МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

***Циклова комісія програмування***

**Робочий план**

з курсу «Об’єктно-орієнтоване програмування»

Практична робота №12

Солом’яного Ярослава Сергійовича

*ПІБ студента*

студента групи **2П-18**

|  |  |  |
| --- | --- | --- |
| **Вид роботи** | **Дата** | **Оцінка** |
| *Пром. контроль* |  |  |
| *Залік* |  |  |

Викладач Марченко С. В.

Черкаси-2021

Завдання 1. (Barrier) Припустимо, що потрібно написати програмне забезпечення для приготування страви. Наприклад, у замовленні в Mcdonalds працюватиме 3 потоки: з приготування бургеру, картоплі фрі та коли. Проте замовлення буде готове до подачі лише тоді, коли будуть готові всі його компоненти. Створіть бар’єр на 3 завдання (кожне у своєму потоці). Запустіть приготування замовлення та виведіть на екран інформацію щодо порядку запуску та приготування компонентів замовлення.

Лістинг 1. Код для виконання завдання 1.

using System;

using System.Diagnostics;

using System.Globalization;

using System.Text;

using System.Threading;

namespace Ex\_1\_Practice\_12

{

class Program

{

static Stopwatch time = new Stopwatch();

static Barrier orderbarier = new Barrier(3, (barrier) => {

time.Stop();

TimeSpan ts = time.Elapsed;

Console.WriteLine("Your order is ready. Cooking time: {0:00}:{1:00}:{2:00}.{3:00}", ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);

PrintTimeStamp("Yaroslav Solomianiy, студент групи 2П-18");

});

static void Main(string[] args)

{

Stopwatch timeFries = new Stopwatch();

Stopwatch timeHamburger = new Stopwatch();

Stopwatch timeCola = new Stopwatch();

time.Start();

Console.WriteLine("We are preparing your order");

Thread drink = new Thread(() =>

{

time.Start();

Thread.Sleep(2000);

TimeSpan ts1 = time.Elapsed;

Console.WriteLine("Your Cola is ready. Cooking time:{0:00}:{1:00}:{2:00}.{3:00}", ts1.Hours, ts1.Minutes, ts1.Seconds, ts1.Milliseconds / 10);

orderbarier.SignalAndWait();

timeCola.Stop();

});

Thread fries = new Thread(() =>

{

timeFries.Start();

Thread.Sleep(13000);

TimeSpan ts2 = timeFries.Elapsed;

Console.WriteLine("Your Fries is ready. Cooking time:{0:00}:{1:00}:{2:00}.{3:00}", ts2.Hours, ts2.Minutes, ts2.Seconds, ts2.Milliseconds / 10);

orderbarier.SignalAndWait();

time.Stop();

timeFries.Stop();

});

Thread hamburger = new Thread(() => {

timeHamburger.Start();

Thread.Sleep(10000);

TimeSpan ts3 = timeHamburger.Elapsed;

Console.WriteLine("Your Burger is ready. Cooking time:{0:00}:{1:00}:{2:00}.{3:00}", ts3.Hours, ts3.Minutes, ts3.Seconds, ts3.Milliseconds / 10);

orderbarier.SignalAndWait();

time.Stop();

});

fries.Start();

hamburger.Start();

drink.Start();

}

public static void PrintTimeStamp(string name)

{

Console.OutputEncoding = Encoding.UTF8;

DateTime localDate = DateTime.Now;

var culture = new CultureInfo("ru-RU");

Console.WriteLine("Дата та час компіляції: {0}", localDate.ToString(culture));

Console.WriteLine("Автор: {0}", name);

