МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Д.В. Настенко,
О.І. Буханенко,
А.А. Марченко

Об'єктно-орієнтоване програмування:

Лабораторний практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Управління, захист та автоматизація енергосистем»
спеціальності 141 Електроенергетика, електротехніка та електромеханіка

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Об'єктно-орієнтоване програмування. Лабораторний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» / КПІ ім. Ігоря Сікорського; уклад.: Д.В. Настенко, О.І. Буханенко , А.А. Марченко– Електронні текстові данні (1 файл, pdf: 608 КБ). – Київ: КПІ ім. Ігоря Сікорського, 2022. – 51с.

Відповідальний

Редактор О.С. Яндульський, професор, д.т.н.

Рецензент Т.Л. Кацадзе, канд. техн. наук

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 6 від 24.06.2022 р.)
за поданням Вченої ради Факультету електроенерготехніки та автоматики
(протокол № 9 від 17.05.2022 р.)

Посібник містить матеріали 9-ти занять, які включають в себе теоретичні відомості та приклади для вирішення інженерних завдань у сфері енергопостачання за допомогою алгоритмічної мови C#: розробляти програмні проекти для комп'ютерів на основі використання технології об'єктно-орієнтованого програмування, застосовувати потоки даних, створювати багатопоточні програмні додатки в галузі електроенергетики. Для кожного заняття приведено індивідуальні завдання для студентів, виконання яких дозволить закріпити знання з використанням сучасної алгоритмічної мови програмування C#.

Реєстр. № 21/22-734. Обсяг 51 стор.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

ВСТУП

Розвиток економіки, промисловості, науки і техніки, сфери освіти сьогодні значною мірою залежить від масового запровадження та використання обчислювальної техніки. Це вимагає підготовки і перепідготовки фахівців з програмування і використання персональних комп'ютерів.

Вибір мови програмування C# пояснюється такими чинниками:

- простотою і природністю основних конструкцій мови, що дозволяє швидко її освоїти і створювати алгоритмічно складні програми;
- можливістю використання розвинених засобів подання структур даних, що забезпечує зручність роботи як з числовою, так і з символьною інформацією;
- відповідністю принципам об'єктно-орієнтованого програмування, що робить програми наочними;
- наявністю бібліотеки процедур і функцій для роботи як з текстовою, так і з графічною інформацією, що дозволяє створювати досить складні програми.

ВИМОГИ БЕЗПЕКИ ПІД ЧАС РОБОТИ В ЛАБОРАТОРІЇ ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

- Ці правила є обов'язковими для всіх студентів та інших осіб, які працюють в лабораторіях постійно чи тимчасово.
- Перевірка знань цих правил проводиться:
    - обслуговуючого персоналу – завідуючим лабораторією;
    - студентів – викладачами, які є керівниками лабораторних робіт.
- Після перевірки знань цих правил, кожен з тих, хто працює в лабораторії ставить свій підпис в спеціальному журналі. Без цього підпису ніхто до роботи в лабораторії не повинен бути допущеним.
- Дотримання правил з техніки безпеки повинно гуртуватися на високій свідомості всіх, хто працює в лабораторіях. Кожен, хто помітить порушення правил, а також несправність, яка являє собою небезпеку для людей і обладнання, повинен сповістити про це керівника.
- Робота в лабораторії по виконанню конкретного завдання може проводитися тільки після ретельного ознайомлення студентів з обладнанням і роботою, та чіткого уявлення про те, які елементи установки будуть під напругою та дотик до яких є небезпечними у стані роботи.
- Забороняється виконання ремонтних робіт на обладнанні, яке знаходиться під напругою.
- Забороняється наближатися або торкатися до струмоведучих частин, що обертаються, або усувати несправності без відключення установок.
- Забороняється проводити переключення в схемі, яка знаходиться під напругою.
- Перевірку наявності напруги дозволяється проводити тільки за допомогою вольтметра.
- Апарати управління та вимірювальні прилади слід розташовувати так, щоб було зручно вести спостереження за приладами, не перегинаючись через проводи та апарати.
- У випадку виникнення будь-яких несправностей необхідно негайно

вимкнути живленим установки та сповістити керівника занять про це.

- Кнопки управління, рубильники встановлювати в легкодоступних місцях для швидкого виключення схеми.
- У випадку припинення подачі електроенергії в лабораторію всі установки в лабораторії обов'язково вимикаються на робочих місцях.
- В лабораторіях категорично забороняється:
    - Палити в усіх приміщеннях, крім спеціально відведених для цього місць;
    - Прокладати без дозволу постійні та тимчасові лінії;
    - Користуватися побутовими електронагрівальними приладами;
    - Користуватися зіпсованим електрообладнанням, саморобними запобіжниками, провідниками із зіпсованою ізоляцією та саморобними електросвітильниками;
    - Проводити в непристосованих приміщеннях обмивку та фарбування деталей горючими рідинами та фарбниками;
    - Зберігати паливно-мастильні матеріали, хімікати та інші горючі речовини;
    - Загромаджувати проходи в лабораторіях;
- Всі, хто працює в лабораторії повинні знати, де знаходиться аптечка з медикаментами для наданим першої допомоги.
- При ураженні людини електричним струмом треба негайно вимкнути напругу, надати першу допомогу та покликати лікаря.
- Порятунок осіб, які постраждали, залежить від тою, як швидко вони будуть звільнені від електричного струму та як швидко їм буде надано першу допомогу.
- Першу допомогу необхідно надати негайно на місці події.
- Переносити людину, яка постраждала в інше місце необхідно тільки в тих випадках, коли небезпека продовжує загрожувати або надання допомоги на місці неможливе.
- При відсутності у постраждалого дихання, серцебиття, пульсу ніколи не треба ставити під сумнів необхідність першої допомоги, тому що при

ураженні електричним струмом смерть часто буває несправжньою. Тільки лікар може дати висновок про смерть постраждалого.

- До приїзду лікаря постраждалому необхідно надати допомогу і провести штучне дихання з дотриманням всіх правил надання першої допомоги.
- Про випадок негайно треба сповістити керівництво кафедри, деканату та інституту.
- Недотримання цих вимог не дозволяється. Якщо розпорядження суперечить діючим правилам, необхідно надати роз'яснення з приводу неухильною виконання цих правил і довести це до відома керівництва.

ВИМОГИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Підготовка до кожної роботи проводиться студентами самостійно в поза аудиторний час. Студенти ознайомлюються із теоретичними відомостями, пишуть необхідні програми відповідно до отриманого індивідуального завдання на роботу.

Усі лабораторні роботи виконуються на мові програмування C#. Під час занять студенти проводять тестування написаних програм, тобто запускають їх в інтегрованому середовищі розробки на персональних комп'ютерах, займаються налагодженням і виконують необхідні розрахунки. Після виконання роботи студент оформляє звіт, який складається з таких розділів:

1. Назва, тема, мета роботи, стислі теоретичні відомості
2. Індивідуальне завдання
3. Текст програми
4. Результати виконання програми

Робота оформлюється в друкованому вигляді на папері стандартного формату А4.

Під час захисту роботи необхідно відповісти на контрольні питання і вміти пояснити роботу програми.

ЛАБОРАТОРНА РОБОТА №1.
ПОНЯТТЯ ПОТОКІВ. КЛАС THREAD.

Мета роботи – навчитися створювати багатопоточні програмні додатки.

1.1. Теоретичні відомості

При розробці програми для комп'ютерів з одним або декількома процесорами може знадобитися, щоб додаток забезпечувало взаємодія з користувачем, навіть якщо це додаток зараз зайнято іншими діями. Одним із способів, що забезпечують взаємодію програми і користувача в процесі обробки даних додатком інших подій, є використання декількох потоків виконання. Оскільки в цьому розділі представлені основні принципи роботи з потоками, увага буде зосереджена на принципах керованих потоків і їх використанні.

Переваги багатопоточності

Використання більше одного потоку є найбільш потужним засобом збільшення швидкості відповіді системи на дії користувача, а також засобом обробки даних, необхідних для одночасного виконання завдання. Для обробки даних у фоновому режимі на комп'ютері з одним процесором цей ефект досягається шляхом використання невеликих періодів часу між подіями користувача. Наприклад, поки користувач вносить зміни в електронну таблицю, інший потік в цьому додатку може перераховувати інші частини цієї таблиці.

Продуктивність такого додатка на комп'ютері з кількома процесорами буде істотно вище. Для виконання таких завдань в домені додатку необхідно використовувати декілька потоків.

- Взаємодія з мережі з веб-сервером і базою даних.
- Виконання операцій, що займають багато часу.
- Поділ завдань за пріоритетами. Наприклад, потік з високим пріоритетом відповідає за термінові завдання, з низьким пріоритетом - за всі інші.
- Збереження можливості відповіді для інтерфейсу користувача при виділенні часу для фонової завдання.

