**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).

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.

Вимоги до програми шифрування файлів

1. Графічний інтерфейс користувача з можливістю вводу ключа для

шифрування та діалогу вибору довільного файлу, який буде шифруватись

або розшифровуватись.

2. Здатність працювати з файлами довільного розміру та формату.

3. В процесі шифрування, повинен відображатись індикатор прогресу (0-100%)

та час, який пройшов від запуску шифрування.

4. По завершенні шифрування повинне бути відображене вікно з інформацією

про розмір зашифрованого файла, його назву та час, затрачений на

шифрування.

5. Необхідно запобігти «підвисанню» вікна при здійсненні операції шифрування

(шифрування здійснюватись в окремому обчислювальному потоці за

допомогою класу BackgroundWorker) .

6. Коректна обробка виключень, що можуть виникати під час роботи.

7. Для успішного виконання роботи необхідно ознайомитися з наступними

темами: асинхронні операції (клас BackgroundWorker), класи File, FileStream,

Timer, DateTime, OpenFileDialog, SaveFileDialog, обробка виключень.

Висновок: Я навчився використовувати методи розширення та узагальнення у мові програмування C#.

Можливість запуску програм: Калькулятор, Microsoft Word та ще трьох на

ваш вибір.

2. Перегляд список запущених процесів з інформацією про обсяг оперативної

пам'яті, виділений під процес, часу, коли був запущений процес, пріоритет

процесу, кількість потоків, які запущені процесом. Відобразити у вигляді

таблиці.

3. Можливість зупинки вибраного процесу.

4. Можливість зміни пріоритету вибраного процесу.

1. **шифрування файлів;**

**Лістинг бібліотеки Сrypt:**

public class Crypt

{

public string Filepath { private set; get; }

public string Password { private set; get; }

public Crypt(string path,string pass)

{

Filepath = path;

Password = pass;

}

public Crypt()

{

}

public float FileSize()

{

FileInfo file = new FileInfo(Filepath);

return file.Length;

}

public event EventHandler<EventHandlerProgres> Progress;

protected void InProgress(int percent)

{

if (Progress != null)

Progress(this, new EventHandlerProgres(percent));

}

public event EventHandler<EventHandlerCodeTime> Timer;

protected void InTime(int strHour, int strMin, int strSec, int milisec)

{

if (Timer != null)

Timer(this, new EventHandlerCodeTime(strHour, strMin, strSec, milisec));

}

public void CryptFile()

{

Stopwatch watch = new Stopwatch();

BackgroundWorker worker = new BackgroundWorker();

worker.DoWork += (o, e) =>

{

using (FileStream flstrIN = File.OpenRead(Filepath))

using (FileStream flstrOUT = File.OpenWrite(Filepath + ".coded"))

{

watch.Start();

int symbol;

int counter = 1;

int PasPosition = 0;

while ((symbol = flstrIN.ReadByte()) != -1)

{

symbol = symbol ^ Password[PasPosition];

PasPosition++;

flstrOUT.WriteByte((byte)symbol);

if (PasPosition >= Password.Length) PasPosition = 0;

InProgress((int)((counter \* 1.0) / flstrIN.Length \* 100));

counter++;

}

}

watch.Stop();

InTime(watch.Elapsed.Hours, watch.Elapsed.Minutes, watch.Elapsed.Seconds, watch.Elapsed.Milliseconds);

};

worker.RunWorkerAsync();

}

public void DecriptFile()

{

BackgroundWorker worker = new BackgroundWorker();

worker.DoWork += (o, e) =>

{

using (FileStream flstrIN = File.OpenRead(Filepath))

{

Filepath = Filepath.Remove(Filepath.Length - 6);

using (FileStream flstrOUT = File.OpenWrite(Filepath + ".decoded"))

{

int symbol;

int counter = 1;

int PasPosition = 0;

while ((symbol = flstrIN.ReadByte()) != -1)

{

symbol = symbol ^ Password[PasPosition];

PasPosition++;

flstrOUT.WriteByte((byte)symbol);

if (PasPosition >= Password.Length) PasPosition = 0;

InProgress((int)((counter \* 1.0) / flstrIN.Length \* 100));

counter++;

}

}

}

};

worker.RunWorkerAsync();

//flstrIN.Close();

//flstrOUT.Close();

}

}

**Лістинг EventTime:**

public class EventHandlerCodeTime:EventArgs

{

public float timeSeconds { set; get; }

public int timeMinutes { set; get; }

public int timeHours { set; get; }

public int timeMilisec { set; get; }

public string totalTimeString { set; get; }

public EventHandlerCodeTime(int strHour, int strMin, int strSec, int milisec)

