Лаба 15  
using System; // Подключаем пространство имен System для основных функций

using System.Collections.Generic; // Подключаем пространство имен Collections.Generic для работы с обобщенными коллекциями

using System.Diagnostics; // Подключаем пространство имен Diagnostics для работы с временем выполнения

using System.Linq; // Подключаем пространство имен Linq для использования LINQ-функций

using System.Threading; // Подключаем пространство имен Threading для многопоточности

using System.Threading.Tasks; // Подключаем пространство имен Tasks для асинхронного программирования

class Program // Определяем класс Program

{

static void Main(string[] args) // Главный метод программы

{

// Размер до которого будем искать простые числа

int max = 100000; // Устанавливаем максимальное число для поиска простых чисел

// Создание CancellationToken для отмены задачи

CancellationTokenSource cts = new CancellationTokenSource(); // Создаем источник токена для отмены

CancellationToken token = cts.Token; // Получаем токен отмены

// Создаем Stopwatch для измерения времени выполнения

Stopwatch stopwatch = new Stopwatch(); // Создаем новый объект Stopwatch

// Запуск параллельного поиска простых чисел с использованием TPL

stopwatch.Start(); // Запускаем измерение времени

Task findPrimesTask = Task.Run(() => FindPrimesParallel(max, token)); // Запускаем задачу для поиска простых чисел параллельно

// Задача для вычисления суммы простых чисел

Task<int> sumPrimesTask = CalculateSumOfPrimesAsync(max, token); // Запускаем асинхронную задачу для вычисления суммы простых чисел

// Задача продолжения с использованием ContinueWith

findPrimesTask.ContinueWith((prevTask) => // Устанавливаем продолжение для задачи поиска простых чисел

{

// Выводим статус завершения задачи

Console.WriteLine("Задача завершена. Результат: " + (prevTask.Status == TaskStatus.RanToCompletion ? "Успешно" : "Ошибка"));

}, TaskContinuationOptions.OnlyOnRanToCompletion); // Продолжение только в случае успешного завершения

// Завершаем измерение времени

stopwatch.Stop(); // Останавливаем измерение времени

Console.WriteLine($"Время выполнения (параллельный поиск): {stopwatch.ElapsedMilliseconds} мс"); // Выводим время выполнения

// Ожидаем завершения задачи

findPrimesTask.Wait(); // Ждем завершения задачи поиска простых чисел

sumPrimesTask.Wait(); // Ждем завершения задачи вычисления суммы

// Вывод суммы простых чисел

Console.WriteLine($"Сумма всех простых чисел до {max}: {sumPrimesTask.Result}"); // Выводим результат суммы простых чисел

// Пример асинхронной операции

PerformAsyncTask().Wait(); // Запускаем асинхронную задачу и ждем ее завершения

// Ожидаем завершения других задач

Console.WriteLine("Нажмите любую клавишу для завершения..."); // Информируем пользователя о завершении

Console.ReadKey(); // Ожидаем нажатия клавиши

}

// Алгоритм поиска простых чисел с использованием решета Эратосфена

static void FindPrimesParallel(int max, CancellationToken token) // Метод для поиска простых чисел

{

bool[] isPrime = Enumerable.Repeat(true, max + 1).ToArray(); // Создаем массив для отметки простых чисел

isPrime[0] = isPrime[1] = false; // 0 и 1 не являются простыми числами

Parallel.For(2, (int)Math.Sqrt(max) + 1, (i, state) => // Параллельно перебираем числа от 2 до корня из max

{

if (token.IsCancellationRequested) // Проверяем, была ли запрошена отмена

{

Console.WriteLine("Задача была отменена."); // Сообщаем об отмене задачи

state.Break(); // Прерываем выполнение

return; // Выходим из метода

}

if (isPrime[i]) // Если число i является простым

{

for (int j = i \* i; j <= max; j += i) // Отмечаем все кратные i как составные

{

isPrime[j] = false; // Устанавливаем значение в false

}

}

});