Недоліки багатопоточності

Для збільшення продуктивності та зменшення кількості використовуваних ресурсів системи рекомендується використовувати якомога менше потоків. Крім того, при використанні потоків необхідно враховувати вимоги до ресурсів і можливість виникнення помилок при розробці програми. Існують наступні вимоги до ресурсів.

- Пам'ять, яка використовується системою для обробки контекстних даних, необхідних у процесах, об'єктах AppDomain і потоках. Тому кількість створюваних процесів, об'єктів AppDomain і потоків обмежено обсягом доступної пам'яті.
- Час процесора, що використовується для відстеження кількості потоків. Якщо кількість потоків велике, більшість з них не буде працювати належним чином. Якщо більшість поточних потоків знаходиться в одному процесі, потоки інших процесів плануються рідше.
- Контроль виконання коду з великою кількістю потоків досить складний і може бути причиною помилок.
- При знищенні потоків необхідно враховувати можливі проблеми і передбачити способи їх вирішення.

Синхронізація даних

При спільному доступі до ресурсів можуть виникати конфлікти. Для їх запобігання треба синхронізувати або контролювати доступ до загальних ресурсів. Помилка синхронізації доступу (в одному або різних доменах програми) може привести до зависання (потік перестає виконуватися, очікуючи завершення роботи іншого потік, який, в свою чергу, чекає завершення роботи першого потоку) і станам гонки (виникнення непередбачених результатів через невірну тимчасової залежності між двома подіями). Система містить синхронізуючі об'єкти, що використовуються для управління розподілом ресурсів між потоками. Синхронізація ресурсів спрощується, якщо кількість потоків невелика.

Ресурси, що вимагають синхронізації:

- системні ресурси (послідовні порти);
- ресурси, що використовуються декількома процесами (дескриптори файлів);
- ресурси окремого домену додатки (глобальні, статичні поля або поля примірників), до яких звертаються кілька потоків.

Клас System.Threading

Основний функціонал для використання потоків в програмі надається простором **System.Threading**. В ньому визначено клас, що описує окремий потік – це клас **Thread**.

Thread (нитка, павутинка) – створює і контролює потік, задає пріоритет і повертає стан потоку.

Основні властивості та методи

- Статична властивість **CurrentThread** – повертає посилання на поточний потік, що виконується в даний момент.
- Властивість **Name** – дозволяє встановлювати та змінювати ім'я потоку.
- Статичний метод **Sleep** зупиняє потік на певну кількість мілісекунд.
- Метод **Start** - запускає потік на виконання.

Щоб запустити потік потрібно в конструкторі класу **Thread** вказати делегат, що вказує на метод який буде виконуватись. Потім необхідно викликати у створеного потоку метод **Start**.

Якщо параметри в потік передавати не потрібно, то делегат матиме наступну сигнатуру:

```
public delegate void ThreadStart();
```

Наприклад:

```
using System;
using System.Threading;

namespace ThreadEx {
    class Program {
        static void Test() {
            for (int i = 0; i < 10; i++) {
                Console.WriteLine(Thread.CurrentThread.Name + ":" + i);
                Thread.Sleep(10);
```

```
            }
        }
        static void Main(string[] args) {
            Thread.CurrentThread.Name = "Потік 0";
            Thread tr1 = new Thread(Program.Test);
            tr1.Name = "Потік 1";
            Thread tr2 = new Thread(Program.Test);
            tr2.Name = "Потік 2";
            tr1.Start();
            tr2.Start();
            Test();
            Console.ReadKey();
        }
    }
}
```

Якщо в потік потрібно передати параметр, то делегат матиме наступну сигнатуру:

```
public delegate void ParameterizedThreadStart(object obj);
```

і в метод **Start** потрібно буде передати об'єкт параметр.

```
using System;
using System.Threading;

namespace ThreadParam {
    class Param {
        internal int time;
        internal ConsoleColor color;
        internal Param(int time, ConsoleColor color) {
            this.time = time;
            this.color = color;
        }
    }
    class Program {
        static void Test(object o) {
            Param p = o as Param;
            if (p == null) return;
            for (int i = 0; i < 10; i++) {
                Console.ForegroundColor = p.color;
                Console.WriteLine(Thread.CurrentThread.Name + ":" + i);
                Thread.Sleep(p.time);
            }
        }
        static void Main(string[] args) {
            Thread.CurrentThread.Name = "Потік 0";
            Thread tr1 = new Thread(Program.Test);
            tr1.Name = "Потік 1";
            Thread tr2 = new Thread(Program.Test);
            tr2.Name = "Потік 2";

            tr1.Start(new Param(100, ConsoleColor.Red));
            tr2.Start(new Param(200, ConsoleColor.Green));
            Test(new Param(300, ConsoleColor.Blue));
            Console.ReadKey();
        }
    }
}
```

1.2. Порядок виконання роботи

1. **Ознайомитись** із теоретичним матеріалом.
2. Розглянути та **описати в протоколі** основні властивості та методи класу **`Thread`**.
3. Написати програму на мові C#, яка б наочно демонструвала паралельну роботу 3 потоків. Для цього в кожний з потоків необхідно включити цикл виводу з затримкою (**`Thread.Sleep(prm)`**). Розглянути роботу потоків при різних значеннях параметру **`prm.`**
4. Доповнити створену програму прикладом використання параметризованих потоків.
5. Підготувати звіт по роботі.
6. В звіт включити отримані на екрані результати.

1.3 Контрольні питання

1. Що таке багатопоточність?
2. Які переваги та недоліки багатопоточності?
3. Для чого необхідна синхронізація даних при використанні багатопоточності?
4. Назвіть основні властивості та методи класу System.Threading

ЛАБОРАТОРНА РОБОТА №2.
ПОТОКИ STREAM.

2.1. Теоретичні відомості

Stream (потік, джерело, течія, річка) – з простору імен System.IO, є абстрактним базовим класом для всіх потоків читання та запису. Потік - це абстракція послідовності байтів, наприклад файлу, пристрою вводу-виводу, міжпроцесного каналу зв'язку або сокету TCP/IP. Клас Stream та похідні від нього класи забезпечують універсальне подання різних типів вводу та виводу і віддаляють програміста від конкретних особливостей операційної системи і базових пристроїв.

Потоки включають три основні операції:

- Можливість читання з потоку. Читання — це перенос даних з потоку в структуру даних, наприклад масив байтів.
- Можливість запису в потік. Це передача даних зі структури даних в потік.
- Потоки можуть підтримувати пошук. Пошук включає в себе запит та зміну поточної позиції в середині потоку. Наявність пошуку залежить від типу збереження даних потоку. Наприклад в мережевих потоках відсутнє стандартне представлення поточного положення і тому, зазвичай, не підтримується пошук.

В залежності від базового джерела даних або репозиторія потоки можуть підтримувати тільки деякі з цих можливостей. Про наявність цих можливостей можна дізнатися за допомогою властивостей CanRead, CanWrite та CanSeek класу Stream.

Репозито́рій - це місце, де зберігаються й підтримуються які-небудь дані.

Для читання та запису даних різних форматів в потоках найчастіше використовуються методи Read і Write. Для потоків, що підтримують пошук,

використовуються методи Seek і SetLength та властивості Position і Length, які використовуються щоб дізнаватися та змінювати поточне положення та довжину потоку.

Клас Stream є нащадком інтерфейсу IDisposable. Це означає що по закінченні роботи з потоком, обов'язково необхідно звільнити його напряму безпосередньо або опосередковано. Щоб звільнити потік безпосередньо потрібно викликати його метод Close або Dispose в блоці try / catch / finnaly. Щоб звільнити опосередковано - в C# використовуйте язикову конструкцію using (). Dispose також звільняє ресурси операційної системи, такі як дескриптори файлів, мережеві підключення або пам'ять задіяну для внутрішньої буферизації.

Для класу Stream важливим також є метод Flush – що очищує буфери потоку і викликає запис всіх буферизованих (кешованих) даних.

Видалення об'єкту класу Stream також виконує запис і очищення всіх буферизованих даних і по суті викликає метод Flush.

Найпоширеніші нащадки класу Stream – є FileStream, MemoryStream та NetworkStream.

Клас **FileStream** – надає реалізацію Stream для файлу, і підтримує синхронні та асинхронні операції читання запису.

Простір імен: System.IO

Наслідування:

Object->MarshalByRefObject->Stream->FileStream

Клас **NetworkStream** - надає реалізацію Stream для читання та запису в мережі.

Простір імен: System.Net.Sockets

Наслідування:

Object->MarshalByRefObject->StreamNetwork->Stream

Для роботи з різними кодуваннями файлів є спеціальні класи.

Класи BinaryReader і BinaryWriter виконують читання та запис кодованих рядків та простих типів даних в потоки.

Наслідування:

Object->BinaryReader

Клас StreamReader зчитує символи з потоків, використовуючи Encoding для перетворення символів в байти, та навпаки. StreamReader має конструктор, що намагається з'ясувати кодування потоку, на основі наявності специфічної для кодування преамбули, такої як метка порядку байтів.

