Лабораторна робота №3

**Тема:** Багатопотоковість у C#.

**Мета роботи:** навчитися працювати з потоками та процесами у мові C#.

**Виділений час:** 12 годин (4 години лабораторних робіт та 8 годин самостійної роботи)

**Хід роботи:**

***Завдання :***

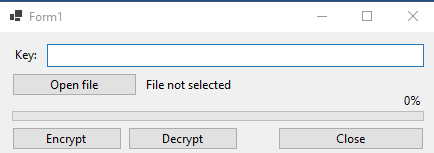
Розробити дві програми:

1) шифрування файлів;

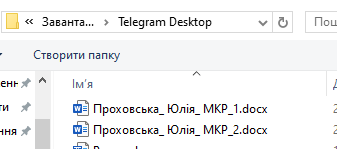
2) менеджер процесів.

Програмний код має бути написаний максимально універсально без прив’язки у класах, що реалізують основний функціонал до інтерфейсу. Передбачається, що дані класи потрібно буде використовувати в наступних лабораторних роботах.

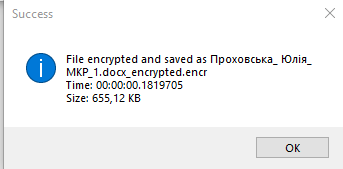
**Результат виконання програми шифрування:**



**Рис.1.**Вигляд форми



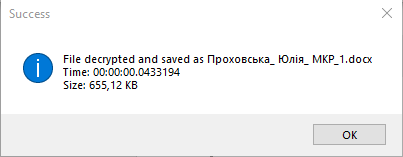
**Рис.2.**Шифрування файлу(вибір файлу)



**Рис.3.**Шифрування файлу



**Рис.4.**Шифрований файл



**Рис.5.**Розшифрування файлу

**Лістинг Cryptography.cs:**

using System.ComponentModel;

namespace Lab3.Classes

{

internal class Cryptography

{

public static string Encrypt(string filePath, string key, BackgroundWorker

sender)

{

var startTimer = DateTime.Now;

var data = File.ReadAllBytes(filePath);

byte[] result = new byte[data.Length];

for (int i = 0; i < data.Length; i++)

{

result[i] = (byte)(data[i] ^ key[i % key.Length]);

if (i % (data.Length / 100) == 0 && sender != null)

sender.ReportProgress((int)(i / (data.Length / 100)));

}

var endTimer = DateTime.Now;

var time = endTimer - startTimer;

var resultPath = Path.GetFileName(filePath) + "\_encrypted.encr";

using (var fileStream = new FileStream(resultPath, FileMode.Create))

{

using (var binaryWriter = new BinaryWriter(fileStream))

{

binaryWriter.Write(result);

}

}

return "File encrypted and saved as " + resultPath + "\nTime: " + time.ToString() + "\nSize: " + Helpers.SizeConvert(result.Length) + Environment.NewLine;

}

public static string Decrypt(string filePath, string key, BackgroundWorker sender)

{

var startTimer = DateTime.Now;

var data = File.ReadAllBytes(filePath);

byte[] result = new byte[data.Length];

for (int i = 0; i < data.Length; i++)

{

result[i] = (byte)(data[i] ^ key[i % key.Length]);

if (i % (data.Length / 100) == 0 && sender != null)

sender.ReportProgress((int)(i / (data.Length / 100)));

}

var endTimer = DateTime.Now;

var time = endTimer - startTimer;

var resultPath = Path.GetFileName(filePath).Replace("\_encrypted.encr", "");

using (var fileStream = new FileStream(resultPath, FileMode.Create))

{

using (var binaryWriter = new BinaryWriter(fileStream))

{

binaryWriter.Write(result);

}

}

return "File decrypted and saved as " + resultPath + "\nTime: " + time.ToString() + "\nSize: " + Helpers.SizeConvert(result.Length) + Environment.NewLine;

}

}

}

**Лістинг Form1:**

using Lab3.Classes;

using System.ComponentModel;

namespace Lab3

{

public partial class Form1 : Form

{

private static string \_path = string.Empty;

private static string \_key = string.Empty;

public Form1()