}

}

}

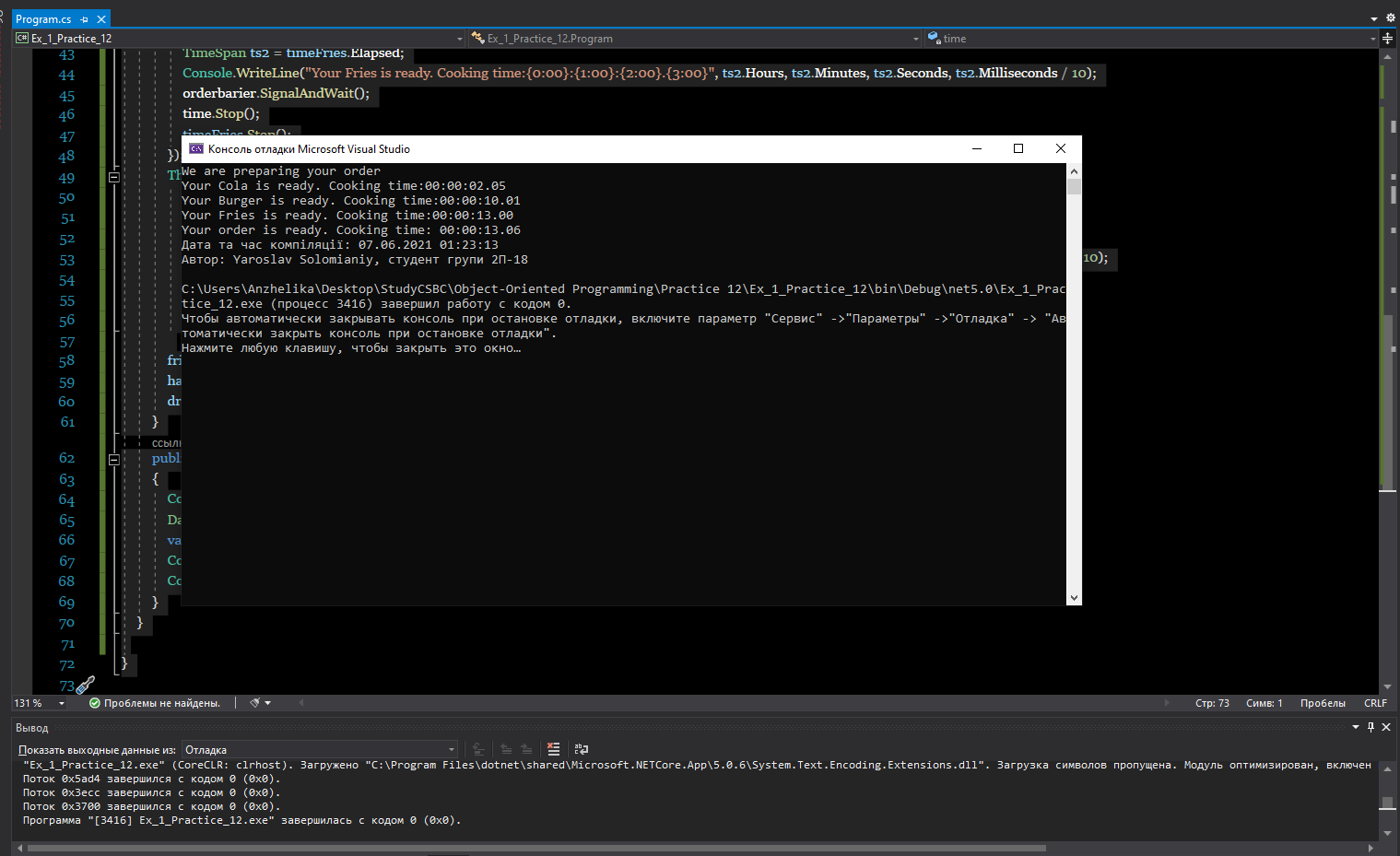


Рис 1. Результат роботи програми №1.

Завдання 2. (Semaphore) Уявіть кабіну з 4 банкоматами, за допомогою яких відбувається обслуговування населення. Це означає, що в кожний момент часу може обслуговуватись максимум 4 людини. Створіть 6 потоків (людей у спільній черзі), кожен з яких працюватиме зі спільним семафором. Потік має виконувати 4 операції, кожна з яких сповіщатиме про її виконання та спатиме 1 секунду. Також потік повинен завершувати роботу (сповіщати про звільнення блокування (lock)).