Object->MarshalByRefObject->TextReader->StreamReader

Клас StreamWriter записує символи в потоки, використовуючи Encoding для перетворення символів в байти.

TextReader є абстрактним базовим класом для класу StringReader і класу StreamReader. В то час як реалізації абстрактного класу Stream призначені для побайтового вводу і виводу, реалізації TextReader призначені для виводу знаків в кодуванні Unicode.

TextWriter є абстрактним базовим класом для класу StringWriter і класу StreamWriter. В то час як реалізації абстрактного класу Stream призначені для побайтового вводу і виводу, реалізації TextWriter призначені для вводу знаків в кодуванні Unicode.

Клас **MarshalByRefObject** – дозволяє доступ до об'єктів за межами доменів в програмах, що підтримують віддалену взаємодію.

Приклад копіювання файлу, за допомогою FileStream, з використанням буферу:

```
using System;
using System.IO;

namespace StreamExample {
    class Program {
        static void Main(string[] args) {
            string fileNameOld = @"D:\Tmp\222.jpg";
            string fileNameNew = @"D:\Tmp\444.jpg";
            byte[] buff = new byte[1024];
            try {
                using (FileStream read = new FileStream(fileNameOld, FileMode.Open)) {
                    using (FileStream write =
                          new FileStream(fileNameNew, FileMode.Create)) {
                        int iRead = 0;
```

```
                    int iSize = 0;
                    do {
                        iRead = read.Read(buff, 0, buff.Length);
                        if (iRead <= 0) break;
                        write.Write(buff, 0, iRead);
                        iSize += iRead;
                    } while (true);
                    Console.WriteLine("File : {0} Size : {1} bytes",
                      fileNameNew, iSize);
                }
            }
        } catch (Exception ex) {
            Console.WriteLine("Error : " + ex.Message);
        }
        Console.ReadKey();
    }
  }
}
```

2.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Описати основні властивості та методи класів FileStream, BinaryReader, BinaryWriter, StreamReader, StreamWriter, TextReader та TextWriter.
3. Написати програму, що б виконувала паралельне копіювання кількох файлів за допомогою операцій читання/запису з використанням методів класу FileStream. При цьому в консоль потрібно виводить відсоток виконання по кожному з файлів. Для наочності файли для копіювання взяти великі – порядку кількох Гб.
4. Дослідити як впливає розмір буферу на швидкість копіювання.
5. Налагодити програму на комп’ютері.
6. Підготувати звіт по роботі.

2.3 Контрольні запитання

1. Що таке потоки Stream?
2. Які можливості включають в себе потоки Stream?
3. Назвіть основні методи класу Stream.
4. Яке призначення класів FileStream, MemoryStream та NetworkStream?
5. Як впливає розмір буферу FileStream на швидківсть копіювання файлів?

ЛАБОРАТОРНА РОБОТА №3.
РОБОТА З КОНТРОЛАМИ ФОРМ З РІЗНИХ ПОТОКІВ.

Мета роботи – навчитися працювати з елементами керування форм з різних потоків.

3.1. Теоретичні відомості

Здійснення потокобезпечних викликів елементів управління Windows Forms.

При використанні багатопоточності для поліпшення продуктивності додатків Windows Forms під час виклику елементів управління необхідно дотримуватися принципів безпеки потоків.

Доступ до елементів управління Windows Forms по суті не є потокобезпечна. При наявності двох або більше потоків, контролюючих стан елемента керування, останній може виявитися в неузгодженому стані. Крім того, можуть виникнути інші помилки, пов'язані з потоками, включаючи стану гонки і взаємоблокування. Важливо забезпечити потокобезпечна доступ до елементів управління.

Виклик елемента управління з будь-яких інших потоків, за винятком потоку, в якому був створений елемент управління, без використання методу **`Invoke`** є порушенням безпеки. Далі наведено приклад виклику, виконуваного з порушенням принципів потокобезпеки.

```
// This event handler creates a thread that calls a
// Windows Forms control in an unsafe way.
private void setTextUnsafeBtn_Click(
      object sender,
      EventArgs e){
      this.demoThread = new Thread(new ThreadStart(this.ThreadProcUnsafe));
      this.demoThread.Start();
}
// This method is executed on the worker thread and makes
// an unsafe call on the TextBox control.
private void ThreadProcUnsafe(){
      this.textBox1.Text = "This text was set unsafely.";
}
```

NET Framework дозволяє виявити звернення до елементів керування, що не являються потокобезпечними. При виконанні програми у відладчику і намаганні виклику елемента керування з потоку, відмінного від того що створив цей елемент керування, у відладчику створюється виняток InvalidOperationException з повідомленням "Звернення до елемента керування <<ім'я_елемента_керування>> з потоку, що не є творцем цього елемента керування".

Це виключення завжди виникає під час налагодження, а в де-яких випадках і під час виконання. Цю проблему можна розв'язати за допомогою потокобезпечного звернення до елемента.

Порядок виконання потокобезпечного виклику елемента керування Windows Forms:

1. Опитайте властивість InvokeRequired елемента керування .
2. Якщо InvokeRequired повертає значення true, необхідно викликати метод Invoke з делегатом, що фактично викликатиме елемент керування.
3. Якщо InvokeRequired повертає false, звертайтеся до елемента керування безпосередньо.

Наприклад:

```
delegate void SetTextCallback(string text);
private void SetText(string text){
      if(this.richTextBox1.InvokeRequired) {
            SetTextCallback d =
                  new SetTextCallback(SetText);
        try {
            this.Invoke(d, new object[] { text });
        } catch(Exception ex) {
            Trace.WriteLine(ex.Message);
        }
    } else {
        this.richTextBox1.Text += '\n' + text;
    }
}
```

Елемент керування `RichTextBox` – управління кольором виводу.

Розглянемо форму з двома елементами керування richTextBox1 та button1 (рис 3.1).

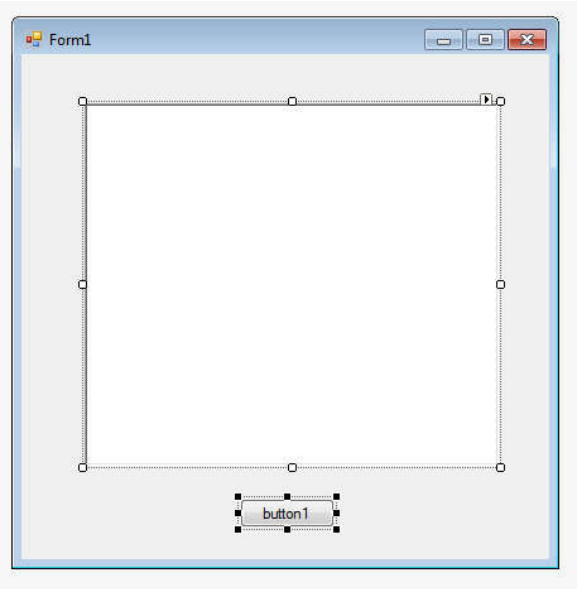

Рисунок 3.1 – Форма з елементами керування

Приклад зміни кольорів тексту, що виводиться:

```
private void button1_Click(object sender, EventArgs e) {
      for(int i = 0; i < 10; i++) {
            this.richTextBox1.SelectionStart =
               this.richTextBox1.Text.Length;
            this.richTextBox1.SelectionColor =
                  (i<3)?Color.Red:Color.Blue;
            if(i > 5) {
                  this.richTextBox1.SelectionFont =
                              new Font("Courier New", 16);
                        this.richTextBox1.SelectionColor = Color.Green;
            }
            this.richTextBox1.AppendText("Hhsdsh\n");
      }
}
```

Результат роботи програми при натисканні на кнопку показано на рис 3.2

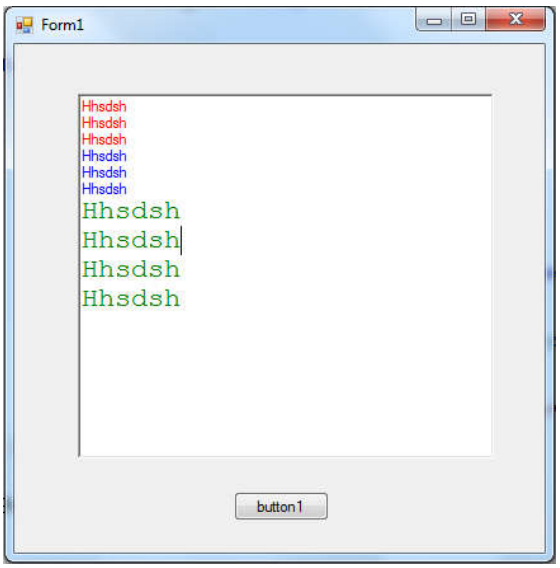

Рисунок 3.2 – Вивід в елемент керування **RichTextBox**

3.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. **Описати в протоколі** властивість **`InvokeRequired`** та метод **`Invoke.`**
3. Доповнити програму розроблену на попередній роботі виводом результатів у елементи керування форми.
4. При виводі результатів в **`RichTextBox`**, відображати інформацію від різних потоків різними кольорами.
5. Налагодити програму на комп'ютері.
6. Підготувати звіт по роботі.