{

InitializeComponent();

EncryptionBw.WorkerReportsProgress = true;

EncryptionBw.WorkerSupportsCancellation = true;

}

private void openFileBtn\_Click(object sender, EventArgs e)

{

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

\_path = openFileDialog.FileName;

filePathLbl.Text = \_path;

}

}

}

private void EncryptBtn\_Click(object sender, EventArgs e)

{

try

{

Setup();

if (EncryptionBw.IsBusy != true)

EncryptionBw.RunWorkerAsync("Encrypt");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void DecryptBtn\_Click(object sender, EventArgs e)

{

try

{

Setup();

if (EncryptionBw.IsBusy != true)

EncryptionBw.RunWorkerAsync("Decrypt");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void Setup()

{

\_key = keyTb.Text;

if (string.IsNullOrEmpty(\_path) || string.IsNullOrEmpty(\_key))

{

MessageBox.Show("Please select file and enter key", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

openFileBtn.Enabled = false;

DecryptBtn.Enabled = false;

EncryptBtn.Enabled = false;

}

private void EncryptionBw\_DoWork(object sender, DoWorkEventArgs e)

{

BackgroundWorker worker = sender as BackgroundWorker;

if (e.Argument == "Encrypt")

e.Result = Cryptography.Encrypt(\_path, \_key, worker);

else if (e.Argument == "Decrypt")

e.Result = Cryptography.Decrypt(\_path, \_key, worker);

}

private void EncryptionBw\_ProgressChanged(object sender, ProgressChangedEventArgs e)

{

progressBar1.Value = e.ProgressPercentage;

ProgressLbl.Text = e.ProgressPercentage.ToString() + "%";

}

private void EncryptionBw\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

MessageBox.Show(e.Result.ToString(), "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

\_path = string.Empty;

\_key = string.Empty;

filePathLbl.Text = string.Empty;

keyTb.Text = string.Empty;

openFileBtn.Enabled = true;

EncryptBtn.Enabled = true;

DecryptBtn.Enabled = true;

progressBar1.Value = 0;

ProgressLbl.Text = "0%";

}

private void CloseBtn\_Click(object sender, EventArgs e)

{

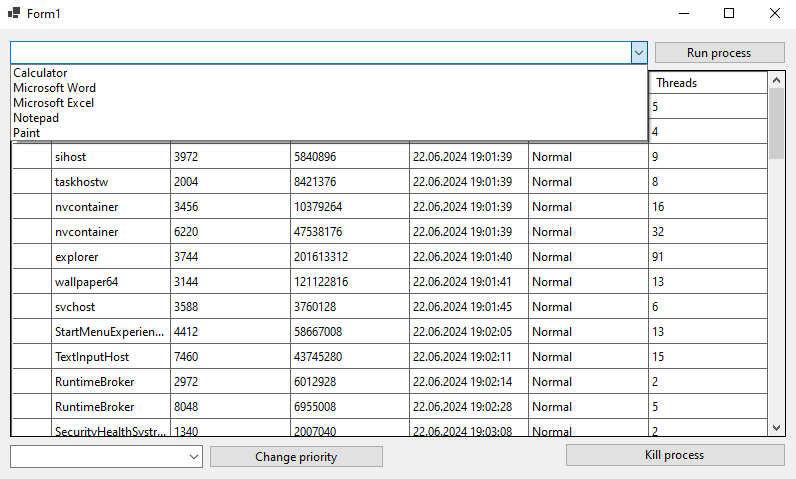
this.Close();