Лістинг 2. Код для виконання завдання 2.

using System;

using System.Globalization;

using System.Text;

using System.Threading;

namespace Ex\_2\_Practice\_12

{

class Program

{

private static Semaphore semaphore;

static void Main(string[] args)

{

PrintTimeStamp("Yaroslav Solomianiy, student group 2P-18");

semaphore = new Semaphore(0, 4);

Thread firstClient = new Thread(new ParameterizedThreadStart(Client));

Thread secondClient = new Thread(new ParameterizedThreadStart(Client));

Thread thridClient = new Thread(new ParameterizedThreadStart(Client));

Thread fourthClient = new Thread(new ParameterizedThreadStart(Client));

Thread fifthClient = new Thread(new ParameterizedThreadStart(Client));

Thread sixthClient = new Thread(new ParameterizedThreadStart(Client));

Thread[] arrayTread = new Thread[] { firstClient, secondClient, thridClient, fourthClient, fifthClient, sixthClient };

for (int i = 0; i < 6; i++)

{

arrayTread[i].Start();

}

Thread.Sleep(500);

semaphore.Release(4);

}

public static void Client(object num)

{

Console.WriteLine("acquiring lock...");

semaphore.WaitOne();

Console.WriteLine("got the permit!");

for (int i = 0; i < 5; i++)

{

Console.WriteLine("Is performing operation " + i);

Thread.Sleep(1000);

}

Console.WriteLine(" releasing lock...");

semaphore.Release();

}

public static void PrintTimeStamp(string name)

{

Console.OutputEncoding = Encoding.UTF8;

DateTime localDate = DateTime.Now;

var culture = new CultureInfo("ru-RU");

Console.WriteLine("Дата та час компіляції: {0}", localDate.ToString(culture));

Console.WriteLine("Автор: {0}", name);