{

timeHours = strHour;

timeMinutes = strMin;

timeSeconds = strSec;

timeMilisec = milisec;

if (timeHours == 0 && timeMinutes != 0)

{

totalTimeString = $"{timeMinutes}:{timeSeconds} sec.";

}

else if (timeHours == 0 && timeMinutes == 0)

{

totalTimeString = $"{timeSeconds}.{timeMilisec} sec.";

}

}

}

**Лістинг Program:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using ArchivatorLibrary;

using CodeLibrary;

namespace WinForms

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private string file;

private float filesize;

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

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "всі файли (\*.\*)|\*.\*";

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

{

file = openFileDialog.FileName;

removeToolStripMenuItem.Visible = true;

decodeToolStripMenuItem.Visible = true;

codeToolStripMenuItem.Visible = true;

labelProgress.Visible = false;

panelTimer.Visible = false;

labelHome.Text = "Take Action";

}

}

private bool ErrorExceptionPass()

{

bool flag = true;

if (textBoxPass.Text.Length == 0)

{

MessageBox.Show("Password can't be a null");

flag = false;

}

return flag;

}

private void ProgressBarUpdateAction(object sender, EventHandlerProgres e)

{

this.Invoke(

new Action(() =>

{

if (e.progress == 100)

{

labelHome.Text = "Crypt Succsessful";

labelHome.Visible = true;

progressBar.Visible = false;

openToolStripMenuItem.Visible = true;

}

progressBar.Value = e.progress;

labelProgress.Text = e.progress.ToString() + "%";

}));

}

private void Timer(object sender, EventHandlerCodeTime e)

{

this.Invoke(

new Action(() =>

{

panelTimer.Visible = true;

labelTimer.Text = e.totalTimeString;

labelSizeFile.Text = filesize.ToString()+"Bites";

}));

}

int flag;

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

{

labelHome.Text = "Coding...";

panelPass.Visible = true;

decodeToolStripMenuItem.Visible = false;

flag = 1;

}

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

{

labelHome.Text = "Decoding...";

panelPass.Visible = true;

flag = 0;

codeToolStripMenuItem.Visible = false;

}

private void Form1\_Load(object sender, EventArgs e)

{

panelPass.Visible = false;

}

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

{

if (!ErrorExceptionPass())

return;

Crypt crypt = new Crypt(file, textBoxPass.Text);

crypt.Progress += ProgressBarUpdateAction;

crypt.Timer += Timer;

filesize = (float)(crypt.FileSize());

labelProgress.Visible = true;

if (flag == 1)

crypt.CryptFile();

if (flag == 0)

crypt.DecriptFile();

panelPass.Visible = false;

removeToolStripMenuItem.Visible = false;

decodeToolStripMenuItem.Visible = false;

codeToolStripMenuItem.Visible = false;

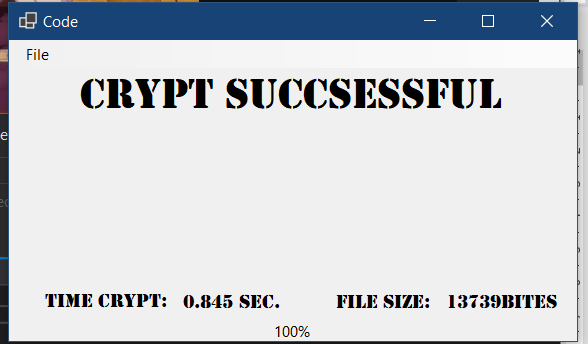
openToolStripMenuItem.Visible = false;

}

}

}

**Результат:**



**2)Менеджер процесів**

**Лістинг Programe:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Reflection.Metadata;

namespace ProcessApplication

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private int RowPosition;

private Process[] processes;

private int ProcessDataGridId;

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

{

panelInfo.Visible = false;

}

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

{

panelStart.Visible = false;

panelInfo.Visible = true;

this.Width = 818;

this.Height = 511;

GetProcesses();

}

private void GetProcesses()

{

processes = Process.GetProcesses();

dataGridView1.RowCount = processes.Length;

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

{

double mem = processes[i].PagedMemorySize64 / 1048576.0;

dataGridView1.Rows[i].HeaderCell.Value = i.ToString();

dataGridView1[0, i].Value = processes[i].ProcessName;

dataGridView1[1, i].Value = $"{mem:F1}";

//dataGridView1[2, i].Value = processes[i].StartTime; ///no access

dataGridView1[3, i].Value = ReturnPriorityType(processes[i]);

dataGridView1[4, i].Value = processes[i].Threads.Count;

}

}

private void Form1\_Load(object sender, EventArgs e)

{

this.Width = 694;

this.Height = 164;

dataGridView1.RowHeadersWidth = 65;

dataGridView1.MouseClick += new MouseEventHandler(DataGrid\_MouseClick);

}

private void DataGrid\_MouseClick(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Right)

{

ContextMenuStrip myMenu = new ContextMenuStrip();

RowPosition = dataGridView1.HitTest(e.X, e.Y).RowIndex;

string StrProcessDataGridId = dataGridView1.Rows[RowPosition].HeaderCell.Value.ToString();

//коли відбувається впорядковування таблиці по значенню стовпця, звичайна позиція RowPosition не може бути використана.