// Создаем список простых чисел

var primes = isPrime.Select((prime, index) => new { prime, index }) // Создаем анонимный объект с индексами

.Where(x => x.prime) // Фильтруем по простым числам

.Select(x => x.index) // Извлекаем индексы простых чисел

.ToList(); // Конвертируем в список

Console.WriteLine($"Найдено {primes.Count} простых чисел."); // Выводим количество найденных простых чисел

}

// Асинхронный метод для вычисления суммы простых чисел

static async Task<int> CalculateSumOfPrimesAsync(int max, CancellationToken token) // Метод для вычисления суммы простых чисел

{

return await Task.Run(() => // Запускаем задачу в отдельном потоке

{

bool[] isPrime = Enumerable.Repeat(true, max + 1).ToArray(); // Создаем массив для отметки простых чисел

isPrime[0] = isPrime[1] = false; // 0 и 1 не являются простыми числами

Parallel.For(2, (int)Math.Sqrt(max) + 1, (i, state) => // Параллельно перебираем числа

{

if (token.IsCancellationRequested) // Проверяем, была ли запрошена отмена

{

state.Break(); // Прерываем выполнение

return; // Выходим из метода

}

if (isPrime[i]) // Если число i является простым

{

for (int j = i \* i; j <= max; j += i) // Отмечаем все кратные i как составные

{

isPrime[j] = false; // Устанавливаем значение в false

}

}

});

// Создаем список простых чисел

var primes = isPrime.Select((prime, index) => new { prime, index }) // Создаем анонимный объект с индексами

.Where(x => x.prime) // Фильтруем по простым числам

.Select(x => x.index) // Извлекаем индексы простых чисел

.ToList(); // Конвертируем в список

return primes.Sum(); // Возвращаем сумму простых чисел

});

}

// Пример асинхронного метода с использованием async/await

static async Task PerformAsyncTask() // Асинхронный метод

{

Console.WriteLine("Запуск асинхронной операции..."); // Информируем о запуске

await Task.Delay(2000); // Эмулируем долгую операцию с задержкой 2 секунды

Console.WriteLine("Операция завершена."); // Сообщаем о завершении операции

}

}

Лаба 14

using System; // Подключаем пространство имен System для основных функций

using System.Collections.Generic; // Подключаем пространство имен Collections.Generic для использования обобщенных коллекций

using System.Diagnostics; // Подключаем пространство имен Diagnostics для работы с процессами

using System.Diagnostics.Metrics; // Подключаем пространство имен Metrics для производительности (не используется в коде)

using System.Linq; // Подключаем пространство имен Linq для LINQ-функциональностей (не используется в коде)

using System.Reflection; // Подключаем пространство имен Reflection для доступа к метаданным о сборках

using System.Text; // Подключаем пространство имен Text для работы с текстом (не используется в коде)

using System.Threading; // Подключаем пространство имен Threading для многопоточности

using System.Threading.Tasks; // Подключаем пространство имен Tasks для асинхронного программирования (не используется в коде)

static class Info // Определяем статический класс Info

{

private static string name\_file = "text.txt"; // Определяем статическую строку для имени файла

private static Mutex mutex = new(); // Создаем новый Mutex для синхронизации потоков

// Метод для отображения информации о всех процессах

public static void ShowProcesses()

{

var allProcesses = Process.GetProcesses(); // Получаем массив всех запущенных процессов

foreach (var process in allProcesses) // Перебираем каждый процесс

{

try

{

// Отображаем информацию о процессе

Console.WriteLine($"Id процесса - {process.Id}\n" +

$"Имя процесса - {process.ProcessName}\n" +

$"Приоритет процесса - {process.BasePriority}\n" +

$"Время запуска процесса - {process.StartTime}\n" +

$"Текущее состояние процесса - {process.Responding}\n" +

$"Время работы процесса - {process.TotalProcessorTime}\n" +

$"Время загрузки процесса - {process.UserProcessorTime}\n");