}

}

}

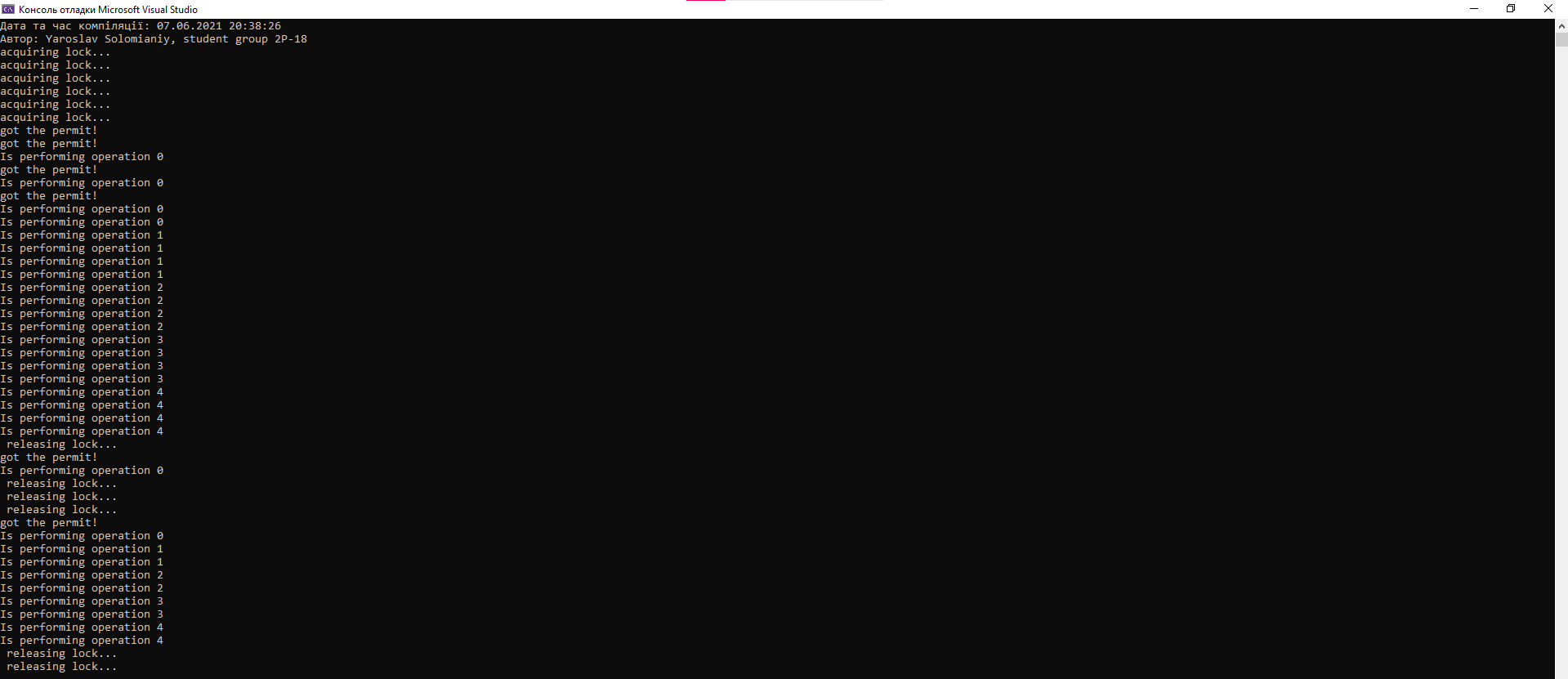


Рис 2. Результат роботи програми

Завдання 3. Condition-об’єкти) Уявіть, що пишете відеоплеєр, який відтворюватиме потокове відео. Тут буть задіяні мінімум два потоки: один буде буферизувати відео для подальшого відтворення, а інший – власне відтворювати відеоряд. Основна проблема синхронізації – плеєр не може відтворювати небуферизоване відео.

Лістинг 3.Код для виконання завдання 3

using System;

using System.Globalization;

using System.Text;

using System.Threading;

namespace Ex\_3\_Practice\_12

{

class Program

{

private static AutoResetEvent ResetEvent = new AutoResetEvent(false);

static void Main(string[] args)

{

Thread video\_bufer = new Thread(() =>

{

Console.WriteLine("Буферизування відео...");

Thread.Sleep(10000);

ResetEvent.Set();

});

Thread out\_video = new Thread(() =>

{

ResetEvent.WaitOne();

Console.WriteLine("Відтворення відео");

});

PrintTimeStamp("Solomianiy Yaroslav, Student group 2P-18");

video\_bufer.Start();

out\_video.Start();

}

public static void PrintTimeStamp(string name)

{

Console.OutputEncoding = Encoding.UTF8;

DateTime localDate = DateTime.Now;

var culture = new CultureInfo("ru-RU");

Console.WriteLine("Дата та час компіляції: {0}", localDate.ToString(culture));

Console.WriteLine("Автор: {0}", name);

}

}

}

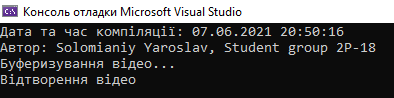


Рис 3. Результат роботи програми №3.

Завдання 4. Створіть програму з двома потоками. Один потік буде «буферизувати» відео – генеруватиме випадкове число від 1 до 5 (відсоток буферизації) та спатиме 1 секунду. Як тільки у буфері накопичилось ще 10% відео, можна розпочати «відтворення» (в іншому потоці, який спить 3 секунди). Виводьте повідомлення щодо всіх подій у програмі: новий відсоток при буферизації, спрацювання умови, запуск та завершення роботи потоків.