}

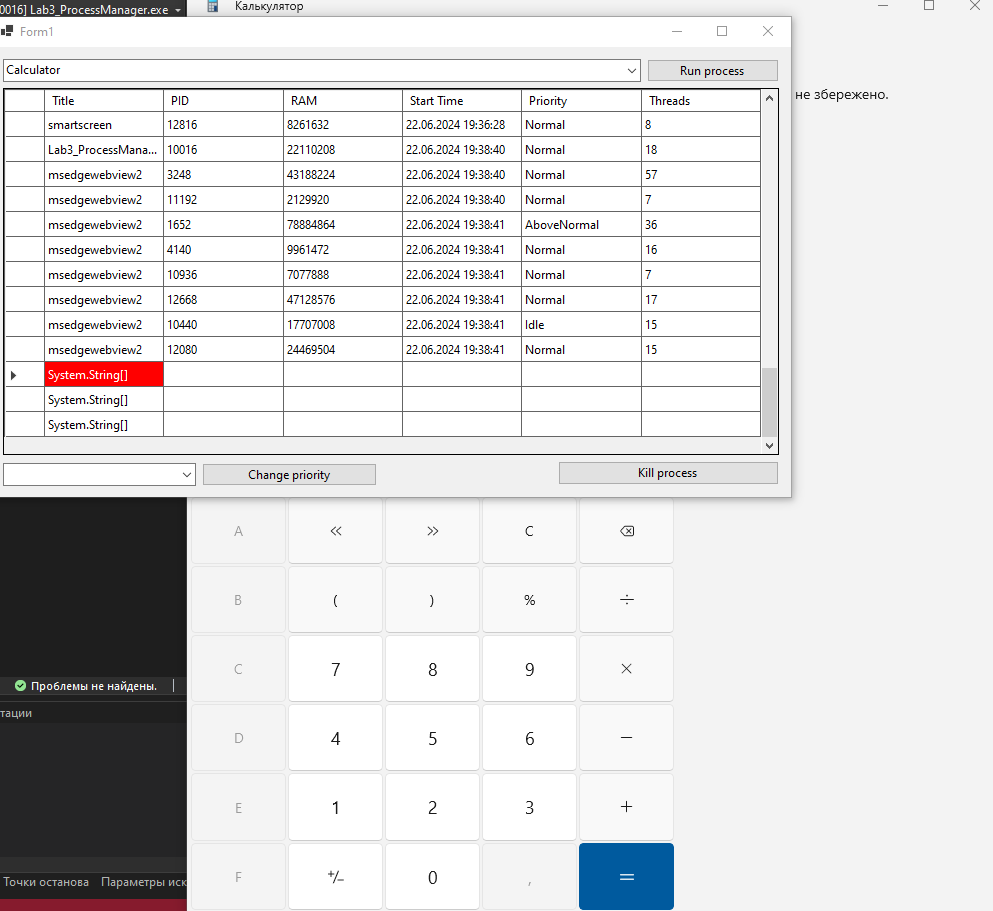
}

}

**Результат виконання програми процес менеджер:**



**Рис.1.**Вигляд форми



**Рис.2.**Відкриваємо Блокнот





**Рис.3-4.** Видалення Телеграму





**Рис.4 - 5.**Змінюємо пріорітет Телеграму

**Лістинг class MainForm:**

using System;

using System.ComponentModel;

using System.Diagnostics;

namespace Lab3\_ProcessManager

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

UpdateData();

}

List<Process> processes = new List<Process>();

private Process selectedProcess;

private void runProcBtn\_Click(object sender, EventArgs e)

{

try

{

var selectedProcess = selectProcessCbx.SelectedItem.ToString();

var process = ProcManager.RunProcessByName(selectedProcess);

processes.Add(process);

Thread.Sleep(500);

UpdateData();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return;

}

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

return;

var pid = Convert.ToInt32(dataGridView1.Rows[e.RowIndex].Cells[1].Value);

selectedProcess = processes.FirstOrDefault(p => p.Id == pid);

dataGridView1.Rows[e.RowIndex].Selected = true;

for (int i = 0; i < dataGridView1.Rows.Count; i++)

{

if (i != e.RowIndex)

dataGridView1.Rows[i].DefaultCellStyle.SelectionBackColor = Color.White;

}

dataGridView1.Rows[e.RowIndex].DefaultCellStyle.SelectionBackColor = Color.Red;

dataGridView1.Refresh();

}

private void killProcessBtn\_Click(object sender, EventArgs e)

{

if (selectedProcess == null)

{

MessageBox.Show("Please select a process to kill");

return;

}

var pid = selectedProcess.Id;

processes.Remove(selectedProcess);

dataGridView1.Rows.RemoveAt(dataGridView1.SelectedRows[0].Index);