3.3. Контрольні запитання

1. Що забезпечує використання методу Invoke при побудові багатопоточних програм?
2. Який порядок виконання потокобезпечного виклику елемента керування Windows Forms?
3. Для чого призначена властивість InvokeRequired елементів форм?
4. В чому полягають переваги використання RichTextBox над TextBox?

ЛАБОРАТОРНА РОБОТА №4.
ОБМІН ФАЙЛАМИ В МЕРЕЖІ ПО ПРОТОКОЛУ FTP

Мета роботи – знайомство з мережевим протоколом FTP.

4.1. Теоретичні відомості

Протокол передачі файлів FTP (англ. File Transfer Protocol,) — надає можливість абоненту обмінюватися двійковими і текстовими файлами з будь-яким комп'ютером мережі, що підтримує протокол FTP. Установивши зв'язок з віддаленим комп'ютером, користувач може скопіювати файл з віддаленого комп'ютера на свій, або скопіювати файл зі свого комп'ютера на віддалений. FTP – є протоколом гарантованої доставки, тобто протокол гарантує передачу даних (або видачу помилки) за рахунок застосування протоколу TCP.

При розгляді FTP як сервісу Інтернет мають на увазі не просто протокол, а саме сервіс — доступ до файлів, які знаходяться у файлових архівах.

FTP — стандартна програма, яка працює за протоколом TCP, яка завжди поставляється з операційною системою. Її початкове призначення — передача файлів між різними комп'ютерами, які працюють у мережах TCP/IP: на одному з комп'ютерів працює програма-сервер, на іншому — програма-клієнт, запущена користувачем, яка з'єднується з сервером і передає або отримує файли через FTP-сервіс. Все це розглядається з припущенням, що користувач зареєстрований на сервері та використовує логін та пароль на цьому комп'ютері.

В енергетиці FTP використовується, наприклад, в пристроях релейного захисту для передачі файлів осцилограм аварійних процесів в форматі Comtrade або зчитування та запису файлів конфігурації пристроїв.

Для роботи по протоколу FTP в середовищі .NET Framework в просторі імен System.Net створено класи FtpWebRequest і FtpWebResponse. Ці класи є нащадками класів WebRequest і WebResponse - відповідно.

Клас FtpWebRequest реалізує FTP-клієнт для відправки FTP-запитів до сервера.

Клас FtpWebResponse – інкапсулює відповідь FTP – сервера на запит.

Для створення запиту FtpWebRequest можна скористатися конструктором

WebRequest вказавши FTP – шлях.

Наприклад:

```
FtpWebRequest request = (FtpWebRequest)WebRequest.Create(
@"ftp://fea.kpi.ua/Книги студентов АЕ/"
);
```

Далі необхідно вказати тип запита. Наприклад для перегляду вмісту каталога:

```
request.Method = WebRequestMethods.Ftp.ListDirectoryDetails;
```

де WebRequestMethods.Ftp – це клас поля, якого визначають можливий тип запиту.

Потім при необхідності вказати логін та пароль для авторизації:

```
request.Credentials = new NetworkCredential("Логін", "Пароль");
```

Для того, щоб відправити запит та отримати відповідь використовується метод GetResponse():

```
FtpWebResponse response =
(FtpWebResponse)request.GetResponse();
```

Що б прочитати відповідь є метод GetResponseStream() класу FtpWebResponse, який повертає потік для читання, а для отримання статусу відповіді властивість StatusDescription.

Нижче наведено повний код методу, що зчитує інформацію про вміст каталогу та відображає його в контролі `richTextBox1`:

```
private bool LoadFilesList(string fullFolderName)
        {
            bool res = false;
            WriteLineToStatusLabel("Loading...");

            // Get the object used to communicate with the server.
            FtpWebRequest request = (FtpWebRequest)WebRequest.Create(fullFolderName);
            request.Method = WebRequestMethods.Ftp.ListDirectoryDetails;
            request.Credentials = new NetworkCredential(this.textBoxUser.Text,
this.textBoxPass.Text);
            request.Timeout = 5000;
            request.ReadWriteTimeout = 5000;
            request.KeepAlive = false;

            try
            {
                FtpWebResponse response = (FtpWebResponse)request.GetResponse();

                Stream responseStream = response.GetResponseStream();
                StreamReader reader = new StreamReader(responseStream);
                this.WriteLineToRichTextBox(reader.ReadToEnd());
```

```
                WriteLineToRichTextBox(String.Format("Directory List Complete, status
{0}", response.StatusDescription));

                reader.Close();
                responseStream.Close();
                response.Close();
                WriteLineToStatusLabel("OK");
                res = true;
            }
            catch (Exception ex)
            {
                WriteLineToRichTextBox("Err:" + ex.Message);
                WriteLineToStatusLabel("Error.");
            }
            return res;
        }
private void WriteLineToRichTextBox(string str)
        {
            this.richTextBox1.AppendText(str + "\n");
        }

private void WriteLineToStatusLabel(string str)
        {
            this.toolStripStatusLabel2.Text = str;
            this.Refresh();
        }
```

Для читання файлу потрібно обрати метод WebRequestMethods.Ftp.DownloadFile. А для отриманого потоку провести читання через буфер за допомогою класу BinaryReader.

Наприклад:

```
private void DownloadFtpToFile(string name)
{
    string saveFullName = @"D:\" + name;

    string downloadFileFullName = FtpPath + name;

     FtpWebRequest request = (FtpWebRequest)WebRequest.Create(downloadFileFullName);
    request.Method = WebRequestMethods.Ftp.DownloadFile;

    request.Credentials = new NetworkCredential(this.textBoxUser.Text,
this.textBoxPass.Text);

    try
    {
        using (FtpWebResponse response = (FtpWebResponse)request.GetResponse())
        {
            Stream responseStream = response.GetResponseStream();

            using (BinaryReader breader = new
BinaryReader(request.GetResponse().GetResponseStream()))
            {
                using (FileStream fileStream = new FileStream(saveFullName,
FileMode.Create))
                {
                    // читання та запис
                }
```

```
            }
        }
    }
    catch (Exception ex)
    {
        this.WriteLineToRichTextBox("Err: " + ex.Message);
        this.WriteLineToRichTextBox("File: " + downloadFileFullName);
        WriteLineToStatusLabel("Error.");
    }

}
```

На рисунку 4.1 зображено приклад програми, для перегляду каталогів та зчитування файлів з FTP – сервера.

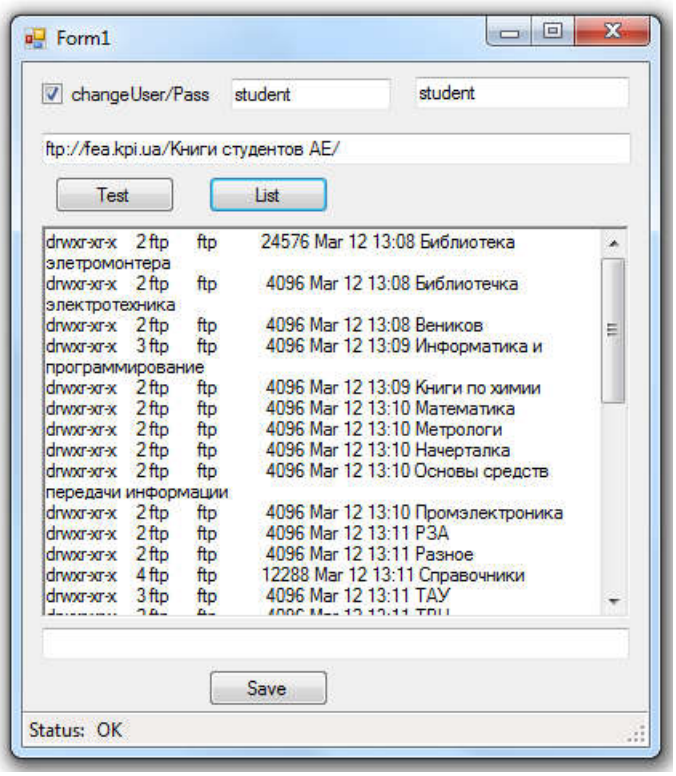

Рисунок 4.1 – Приклад інтерфейсу програми

4.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. **Ознайомитися** з основними властивостями класів FtpWebRequest, FtpWebResponse, WebRequestMethods.Ftp, **та їх описати в протоколі** .
3. Створити програму для зчитування інформації про каталоги FTP з використанням Windows Forms.
4. Доповнити програму можливістю завантаження файлу з FTP – сервера на локальний комп’ютер користувача.

5. При тестуванні програми можна скористатися сервером ftp://fea.kpi.ua/, для авторизації використовувати логін/пароль – student/student
6. Підготувати звіт по роботі.