Лістинг 4.Код для виконання завдання 4.

using System;

using System.Globalization;

using System.Text;

using System.Threading;

namespace Ex\_4\_Practice\_12

{

class Program

{

private static AutoResetEvent autoResetEvent = new AutoResetEvent(false);

static void Main(string[] args)

{

var random\_number = new Random();

int percent = 0;

int buff = 0;

Thread buffering = new Thread(() => {

while (percent <= 100)

{

Thread.Sleep(1000);

while (buff < 10)

{

buff += random\_number.Next(1, 6);

Console.WriteLine("Buffering video : {0}%", buff);

Thread.Sleep(1000);

}

percent += buff;

buff = 0;

autoResetEvent.Set();

}

});

Thread outputing = new Thread(() => {

while (true)

{

autoResetEvent.WaitOne();

if (percent >= 100)

{

percent = 100;

Console.WriteLine("Video downloading : {0}%", percent);

Console.WriteLine("buffering is complete, the video can be viewed");

Thread.Sleep(3000);

PrintTimeStamp("Yaroslav Solomianiy, Student group 2P-18");

break;

}

else

{

Console.WriteLine("Video downloading : {0}%", percent);

Thread.Sleep(3000);

}

}

});

outputing.Start();

buffering.Start();

}

public static void PrintTimeStamp(string name)

{

Console.OutputEncoding = Encoding.UTF8;

DateTime localDate = DateTime.Now;

var culture = new CultureInfo("ru-RU");

Console.WriteLine("Дата та час компіляції: {0}", localDate.ToString(culture));

Console.WriteLine("Автор: {0}", name);

}

}

}

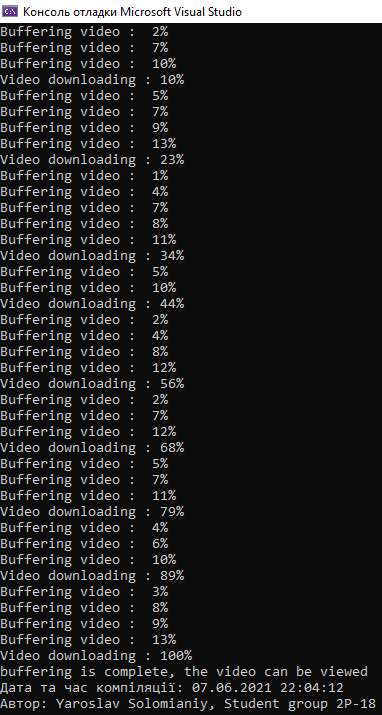


Рис 4. Результат виконання завдання 4.

Завдання №5. Розгляньте, детально опишіть та реалізуйте в звіті туторіал по роботі асинхронних потоків у мові C#.

Лістинг 5. Початковий метод main

static async Task Main(string[] args)

{

//Follow these steps to create a GitHub Access Token

// https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/#creating-a-token

//Select the following permissions for your GitHub Access Token:

// - repo:status

// - public\_repo

// Replace the 3rd parameter to the following code with your GitHub access token.

var key = GetEnvVariable("GitHubKey",

"You must store your GitHub key in the 'GitHubKey' environment variable",

"");

var client = new GitHubClient(new Octokit.ProductHeaderValue("IssueQueryDemo"))

{

Credentials = new Octokit.Credentials(key)

};

var progressReporter = new progressStatus((num) =>

{

Console.WriteLine($"Received {num} issues in total");

});

CancellationTokenSource cancellationSource = new CancellationTokenSource();

try

{

var results = await runPagedQueryAsync(client, PagedIssueQuery, "docs",

cancellationSource.Token, progressReporter);

foreach(var issue in results)

Console.WriteLine(issue);

}

catch (OperationCanceledException)

{

Console.WriteLine("Work has been cancelled");

}

}

Додаток являє собою консольний тип додатку, який використовує інтерфейс GraphQL для отримання проблем, які знаходяться в репозиторії dotnet/docs. Після створення клієнта коду GitHub в Main створено об'єкт відповіді про хід виконання та позначення маркером. Після створення цих об'єктів викликається runPagedQueryAsync, щоб отримати більше 250 нещодавно створених проблем. Результати відображаються після виконання цих завдань.