}

catch (Exception e) // Обрабатываем исключения

{

// Обработка исключений (пусто)

}

}

Console.WriteLine($"Количество процессов - {allProcesses.Length}"); // Выводим количество процессов

}

// Метод для отображения информации о доменах приложения

public static void ShowDomains()

{

AppDomain domain = AppDomain.CurrentDomain; // Получаем текущий домен приложения

Console.WriteLine($"Имя домена - {domain.FriendlyName}"); // Выводим имя домена

Console.WriteLine("Базовый каталог:\n" + domain.BaseDirectory); // Выводим базовый каталог

Console.WriteLine($"Детали конфигурации - {domain.SetupInformation}"); // Выводим детали конфигурации

Console.WriteLine("Сборки, загруженные в домен:"); // Информируем о загруженных сборках

Assembly[] assemblies = domain.GetAssemblies(); // Получаем массив сборок, загруженных в домен

foreach (Assembly asm in assemblies) // Перебираем каждую сборку

{

Console.WriteLine(asm.GetName().Name); // Выводим имя сборки

}

/\*AppDomain newDomain = AppDomain.CreateDomain("New Domain"); // создание нового домена

newDomain.Load(Assembly.GetExecutingAssembly().FullName); // загрузка сборки

AppDomain.Unload(newDomain); // выгрузка домена + всех содержащихся в нём сборок\*/

}

// Метод для отображения простых чисел

public static void ShowSimpleNumbers()

{

var first = new Thread(PrintSimpleNumbers); // Создаем новый поток для печати простых чисел

first.Start(); // Запускаем поток

first.Name = "Simple\_numbers"; // Устанавливаем имя потока

first.Join(); // Ожидаем завершения потока

Console.WriteLine(); // Печатаем пустую строку

}

// Метод для отображения информации о потоке

public static void ShowInfo(object thread)

{

var z = thread as Thread; // Приведение объекта к типу Thread

Console.WriteLine($"Имя потока - {z.Name}\n" +

$"Приоритет потока - {z.Priority}\n" +

$"Статус потока - {z.ThreadState}\n" +

$"Поток фоновый - {z.IsBackground}\n" +

$"Поток запущен - {z.IsAlive}\n" +

$"Поток приостановлен - {z.IsThreadPoolThread}\n");

}

// Метод для печати простых чисел

public static void PrintSimpleNumbers()

{

var first = new Thread(ShowInfo); // Создаем новый поток для отображения информации о текущем потоке

first.Start(Thread.CurrentThread); // Запускаем поток с передачей текущего потока

first.Join(); // Ожидаем завершения потока

Console.WriteLine("Введите n:"); // Запрашиваем у пользователя ввод числа

int n = int.Parse(Console.ReadLine()); // Считываем и преобразуем введенное значение в целое число

for (int i = 2; i < n; i++) // Перебираем числа от 2 до n

{

bool isPrime = true; // Предполагаем, что число простое

for (int j = 2; j < i; j++) // Проверяем делимость числа i на все числа от 2 до i

{

if (i % j == 0) // Если i делится на j

{

isPrime = false; // Число не простое

break; // Выходим из цикла

}

}

if (isPrime) // Если число простое

{

Console.WriteLine(i); // Выводим его

Thread.Sleep(100); // Задержка на 100 мс

}

}

}

// Метод для запуска потоков, показывающих четные и нечетные числа

public static void ThoThreads()

{

var first = new Thread(OddNumbers) { Priority = ThreadPriority.Highest }; // Создаем поток для нечетных чисел с высоким приоритетом

var two = new Thread(EvenNumbers); // Создаем поток для четных чисел

first.Start(); // Запускаем поток нечетных чисел

two.Start(); // Запускаем поток четных чисел

FirstlyEvenSecondlyOdd(); // Вызываем метод для последовательного показа четных и нечетных чисел

ShowOneByOne(); // Вызываем метод для показа чисел по одному

}

// Метод для показа четных и нечетных чисел по одному

private static void ShowOneByOne()