Посилання MSDN:
https://docs.microsoft.com/ru-ru/dotnet/framework/network-programming/ftp

4.3. Контрольні запитання

1. Охарактеризуйте протокол передачі файлів FTP.
2. Які класи дозволяють працювати з FTP в .NET?
3. Як використовується протокол FTP в енергетиці?
4. Назвіть основні методи класу FtpWebRequest і FtpWebResponse.

ЛАБОРАТОРНА РОБОТА №5.
ЗНАЙОМСТВО З SYSTEM.NET.SOCKETS. ВСТАНОВЛЕННЯ СИНХРОННОГО ЗВ'ЯЗКУ ПО ПРОТОКОЛУ TCP.

Мета роботи – навчитися обмінюватись інформацією з використанням протоколу Tcp.

5.1. Теоретичні відомості

System.Net – простір імен

Простір імен System.Net містить класи, що забезпечують простий інтерфейс програмування для різних мережевих протоколів і програмний доступ і оновлення конфігурацій для просторів імен System.Net, що визначають політики кешування веб-ресурсів, а також порядок створення та відправлення повідомлень електронної пошти, що представляють заголовки MIME, забезпечують доступ до даних про трафік і мережевих адресах, а також доступ до функції однорангових мереж. Додаткові дочірні простору імен забезпечують керовану реалізацію інтерфейсу Windows Sockets (Winsock) і доступ до мережевих потоків для захисту обміну даними між вузлами.

System.Net.Sockets - простір імен

Простір імен System.Net.Sockets надає керовану реалізацію інтерфейсу Windows Sockets (Winsock) для тих розробників, яким потрібно жорсткий контроль доступу до мережі.

Класи TcpClient, TcpListener і UdpClient інкапсулюють детальні параметри створення TCP-і UDP-підключень до Інтернету.

TcpListener – клас, що прослуховує підключення від TCP-клієнтів мережі. Клас **TcpListener** надає прості методи, призначені для очікування та прийому в блокуючому синхронному режимі вхідних запитів на підключення. Для підключення до об'єкту **TcpListener,** можна використовувати об'єкт TcpClient або Socket.

Створіть об'єкт **TcpListener**, використовуючи один з конструкторів з параметром класу IPEndPoint, або вказавши локальну IP-адресу і номер локального порту, або ж тільки номер порту. Щоб не задавати адресу та порт

власноруч, можна вказати значення Any для локальної IP-адреси та значення 0 для номера локального порту, тоді ці значення будуть присвоєні автоматично основним постачальником послуг. В цьому випадку, отримані значення адреси та порту можна дізнатися за допомогою властивості LocalEndpoint.

Щоб розпочати очікування вхідних запитів на підключення, треба скористатися методом Start(). Цей метод ставитиме коректні вхідні підключення в чергу до тих пір, поки не буде завершено очікування методом метода Stop або поки не буде досягнуто максимальне значення дозволених підключень - MaxConnections.

Для того щоб прийняти підключення з вхідної черги підключень, можна скористатися методам AcceptSocket або AcceptTcpClient. Причому, ці два методи виконуватимуть блокування у випадку порожньої черги, т.т. призупинять роботу поточного потоку до появи підключення. Якщо ж необхідно уникнути блокування, скористуйтеся спочатку методом Pending для того, щоб визначити чи є в черзі доступні запити на підключення.

Метод Stop – закриває об'єкт **TcpListener**.

TcpClient – клас, що реалізує клієнтські підключення для мережевих служб протоколу TCP. Цей клас надає прості методи для підключення, а також для отримання потоків читання та запису даних в мережі в синхронному режимі з блокуванням.

Для того, щоб об'єкт TcpClient міг виконати підключення та обмін даними, відповідний об'єкт TcpListener або об'єкт Socket, що створено з використанням протоколу TPC, повинен бути в режимі очікування вхідних сигналів на підключення. Підключитися до даного «прослуховувача» можна одним з двох способів:

- створити об'єкт класу TcpClient і викликати одне з трьох перевантажень методу Connect();
- створити об'єкт класу TcpClient, з використанням імені вузла та номеру порту цього вузла. Цей конструктор автоматично розпочне намагання встановити підключення.

Якщо ж необхідно відправляти дані без встановлення підключення в синхронному режимі з блокуванням, необхідно скористатися класом UdpClient.

Для відправки та отримання даних потрібно застосувати метод GetStream(), що поверне потік класу NetworkStream. В цьому потоці викликають методи Write() та Read() для читання та відправлення даних з клієнта до сервера. По завершенні роботи потрібно скористуватися методом Close(), для того щоб звільнити всі зв'язані з об'єктом TcpClient ресурси.

Приклади:

Створення серверу, за допомогою TcpListener.

```
using System;
using System.Net.Sockets;
using System.Text;

public class TcpTimeServer {

    private const int portNum = 1300;

    public static int Main(String[] args) {
        bool done = false;
        TcpListener listener = new TcpListener(portNum);
        listener.Start();
        while (!done) {
            Console.Write("Waiting for connection...");
            TcpClient client = listener.AcceptTcpClient();

            Console.WriteLine("Connection accepted.");
            NetworkStream ns = client.GetStream();
            byte[] byteTime = Encoding.ASCII.GetBytes(DateTime.Now.ToString());
            try {
                ns.Write(byteTime, 0, byteTime.Length);
                ns.Close();
                client.Close();
            } catch (Exception e) {
                Console.WriteLine(e.ToString());
            }
        }
        listener.Stop();
        return 0;
    }

}
```

Створення клієнту, за допомогою TcpListener.

```
using System;
using System.Net.Sockets;
using System.Text;

public class TcpTimeClient {
    private const int portNum = 1300;
    private const string hostName = "127.0.0.1";
```

```
    public static int Main(String[] args) {
        try {
            TcpClient client = new TcpClient(hostName, portNum);

            NetworkStream ns = client.GetStream();

            byte[] bytes = new byte[1024];
            int bytesRead = ns.Read(bytes, 0, bytes.Length);

            Console.WriteLine(Encoding.ASCII.GetString(bytes,0,bytesRead));
            client.Close();
        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }
        return 0;
    }
}
```

5.2.Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Дослідити та описати в протоколі основні властивості та методи класів TcpClient та TcpListener.
3. Створити програму Сервер та програму Клієнт, які б дозволяли обмінюватись даними між двома комп'ютерами в консольному режимі.
4. Налагодити програму на комп'ютері.
5. Забезпечити вивід результатів роботи програми на екран комп'ютера.
6. Підготувати звіт по роботі.

5.3. Контрольні запитання

1. Яке призначення класів TcpClient, TcpListener і UdpClient?
2. Як підключитисть до об'єкту TcpListener?
3. Які методи використовуються для відправки та отримання даних TcpClient?
4. Як дізнатись локальну IP-адресу і порт користувача?

ЛАБОРАТОРНА РОБОТА №6. СТВОРЕННЯ АСИНХРОННОГО ЗВ'ЯЗКУ ЗА ДОПОМОГОЮ КЛАСІВ TCPLISTENER ТА TCPCLIENT. ЧАСТИНА 1. СТВОРЕННЯ КЛІЄНТА.

Мета роботи – навчитися передавати дані по протоколу Tcp в асинхронному режимі.

6.1. Теоретичні відомості

Загальні відомості про асинхронне програмування

Асинхронна модель, що заснована на подіях, дозволяє використовувати переваги багатопоточних програм, приховуючи багато складних проблем, що притаманні багатопоточній архітектурі. Використання класів, які підтримують цю модель, дозволяє виконати наступні дії:

- Виконувати витратні по часу задачі, такі як очікування передачі даних "в фоновому режимі", не зупиняючі роботу програми.
- Запускати одночасно декілька операцій, отримуючи повідомлення про завершення кожної з них.
- Очікувати доступності ресурсів без зупинки ("зависання") прграми.
- Зв'язуватися з асинхронними застосунками в черзі, використовуючи, відому з попередніх занять, модель "події та делегати".

Асинхронна операція, яка використовує шаблон розробки IAsyncResult, реалізується у вигляді двох методів з іменами BeginІмя_операції і EndІмя_операції, які відповідно починають і завершують асинхронну операцію Імя_операції. Наприклад, клас FileStream надає методи BeginRead і EndRead для асинхронного зчитування байт з файлу. Ці методи реалізують асинхронну версію методу Read.

Після виклику методу Beginімя_операції програма може продовжити виконання інструкцій в основному потоці, поки асинхронна операція виконується в іншому потоці. Для кожного виклику методу BeginІмя_операції в додатку також повинен бути присутнім виклик методу EndІмя_операції, щоб

отримати результати операції.

Метод NetworkStream.BeginRead

Метод BeginRead починає операцію асинхронного читання з вхідних мережевих буферів. Виклик цього методу дозволяє отримувати дані в межах окремого потоку виконання.

```
public override IAsyncResult BeginRead (byte[] buffer, int offset, int size, AsyncCallback callback, object state);
```

В одному з параметрів методу BeginRead, необхідно передати метод зворотного виклику, що реалізує делегата AsyncCallback. Через параметр state – можна передавати додаткові параметри, наприклад поточний потік для зчитування даних.