Лістинг 6. Метод runPagedQUeryAsync

private static async Task<JArray> runPagedQueryAsync(GitHubClient client, string queryText, string repoName, CancellationToken cancel, IProgress<int> progress)

{

var issueAndPRQuery = new GraphQLRequest

{

Query = queryText

};

issueAndPRQuery.Variables["repo\_name"] = repoName;

JArray finalResults = new JArray();

bool hasMorePages = true;

int pagesReturned = 0;

int issuesReturned = 0;

// Stop with 10 pages, because these are large repos:

while (hasMorePages && (pagesReturned++ < 10))

{

var postBody = issueAndPRQuery.ToJsonText();

var response = await client.Connection.Post<string>(new Uri("https://api.github.com/graphql"),

postBody, "application/json", "application/json");

JObject results = JObject.Parse(response.HttpResponse.Body.ToString());

int totalCount = (int)issues(results)["totalCount"];

hasMorePages = (bool)pageInfo(results)["hasPreviousPage"];

issueAndPRQuery.Variables["start\_cursor"] = pageInfo(results)["startCursor"].ToString();

issuesReturned += issues(results)["nodes"].Count();

finalResults.Merge(issues(results)["nodes"]);

progress?.Report(issuesReturned);

cancel.ThrowIfCancellationRequested();

}

return finalResults;

JObject issues(JObject result) => (JObject)result["data"]["repository"]["issues"];

JObject pageInfo(JObject result) => (JObject)issues(result)["pageInfo"];

}

Метод runPagedQueryAsync перераховує проблеми від найостанніших до найстаріших. Щоб продовжити з попередньої сторінки, він запитує по 25 випусків на сторінку і перевіряє структуру відповіді pageInfo. Це слід за стандартною підтримкою сторінок GraphQL для багатосторінкових відповідей. Відповідь включає в себе об'єкт pageInfo, який містить значення hasPreviousPages і startCursor, використовувані для запиту попередньої сторінки. Проблеми в масиві nodes. Метод runPagedQueryAsync додає ці вузли в масив, який містить результати з усіх сторінок.

Після отримання та відновлення сторінки результатів runPagedQueryAsync повідомляє про хід виконання і перевіряє наявність скасування. Якщо є запит на скасування, runPagedQueryAsync видає OperationCanceledException.

Лістинг 7. Перетворення в асинхронний метод

private static async IAsyncEnumerable<JToken> runPagedQueryAsync(GitHubClient client string queryText, string repoName)

Також перетворимо код в асинхроний

finalResults.Merge(issues(results)["nodes"]);

progress?.Report(issuesReturned);

cancel.ThrowIfCancellationRequested();

на

foreach (JObject issue in issues(results)["nodes"])

yield return issue;

Лістинг 8. Код після перетворення

private static async IAsyncEnumerable<JToken> runPagedQueryAsync(GitHubClient client,

string queryText, string repoName)

{

var issueAndPRQuery = new GraphQLRequest

{

Query = queryText

};

issueAndPRQuery.Variables["repo\_name"] = repoName;

bool hasMorePages = true;

int pagesReturned = 0;

int issuesReturned = 0;

// Stop with 10 pages, because these are large repos:

while (hasMorePages && (pagesReturned++ < 10))

{

var postBody = issueAndPRQuery.ToJsonText();

var response = await client.Connection.Post<string>(new Uri("https://api.github.com/graphql"),

postBody, "application/json", "application/json");

JObject results = JObject.Parse(response.HttpResponse.Body.ToString());

int totalCount = (int)issues(results)["totalCount"];

hasMorePages = (bool)pageInfo(results)["hasPreviousPage"];

issueAndPRQuery.Variables["start\_cursor"] = pageInfo(results)["startCursor"].ToString();

issuesReturned += issues(results)["nodes"].Count();

foreach (JObject issue in issues(results)["nodes"])

yield return issue;