ProcManager.Close(pid);

UpdateData();

}

private void UpdateData(object sender = null, EventArgs e = null)

{

processes = Process.GetProcesses().ToList();

if (dataGridView1.Rows.Count == 0)

{

foreach (var proc in processes)

{

try

{

dataGridView1.Rows.Add(proc.ProcessName, proc.Id, proc.PagedMemorySize, proc.StartTime, proc.PriorityClass.ToString(), proc.Threads.Count);

}

catch { }

}

return;

}

foreach (var proc in processes)

{

var row = dataGridView1.Rows.Cast<DataGridViewRow>().FirstOrDefault(r => Convert.ToInt32(r.Cells[1].Value) == proc.Id);

if (row == null)

{

try

{

dataGridView1.Rows.Add(ProcManager.GetProcessInfo(proc));

}

catch { }

continue;

}

try

{

row.Cells[0].Value = proc.ProcessName;

row.Cells[1].Value = proc.Id;

row.Cells[2].Value = proc.PagedMemorySize;

row.Cells[3].Value = proc.StartTime;

row.Cells[4].Value = proc.PriorityClass.ToString();

row.Cells[5].Value = proc.Threads.Count;

}

catch { }

}

for (int i = 0; i < dataGridView1.Rows.Count; i++)

{

var pid = Convert.ToInt32(dataGridView1.Rows[i].Cells[1].Value);

var proc = processes.FirstOrDefault(p => p.Id == pid);

if (proc == null)

{

dataGridView1.Rows.RemoveAt(i);

continue;

}

}

dataGridView1.Refresh();

}

private void changePriorityBtn\_Click(object sender, EventArgs e)

{

var newPriority = priorityCbx.SelectedItem.ToString();

if (selectedProcess == null)

{

MessageBox.Show("Please select a process to change priority");

return;

}

try

{

ProcManager.ChangePriority(selectedProcess.Id, newPriority);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return;

}

UpdateData();

}

}

}

**Лістинг ProcManager.cs:**

using System.Diagnostics;

namespace Lab3\_ProcessManager

{

class ProcManager

{

public static Process RunProcessByName(string processName)

{

switch (processName)

{

case "Notepad":

return StartNewProcess("notepad.exe");

case "Calculator":

return StartNewProcess("calc.exe");

case "Paint":

return StartNewProcess("mspaint.exe");

case "Microsoft Word":

return StartNewProcess("winword.exe");

case "Microsoft Excel":

return StartNewProcess("excel.exe");

default:

throw new Exception("Please select a correct process to run");

}

}

private static Process StartNewProcess(string processName)

{

var process = new Process();

process.StartInfo.FileName = "cmd.exe";

process.StartInfo.Arguments = "/c start " + processName;

process.Start();

return process;

}

public static void Close(int pid)

{

Process.GetProcessById(pid).Kill();

}

public static object GetProcessInfo(Process proc)

{

var x = new string[] { proc.ProcessName, proc.Id.ToString(), proc.PagedMemorySize.ToString(), proc.StartTime.ToString(), proc.PriorityClass.ToString(), proc.Threads.Count.ToString()};

return x;

}

public static void ChangePriority(int pid, string newPriority)

{

var selectedProcess = Process.GetProcessById(pid);

if (newPriority == "Normal")

selectedProcess.PriorityClass = ProcessPriorityClass.Normal;

else if (newPriority == "High")

selectedProcess.PriorityClass = ProcessPriorityClass.High;

else if (newPriority == "Low")

selectedProcess.PriorityClass = ProcessPriorityClass.Idle;

else if (newPriority == "Realtime")

selectedProcess.PriorityClass = ProcessPriorityClass.RealTime;

else if (newPriority == "Above normal")

selectedProcess.PriorityClass = ProcessPriorityClass.AboveNormal;

else if (newPriority == "Below normal")

selectedProcess.PriorityClass = ProcessPriorityClass.BelowNormal;

else

{

throw new Exception("Please select a correct priority");

}

}

}

}

Посилання на репозиторій: <https://gitlab.com/ipz22-4/dotnetlab3>

**Висновок:** навчився працювати з потоками та процесами у мові C#.