Метод зворотного виклику в своєму коді повинен викликати метод EndRead. Коли програма викличе метод BeginRead, система буде знаходитися в режимі очікування до тих пір, поки не будуть отримані дані або не виникне помилка, після чого система використає окремий потік для виконання вказаного методу зворотного виклику і заблокує метод EndRead до тих пір, поки об'єкт NetworkStream не виконає читання даних або не створить виключення.

Метод BeginRead буде виконувати читання такого об'єму даних, який вказаний в байтах в параметрі size.

NetworkStream.EndRead - метод

Метод EndRead завершує операцію асинхронного читання, активовану методом BeginRead.

Перед викликом методу BeginRead необхідно створити метод зворотного виклику, який реалізує делегат AsyncCallback (як описано вище).

Всередині методу зворотного виклику за допомогою властивості AsyncState об'єкта IAsyncResult можна отримати об'єкт переданий в параметрі state методу BeginRead.

Приклад. На базі класу `TcpClient` створимо власний клас `TcpIpClient` з методами для:

- підключення і старту читання **`Connect()`**

- відправки текстового повідомлення **Send()**
- перевірки підключення **Connected()**
- відключення **Close()**

В цьому ж класі є метод зворотного виклику **ReadCallback,** що зчитує дані та передає їх в подію **Readed** у вигляді рядка.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net.Sockets;
using System.Diagnostics;
using System.IO;

namespace TestTcpIp {
    public delegate void ReaderDelagete(Object o, string s);
    public class TcpIpClient {
        TcpClient tcpc;
        string hostIp;
        Int32 port;
        StreamWriter writer;
        byte[] readBuff = new byte[1024];

        public event ReaderDelagete Readed;
        private void ReadCallback(IAsyncResult ar) {
            try {
                if(!tcpc.Connected) return;
                int iRead = tcpc.GetStream().EndRead(ar);
                if(iRead < 1) { Close(); return; }
                string str = System.Text.Encoding.UTF8.GetString(readBuff, 0, iRead);
                if(Readed != null) Readed(this, str);
                tcpc.GetStream().BeginRead(readBuff, 0, readBuff.Length, ReadCallback,
null);
            } catch(Exception ex) {
                Trace.WriteLine(ex.Message);
                tcpc.Close();
            }
        }
        public TcpIpClient(TcpClient tcpc) {
            this.tcpc = tcpc;
            this.tcpc.SendTimeout = 1000;
        }
```

```
        public TcpIpClient(string hostIp, Int32 port) {
            this.hostIp = hostIp;
            this.port = port;
            tcpc = new TcpClient();
            tcpc.SendTimeout = 1000;
        }
        public void Connect(out string str_err) {
            str_err = String.Empty;
            try {
                if(!this.tcpc.Connected) {
                    writer = null;
                    this.tcpc.Connect(hostIp, port);
                }
                writer = new StreamWriter(this.tcpc.GetStream());
                tcpc.GetStream().BeginRead(readBuff, 0, readBuff.Length, ReadCallback,
null);
            } catch(Exception ex) {
                Trace.WriteLine(ex.Message);
                str_err = ex.Message;
                writer = null;
            }
        }

        public bool Connected() {
            return this.tcpc.Connected;
        }

        public void Close() {
            mess.Clear();
            writer = null;
            Readed = null;
            if(tcpc.Connected) tcpc.Close();
        }

        public bool Send(string str){
            if(writer == null) return false;
            try {
                writer.Write(str);
                writer.Flush();
            }catch(Exception ex) {
                Trace.WriteLine(ex.Message);
                writer = null;
```

```
                return false;
            }
            return true;
        }
    }
}
```

Створимо програму-клієнт з використанням класу **TcpIpClient**.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using TestTcpIp;
using System.Diagnostics;

namespace TcpIpClientTxt {
    public partial class FoMain : Form {

        delegate void SetTextCallback(string text);

        public FoMain() {
            InitializeComponent();
        }
        private TcpIpClient tcpIpClient;
        private void WriteLine(string str) {
            if(richTextBox1.InvokeRequired) {
                SetTextCallback d = new SetTextCallback(WriteLine);
                try {
                    this.Invoke(d, new object[] { str });
                } catch(Exception ex) {
                    MessageBox.Show(ex.Message);
                    Trace.WriteLine(ex.Message);
                }
            } else {
                richTextBox1.Text += str + Environment.NewLine;
            }
        }

        void OnRead(Object o, string s) {
            WriteLine("Read:" + s);
```

```
        }
        private void buConnect_Click(object sender, EventArgs e) {
            if(tcpIpClient != null) {
                tcpIpClient.Close();
            }
              tcpIpClient = new TcpIpClient("127.0.0.1", 2000);
            string str_err;
            tcpIpClient.Connect(out str_err);
            if(tcpIpClient.Connected()) {
                tcpIpClient.Readed += OnRead;
            } else {
                MessageBox.Show(str_err);
            }
            this.buSend.Enabled = tcpIpClient.Connected();
        }
        private void buSend_Click(object sender, EventArgs e) {
            tcpIpClient.Send(this.textBox1.Text);
            WriteLine("Sent:" + this.textBox1.Text);
        }
    }
}
```

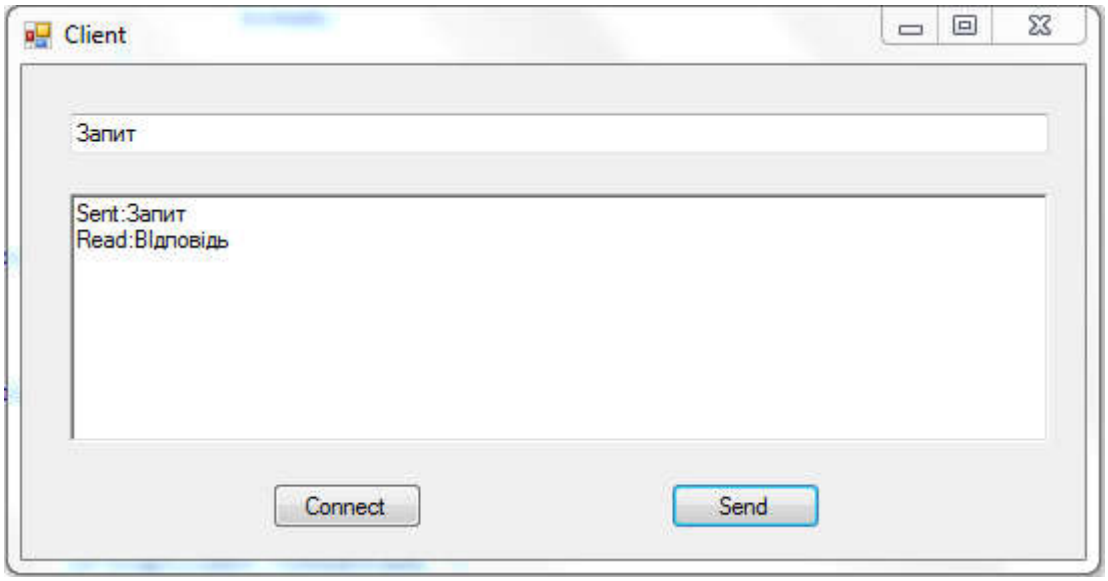

Рисунок 6.1 – зовнішній вигляд програми-клієнту

6.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Створити два проекти Windows Application.

3. За допомогою асинхронного читання реалізувати програму-клієнт для роботи із сервером.
4. Налагодити програму на комп’ютері.
5. Підготувати звіт по роботі.

6.3. Контрольні запитання

1. Які переваги та недоліки асинхронного програмування?
2. Призначення методу NetworkStream.BeginRead.
3. Призначення методу NetworkStream.EndRead.
4. Поясніть принцип роботи програми-клієнта

ЛАБОРАТОРНА РОБОТА №7. СТВОРЕННЯ АСИНХРОННОГО ЗВ'ЯЗКУ ЗА ДОПОМОГОЮ КЛАСІВ TCPLISTENER ТА TCPCLIENT. ЧАСТИНА 2. СТВОРЕННЯ СЕРВЕРА

Мета роботи – навчитися передавати дані по протоколу Tcp в асинхронному режимі.

7.1. Теоретичні відомості

Використання делегата AsyncCallback

Додатки, які можуть виконувати роботу під час очікування результатів асинхронної операції, не повинні блокуватися до завершення цієї операції. Використовуйте один з наступних варіантів, щоб продовжити виконання інструкцій в період очікування асинхронної операції.

- Використовуйте делегат AsyncCallback для обробки результатів асинхронної операції в окремому потоці.
- Щоб визначити, чи завершена операція, використовується властивість IsCompleted об'єкта IAsyncResult, що повертається методом Beginімя_операції асинхронної операції.