}

JObject issues(JObject result) => (JObject)result["data"]["repository"]["issues"];

JObject pageInfo(JObject result) => (JObject)issues(result)["pageInfo"];

}

Лістинг 9. Код який оброблює колекцію

var progressReporter = new progressStatus((num) =>

{

Console.WriteLine($"Received {num} issues in total");

});

CancellationTokenSource cancellationSource = new CancellationTokenSource();

try

{

var results = await runPagedQueryAsync(client, PagedIssueQuery, "docs",

cancellationSource.Token, progressReporter);

foreach(var issue in results)

Console.WriteLine(issue);

}

catch (OperationCanceledException)

{

Console.WriteLine("Work has been cancelled");

}

Тепер замінимо його на асинхроний.

Лістинг 10. Асинхроний код який оброблює колекцію

int num = 0;

await foreach (var issue in runPagedQueryAsync(client, PagedIssueQuery, "docs"))

{

Console.WriteLine(issue);

Console.WriteLine($"Received {++num} issues in total");

}

Лістинг 11. Код до циклу

int num = 0;

var enumerator = runPagedQueryAsync(client, PagedIssueQuery, "docs").GetEnumeratorAsync();

try

{

while (await enumerator.MoveNextAsync())

{

var issue = enumerator.Current;

Console.WriteLine(issue);

Console.WriteLine($"Received {++num} issues in total");

}

} finally

{

if (enumerator != null)

await enumerator.DisposeAsync();

}

Лістинг 12. Змінимо метод

private static async IAsyncEnumerable<JToken> runPagedQueryAsync(GitHubClient client,

string queryText, string repoName, [EnumeratorCancellation] CancellationToken cancellationToken = default)

{

var issueAndPRQuery = new GraphQLRequest

{

Query = queryText

};

issueAndPRQuery.Variables["repo\_name"] = repoName;

bool hasMorePages = true;

int pagesReturned = 0;

int issuesReturned = 0;

// Stop with 10 pages, because these are large repos:

while (hasMorePages && (pagesReturned++ < 10))

{

var postBody = issueAndPRQuery.ToJsonText();

var response = await client.Connection.Post<string>(new Uri("https://api.github.com/graphql"),

postBody, "application/json", "application/json");

JObject results = JObject.Parse(response.HttpResponse.Body.ToString());

int totalCount = (int)issues(results)["totalCount"];

hasMorePages = (bool)pageInfo(results)["hasPreviousPage"];

issueAndPRQuery.Variables["start\_cursor"] = pageInfo(results)["startCursor"].ToString();

issuesReturned += issues(results)["nodes"].Count();

foreach (JObject issue in issues(results)["nodes"])

yield return issue;

}

JObject issues(JObject result) => (JObject)result["data"]["repository"]["issues"];

JObject pageInfo(JObject result) => (JObject)issues(result)["pageInfo"];

}

Атрибут EnumeratorCancellationAttribute змушує компілятор створити код для IAsyncEnumerator <T>, який робить токен, який передається GetAsyncEnumerator, видимим в тексті асинхронного ітератора у вигляді аргументу. Усередині runQueryAsync можна перевірити стан маркера і скасувати подальшу роботу при необхідності.

Використовуйте інший метод розширення, WithCancellation, щоб передати токен скасування асинхронного потоку. Змінимо цикл, перераховуючи проблеми наступним чином:

private static async Task EnumerateWithCancellation(GitHubClient client)

{

int num = 0;

var cancellation = new CancellationTokenSource();

await foreach (var issue in runPagedQueryAsync(client, PagedIssueQuery, "docs")

.WithCancellation(cancellation.Token))

{

Console.WriteLine(issue);

Console.WriteLine($"Received {++num} issues in total");

}

}