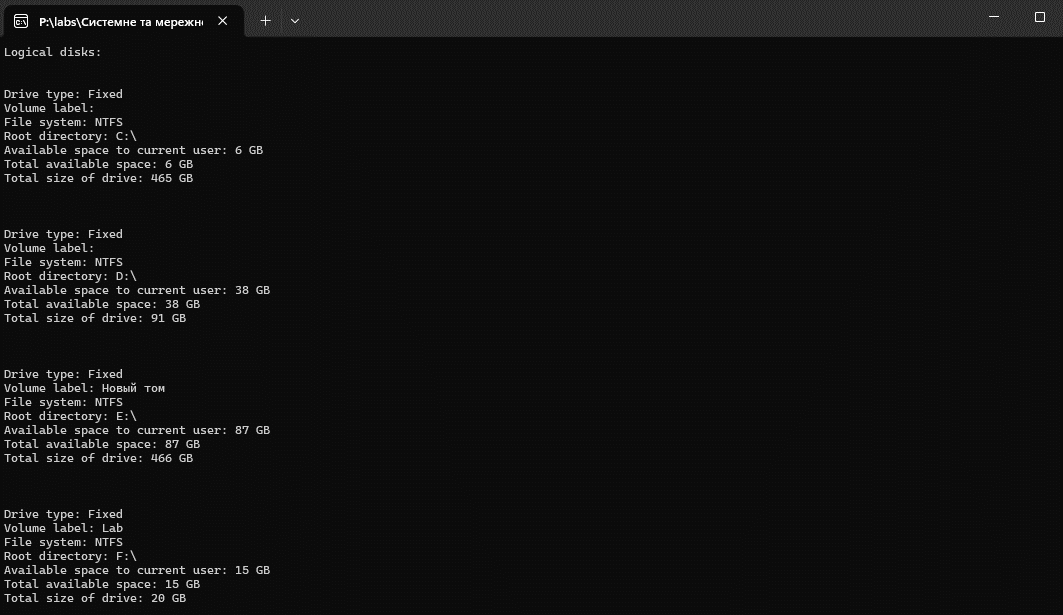


Рис. 1. Результат виконання

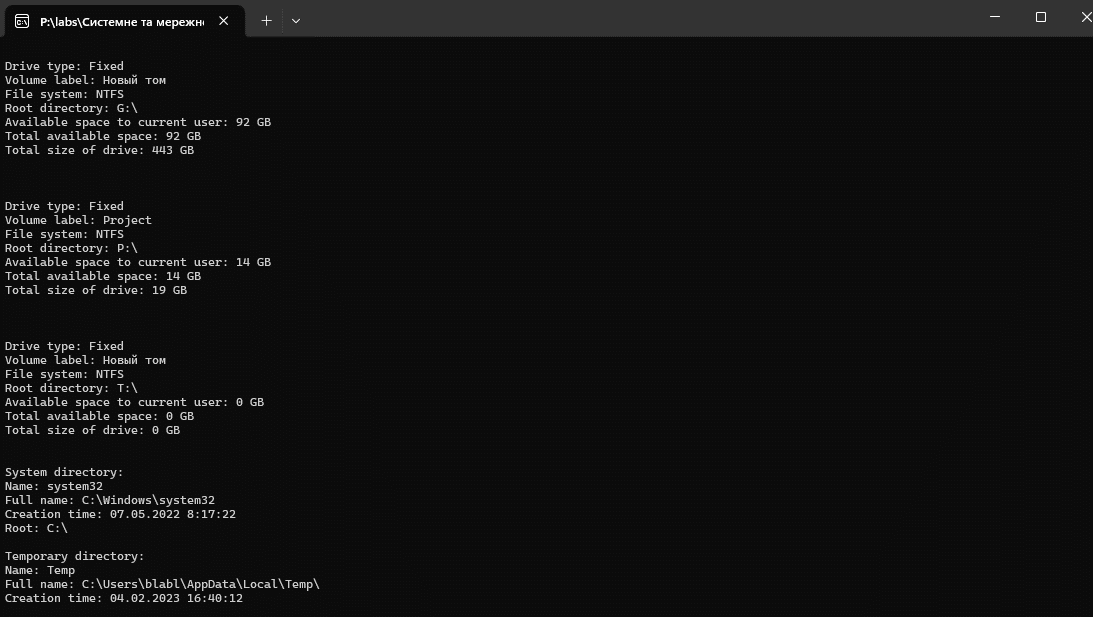


Рис. 2. Результат виконання

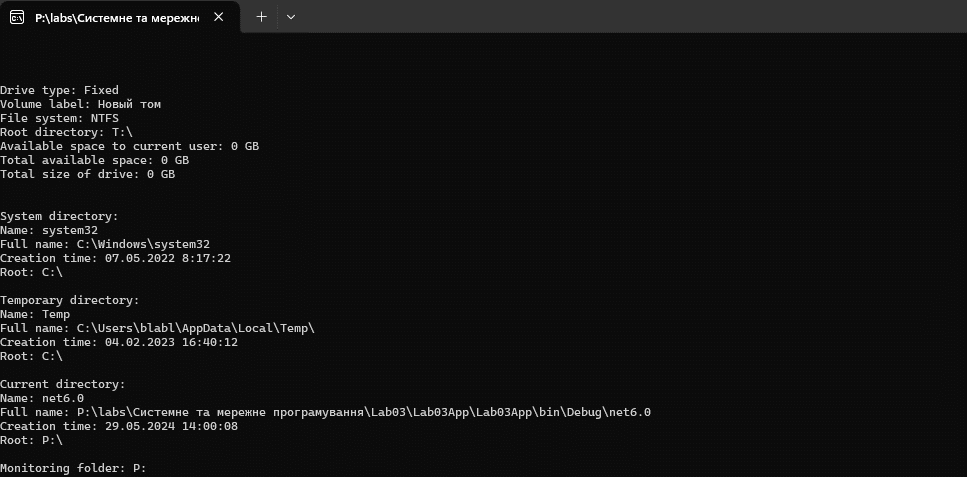


Рис. 3. Результат виконання