Замість опитування делегата про те, чи завершився асинхронно викликаний метод, було б більш ефективно змусити вторинний потік інформувати викликає потік про завершення виконання завдання. Щоб включити таку поведінку, необхідно передати екземпляр делегата System.AsyncCallback як параметр методу BeginInvoke(). Якщо передається об'єкт AsyncCallback, делегат автоматично викличе вказаний метод по завершенні асинхронного виклику.

Метод зворотного виклику буде викликаний у вторинному потоці, а не в первинному. Це має важливий наслідок для потоків з графічним інтерфейсом користувача (WPF або Windows Forms), оскільки елементи управління прив'язані до потоку, який їх створив, і можуть управлятися тільки їм. Далі, при розгляді бібліотеки TPL, будуть показані деякі приклади роботи потоків з графічного інтерфейсу. Як і будь-який делегат, AsyncCallback може викликати тільки методи, відповідні певному шаблону, який в даному випадку вимагає єдиного

параметра IAsyncResult і нічого не повертає

Приклад

На базі класу `tcpListener` створимо програму-сервер.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net;
using System.Diagnostics;
using TestTcpIp;

namespace TcpIpServerTxt {
    public partial class FoMain : Form {

        TcpIpClient tcpIpClient;

        delegate void SetTextCallback(string text);
        delegate void VoidDelegate();

        TcpListener tcpListener;
        public FoMain() {
            InitializeComponent();
        }

        private void buSend_Click(object sender, EventArgs e) {
            tcpIpClient.Send(this.textBox1.Text);
            WriteLine("Sent:" + this.textBox1.Text);
        }

        private void BuSendEnabled() {
            if(buSend.InvokeRequired) {
                this.Invoke(new VoidDelegate(BuSendEnabled));
            } else {
                buSend.Enabled = true;
                buStart.Enabled = false;
            }
        }
```

```
private void WriteLine(string str) {
    if(richTextBox1.InvokeRequired) {
        SetTextCallback d = new SetTextCallback(WriteLine);
        try {
            this.Invoke(d, new object[] { str });
        } catch(Exception ex) {
            Trace.WriteLine(ex.Message);
        }

    } else {
        richTextBox1.SelectionStart = this.richTextBox1.Text.Length;
        switch(str.Substring(0, 5)) {
            case "Sent:":
                richTextBox1.SelectionColor = Color.Red;
                break;
            case "Read:":
                richTextBox1.SelectionColor = Color.Blue;
                break;
            default:
                richTextBox1.SelectionColor = Color.Black;
                break;
        }
        richTextBox1.AppendText(str + Environment.NewLine);
        richTextBox1.ScrollToCaret();
    }
}
void OnRead(Object o, string s) {
    WriteLine("Read:" + s);
}

private void AcceptCallback(IAsyncResult ar) {
     TcpClient tcpClient = tcpListener.EndAcceptTcpClient(ar);
     WriteLine("Connected =" + tcpClient.Connected);
     if(tcpClient.Connected) {
         BuSendEnabled();
         tcpIpClient = new TcpIpClient(tcpClient);
         string str;
         tcpIpClient.Connect(out str);
         tcpIpClient.Readed += OnRead;
         WriteLine(str);
     }
}
```