{

var mutex = new Mutex(); // Создаем новый Mutex для синхронизации

var even = new Thread(ShowEvenNumbers); // Создаем поток для показа четных чисел

var odd = new Thread(ShowOddNumbers); // Создаем поток для показа нечетных чисел

odd.Start(); // Запускаем поток нечетных чисел

even.Start(); // Запускаем поток четных чисел

even.Join(); // Ожидаем завершения потока четных чисел

odd.Join(); // Ожидаем завершения потока нечетных чисел

// Метод для показа четных чисел

void ShowEvenNumbers()

{

for (var i = 0; i < 15; i++) // Перебираем числа от 0 до 14

{

mutex.WaitOne(); // Ожидаем получения Mutex

if (i % 2 == 0) // Если число четное

Console.Write(i + " "); // Выводим его

mutex.ReleaseMutex(); // Освобождаем Mutex

}

}

// Метод для показа нечетных чисел

void ShowOddNumbers()

{

for (var i = 0; i < 10; i++) // Перебираем числа от 0 до 9

{

mutex.WaitOne(); // Ожидаем получения Mutex

Thread.Sleep(200); // Задержка на 200 мс

if (i % 2 != 0) // Если число нечетное

Console.Write(i + " "); // Выводим его

mutex.ReleaseMutex(); // Освобождаем Mutex

}

}

}

// Метод для показа четных и нечетных чисел по очереди

private static void FirstlyEvenSecondlyOdd()

{

var objectToLock = new object(); // Создаем объект для блокировки

var even = new Thread(ShowEvenNumbers); // Создаем поток для показа четных чисел

var odd = new Thread(ShowOddNumbers); // Создаем поток для показа нечетных чисел

even.Start(); // Запускаем поток четных чисел

odd.Start(); // Запускаем поток нечетных чисел

even.Join(); // Ожидаем завершения потока четных чисел

odd.Join(); // Ожидаем завершения потока нечетных чисел

// Метод для показа четных чисел

void ShowEvenNumbers()

{

lock (objectToLock) // Блокируем объект для синхронизации

{

for (var i = 0; i < 15; i++) // Перебираем числа от 0 до 14

{

if (i % 2 == 0) // Если число четное

Console.Write(i + " "); // Выводим его

}

}

}

// Метод для показа нечетных чисел

void ShowOddNumbers()

{

for (var i = 0; i < 10; i++) // Перебираем числа от 0 до 9

{

Thread.Sleep(200); // Задержка на 200 мс

if (i % 2 != 0) // Если число нечетное

Console.Write(i + " "); // Выводим его

}

}

}

// Метод для показа нечетных чисел

public static void OddNumbers()

{

string txt = string.Empty; // Создаем пустую строку для записи нечетных чисел

for (int i = 0; i < 10; i++) // Перебираем числа от 0 до 9

{

Thread.Sleep(200); // Задержка на 200 мс

if (i % 2 != 0) // Если число нечетное

{

txt += i + " "; // Добавляем его в строку

Console.WriteLine(i + " "); // Выводим его

}

}

txt += "\n"; // Добавляем перевод строки

File.AppendAllText(name\_file, txt); // Записываем строку в файл

}

// Метод для показа четных чисел

public static void EvenNumbers()

{

string txt = string.Empty; // Создаем пустую строку для записи четных чисел

for (int i = 0; i < 15; i++) // Перебираем числа от 0 до 14

{

Thread.Sleep(300); // Задержка на 300 мс

if (i % 2 == 0) // Если число четное

{

txt += i + " "; // Добавляем его в строку

Console.WriteLine(i + " "); // Выводим его

}

}

txt += "\n"; // Добавляем перевод строки

File.AppendAllText(name\_file, txt); // Записываем строку в файл

}

// Метод для чтения содержимого файла

public static void ReadFile()

{

var file = File.ReadAllText(name\_file); // Читаем содержимое файла

Console.WriteLine(file); // Выводим содержимое файла

}

}