ProcessDataGridId = int.Parse(StrProcessDataGridId);

dataGridView1.ClearSelection();

dataGridView1.Rows[RowPosition].Selected = true;

if (ProcessDataGridId >= 0)

{

myMenu.Items.Add("Kill Process").Name = "Kill Process";

myMenu.Items.Add("Priority Normal").Name = "Priority Normal";

myMenu.Items.Add("Priority High").Name = "Priority High";

}

myMenu.Show(dataGridView1, new Point(e.X, e.Y));

myMenu.ItemClicked += new ToolStripItemClickedEventHandler(myMenu\_ItemClicked);

}

}

private void myMenu\_ItemClicked(object sender, ToolStripItemClickedEventArgs e)

{

switch (e.ClickedItem.Name.ToString())

{

case "Kill Process":

MessageBox.Show(e.ClickedItem.Name.ToString());

MessageBox.Show(ProcessDataGridId.ToString());

MessageBox.Show(processes[ProcessDataGridId].ProcessName.ToString());

processes[ProcessDataGridId].Kill();

GetProcesses();

break;

case "Priority Normal":

processes[ProcessDataGridId].PriorityClass = ProcessPriorityClass.Normal;

GetProcesses();

break;

case "Priority High":

processes[ProcessDataGridId].PriorityClass = ProcessPriorityClass.High;

GetProcesses();

break;

}

}

private string ReturnPriorityType(Process process)

{

Dictionary<int, string> prior = new Dictionary<int, string>()

{

{0, "0"},

{4, "Idle"},

{5, "Idle"},

{6, "Idle"},

{7, "Idle"},

{8, "Normal"},

{9, "Normal"},

{10, "Normal"},

{11, "Normal"},

{12, "Normal"},

{13, "High"},

{24, "RealTime"}

};

string str = process.BasePriority.ToString();

return prior[Int32.Parse(str)];

}

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

{

Process.Start("calc");

labelStart.Text = "Calculator start";

}

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

{

Process.Start(@"C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE");

labelStart.Text = "Microsoft Word started";

}

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

{

Process.Start(@"C:\Program Files\Microsoft Office\root\Office16\EXCEL.EXE");

labelStart.Text = "Microsoft Exel started";

}

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

{

Process.Start(@"C:\Program Files\Microsoft Office\root\Office16\POWERPNT.EXE");

labelStart.Text = "Microsoft PowerPoint started";

}

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

{

Process.Start(@"C:\Program Files\Microsoft Office\root\Office16\MSACCESS.EXE");

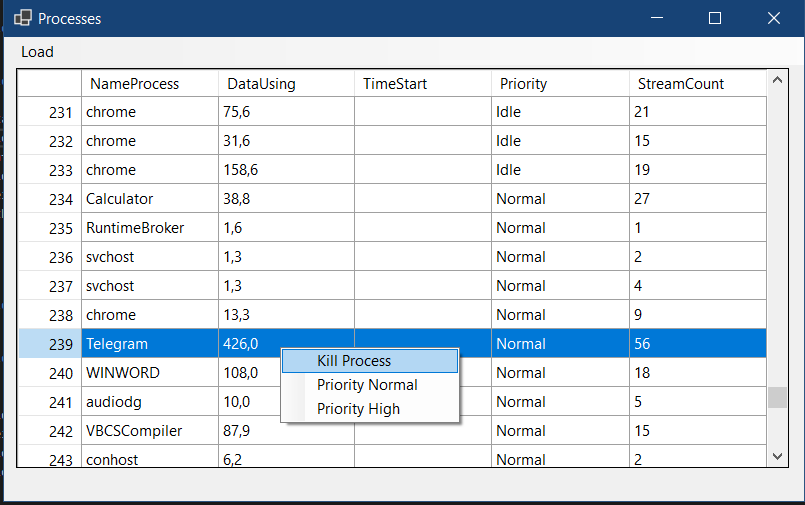
labelStart.Text = "Microsoft Access started";

}

}

}

**Результат:**



**Посилання на git:**

**https://gitlab.com/ipz212\_bvm/dotnetlab2**