```
        private void buStart_Click(object sender, EventArgs e) {
            tcpListener = new TcpListener(IPAddress.Parse("127.0.0.1"), 2000);
            tcpListener.Start();
            tcpListener.BeginAcceptTcpClient(new AsyncCallback(AcceptCallback), null);
        }
    }
}
```

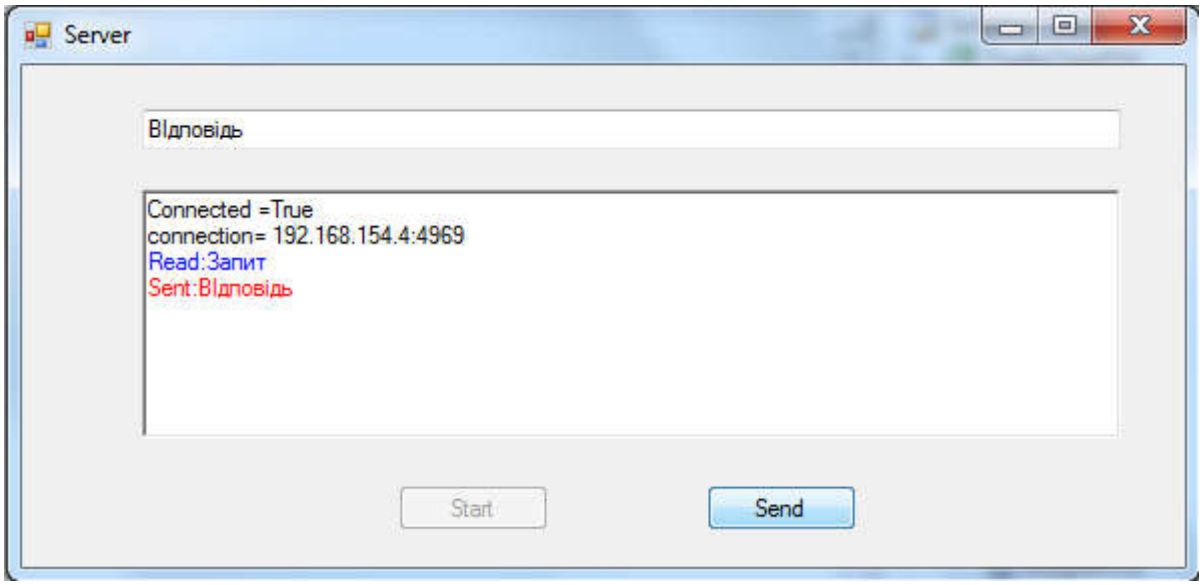

Рисунок 7.1 – зовнішній вигляд програми-серверу

7.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Доповнити проект лабораторної роботи №6 кодом серверної частини.
3. Налагодити програму на комп’ютері.
4. Підготувати звіт по роботі.

7.3. Контрольні запитання

1. Які переваги та недоліки асинхронного програмування?
2. Як використовують делегат AsyncCallback?
3. Які властивості дозволяють визначити визначити статус завершення операції?
4. Поясніть принцип роботи програми-сервера.

ЛАБОРАТОРНА РОБОТА №8.
СТВОРЕННЯ ПРОГРАМИ КЛІЄНТА MODBUS

Мета роботи – знайомство з протоколом Modbus. Створення клієнта Modbus.

8.1. Теоретичні відомості

Modbus — це протокол передачі даних, спочатку опублікований компанією Modicon (тепер Schneider Electric) у 1979 році для використання з її програмованими логічними контролерами (ПЛК). Modbus став де-факто стандартним протоколом зв'язку і тепер є загальнодоступним засобом підключення промислових електронних пристроїв [7].

Modbus заснований на технології master-slave. Широко застосовується в промисловості для організації зв'язку між електронними пристроями. Може використовувати для передачі даних через послідовні лінії зв'язку RS-485, RS-422, RS-232, а також мережі TCP/IP (Modbus TCP)[8].

Технологія master-slave – означає, що в мережі знаходиться один клієнт-хазяїн (master) і один або декілька службових серверів (slave). На електричній підстанції в якості клієнта найчастіше виступає програмне забезпечення типу SCADA, в якості серверів пристрої релейного та автоматики. Клієнт по черзі відправляє кожному серверу запити на потрібну, якщо вони підключені послідовно, та отримує від них відповіді, які розбирає, аналізує та приймає рішення про подальші дії. У випадку підключення по принципу зірка, клієнт може опитувати паралельно кілька пристроїв-серверів з різних потоків, але принцип роботи запит-відповідь зберігається.

Запити та відповіді Modbus зазвичай описуються за допомогою шістнадцяткової системи числення, де числа від 10 до 15 позначаються, відповідно, латинськими літерами від A до F (табл. 8.1).

Таблиця 8.1 – Цифри шіснадцяткової системи

0 = 0	4 = 4	8 = 8	12 = C
1 = 1	5 = 5	9 = 9	13 = D
2 = 2	6 = 6	10 = A	14 = E
3 = 3	7 = 7	11 = B	15 = F

Таким чином кожні 4-біти інформації можна позначити однією цифрою, а байт т.т. 8-біт кодується двома щіснадцятковими цифрами і може містити значення від 00 до FF (0 .. 256 в десятковій).

В залежності від середовища передачі протокол Modbus має кілька різновидів. В своїй роботі ми розглянемо протокол Modbus Tcp, як найбільш сучасний та швидкий. З назви зрозуміло, що використовують його при роботі в мережі Ethernet по протоколу передачі TCP/IP.

Для з'єднання по TCP вказують IP адресу пристрою, зазвичай адреси від 192.168.1.1 до 192.168.1.255, та TCP-порт: 502.

Запит або відповідь Modbus – це набір байтів, який ще називають пакетом. Пакет може містити до 260 байт включно і має структуру зображену на рис. 8.1.

ID транзакції	ID протоколу	Довжина пакету	Адреса пристрою	Номер команди	Дані (змінної довжини)
2 байти (слово)	2 байти (слово)	2 байти (слово)	1 байт	1 байт	N-байт

Рисунок 8.1 – Пакет Modbus

На цьому рисунку:

ID Транзакції – зазвичай нулі 00 00.

ID Протоколу - зазвичай нулі 00 00.

Довжина пакету – кількість байт пакету, що слідує за цим полем.

Адреса пристрою - ідентифікує видалений сервер, у випадку Modbus Tcp - це поле містить нулі чи одиниці, ігнорується сервером і відправляється зворотно в тому ж вигляді.

Номер команди – номери команд бувають стандартні і визначені користувачем.

Дані – набір байтів, що залежить від номеру команди, і відрізняються для запиту та відповіді.

Зауваження: в протоколі Modbus для заповнення слова (перетворення його в два байти) порядок байтів міняють: спочатку передається старший байт потім молодший байт.

Для роботи з пристроями релейного захисту стандартними є наступні номери команд:

- **1 (0x01)**- читання значень з декількох регістрів прапорів*(Read Coil Status)*
- **2 (0x02)**- читання значень з декількох дискретних входів*(Read Discrete Inputs)*
- **3 (0x03)**- читання значень з декількох регістрів зберігання*(Read Holding Registers)*
- **4 (0x04)**- читання значень з декількох регістрів вводу*(Read Input Registers)*
- **5 (0x05)**- запис значення одного прапора(Force Single Coil)
- **6 (0x06)**- запис значення в один регістр зберігання(Preset Single Register)
- **15(0x0F)-** запис значень в кілька регістрів прапорів(Force Multiple Coils)
- **16(0x10)**- запис значень в кілька регістрів зберігання(Preset Multiple Registers)

В цій роботі розглянемо лише команду номер 3 – читання значень декількох регістрів. Вона зазвичай використовується для зчитування уставок та оперативних параметрів (значень струмів, напруг, частоти, тощо) пристрою релейного захисту. Для кожного типу пристрою є своя карта пам'яті в якій вказується які параметри за якими адресами знаходяться, скільки пам'яті під них відведено, та, іноді, формули перетворень для з читаних значень, що переводять їх в реальні показники. Структура цієї команди, два поля останні поля пакету (Номер команди та Дані), зображено на рис. 8.2.

0x03	Адреса 1го слова	Кількість слів
1 байт	2 байти (слово)	2 байти (слово)

Рисунок 8.2 – Закінчення запиту на зчитування кількох слів (регістрів)

Структура полів відповіді після поля Адреса пристрою зображена на рис. 8.3. Де N – це кількість зчитаних слів, а Число зчитаних байт рівне 2*N.

0x03	Число зчитаних байт даних	1-ше зчитане слово	2-ге зчитане слово	…	N-те зчитане слово
1 байт	1 байт	2 байти (слово)	2 байти (слово)	…	2 байти (слово)

Рисунок 8.3 – Закінчення відповіді на зчитування кількох слів (регістрів)

В даній роботі в карту пам’яті представимо у вигляді таблиці 8.2

Таблиця 8.2 – Карта пам’яті оперативних параметрів пристрою РЗ

Адреса	Опис	Тип даних	Од. виміру
0x0030	Значення струму фази А	float, 4 байти	А
0x0034	Значення струму фази В	float, 4 байти	А
0x0038	Значення струму фази С	float, 4 байти	А
0x003C	Напруга фази А	float, 4 байти	В
0x0040	Напруга фази В	float, 4 байти	В
0x0044	Напруга фази С	float, 4 байти	В
0x0048	Частота	UInt16, 2 байти	1/100 Гц

8.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом.
2. Навести приклади пакетів команд для зчитування:
   А. Струмів.
   Б. Напруг.
   В. Всіх згаданих в табл. 8.2 параметрів
3. Написати програму клієнта Modbus, яка б проводила постійне опитування пристрою РЗ за вказаною картою пам’яті, та виводила б результат в поєкті Widows Form.
4. Налагодити програму на комп’ютері.
5. Підготувати звіт по роботі.

8.3. Контрольні запитання

1. Стисло опишіть протокол Modbus?
2. Що означає термін master-slave?
3. Які бувають команди Modbus?
4. За допомогою якої команди виконують зчитування кількох слів?
5. Які завдання програми клієнту?

ЛАБОРАТОРНА РОБОТА №9.
СТВОРЕННЯ ПРОГРАМИ СЕРВЕРА MODBUS

Мета роботи – знайомство з протоколом Modbus. Створення сервера Modbus.

9.1. Теоретичні відомості

Задачі програми серверу сформувати правильну відповідь на запит від клієнту, та виконати поставлені в отриманій команді задачі. Наприклад передати значення оперативних параметрів або встановити у пристрої нові значення уставок.

Для перетворення базових типів даних в масив байт, і масиву байт – в базові типи даних використовують клас BitConverter. Але слід зазначити, що для протоколу Modbus, порядок слідування байтів в масиві слід змінювати на протилежний.

9.2. Порядок виконання роботи

1. Ознайомитись із теоретичним матеріалом цієї і попередньої робіт.
2. Самостійно ознайомитися з класом BitConverter, **та описати його в протоколі.**
3. Створити програму сервера, яка, за допомогою генератора випадкових чисел, формувала б масив даних з таблиці 8.2. Та забезпечувала б передачу цих даних по протоколу Modbus Tcp клієнту у відповідь на його запит.
4. Врахувати, що значення струмів, напруг та частоти повинні змінюватися в діапазонах, що відповідають режиму нормальної роботи підстанції. Наприклад частота не повинна сильно відхилятися від значення 50,0 Гц.
5. Налагодити програму на комп’ютері.
6. Підготувати звіт по роботі.

9.3. Контрольні запитання

1. Які основні методи класу BitConverter?
2. Що означає термін master-slave?

3. Які бувають команди Modbus?
4. За допомогою якої команди виконують зчитування кількох слів?
5. Які завдання програми сервера Modbus?

ЛІТЕРАТУРА

1. Настенко, Д. В. Об’єктно-орієнтоване програмування. Частина 1. Основи об’єктно-орієнтованого програмування на мові C# [Електронний ресурс] : навчальний посібник для бакалаврів напряму підготовки 6.050701 «Електротехніка та електротехнології» програми професійного спрямування «Системи управління виробництвом та розподілом електроенергії» / Д. В. Настенко, А. Б. Нестерко ; НТУУ «КПІ». – Електронні текстові дані (1 файл: 931,2 Кбайт). – Київ : НТУУ «КПІ», 2016. – 76 с. – Назва з екрана. https://ela.kpi.ua/handle/123456789/16671

2. Обчислювальна техніка та програмування. Конспект лекцій. Частина 1 [Електронний ресурс] : навчальний посібник для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» / КПІ ім. Ігоря Сікорського ; уклад.: Г. О. Труніна, Д. В. Настенко, А. Б. Нестерко. – Електронні текстові данні (1 файл: 3,28 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 117 с. – Назва з екрана. https://ela.kpi.ua/handle/123456789/39004

3. Обчислювальна техніка та програмування. Лабораторні роботи. Частина 1 [Електронний ресурс] : навчальний посібник для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» / КПІ ім. Ігоря Сікорського; уклад.: А. Б. Нестерко, Д. В. Настенко, Г. О. Труніна. – Електронні текстові данні (1 файл: 1,99 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 83 с. – Назва з екрана. https://ela.kpi.ua/handle/123456789/39020

4. Обчислювальна техніка та програмування. Домашня контрольна робота. Частина 1 [Електронний ресурс] : навчальний посібник для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» / КПІ ім. Ігоря Сікорського ; уклад.: Д. В. Настенко, Г. О. Труніна, А. Б. Нестерко – Електронні текстові данні (1 файл: 1,31 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 17 с. – Назва з екрана. https://ela.kpi.ua/handle/123456789/39019

5. C# docs - get started, tutorials, reference. | Microsoft Docs [Електронний ресурс] URL: https://docs.microsoft.com/en-us/dotnet/csharp/

6. Simply Modbus [електронний ресурс] URL: https://www.simplymodbus.ca/

7. Drury, Bill (2009). Control Techniques Drives and Controls Handbook (PDF) (2nd ed.). Institution of Engineering and Technology. pp. 508–.

8. Modbus – Вікіпедія. [електронний ресурс] URL: https://uk.wikipedia.org/wiki/Modbus

9. Організація збору оперативної диспетчерської інформації з використанням пристроїв RTU540 на сонячній ПС 35 кВ // Д. В. Настенко, А. О. Рекс, А. М. Панченко - Міжнародний науково-технічний журнал "Сучасні проблеми електроенерготехніки та автоматики" 2019 ст. 93-96

10. Кацадзе, Т., Д. В. Настенко, Панєнко . О., і О. М. Янковська. «Дослідження режиму напруги в дальніх лініях електропередачі змінного струму». Праці Інституту електродинаміки Національної академії наук України, вип. 59, Вересень 2021, с. 043, doi:10.15407/publishing2021.59.043.