**Министерство науки и высшего образования**

**Федеральное государственное бюджетное образовательное учреждение высшего образования “Северо-Осетинский государственный университет им.К.Л.Хетагурова”**

**Факультет математики и компьютерных наук  
  
  
  
  
  
Курсовая работа  
Многопоточность на корабликах**

Студентки 1 курса

Cпециальность - 01.03.02 Прикладная математика и информатика

Квалификация - бакалавр

Форма обучения - очно

А. А. Мамсурова

Научный руководитель:

доцент кафедры прикладной математики и информатики

Гутнова А. К.

**Владикавказ, 2022**

# Оглавление

[**Оглавление**](#_o9y1dzwnkaag) **2**

[**Введение**](#_isu0sp2dosvt) **3**

[Постановка задачи](#_9xlweb5mb49z) 3

[Глава 1. Теоретическая часть](#_4r21dar71lh1) 4

[Основные понятия многопоточности](#_9glic1ntfbqa) 4

[Словесное описание](#_2et92p0) 5

[Глава 2. Практическая часть](#_95s0sotz5wi6) 7

[Способ реализации](#_278q91yqnwd8) 7

[Структура консольного приложения](#_dqzs9fpy9m5n) 7

[Глава 3. Документация](#_8bk5lti8ceaj) 8

[Руководство системного администратора](#_d7fhn3o3ps1k) 8

[Руководство пользователя](#_pbhls8427mak) 8

[Руководство программиста](#_dt6nw9lax6x9) 10

[Заключение](#_ag2qphnfcl07) 15

[Список литературы](#_xan15gs2aory) 16

## 

## 

## **Введение**

**Многопоточность -** свойство платформы(например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков(поток представляет некоторую часть кода программы), выполняющихся «параллельно», то есть без предписанного порядка во времени.

При выполнении программы каждому потоку выделяется определенный квант времени. И при помощи многопоточности мы можем выделить в приложении несколько потоков, которые будут выполнять различные задачи одновременно. Представим что у нас есть приложение которое отправляет фотографии кому-либо. Отправка фотографий будет занимать какое то количество времени. С помощью многопоточности мы сможем дать доступ к UI пользователю во время отправки фотографий.

### Постановка задачи

Передо мной была поставлена следующая задача:

* ознакомиться с таким понятием как многопоточность;
* рассказать и продемонстрировать как плюсы, так и минусы многопоточного программирования;
* разработать консольное приложение для демонстрации работы многопоточности;
* правильно разбить задачу на параллельность;
* синхронизировать потоки.

## **Глава 1. Теоретическая часть**

### **Основные понятия многопоточности**

**Многозадачность (multitasking)** — свойство операционной системы или среды выполнения обеспечивать возможность параллельной (или псевдо параллельной) обработки нескольких задач.[3]

**С точки зрения пользователя:**

**Процесс** — экземпляр программы во время выполнения.

**Потоки** — ветви кода, выполняющиеся «параллельно», то есть без предписанного порядка во времени.

**С точки зрения операционной системы:**

**Процесс** — это просто контейнер, в котором находятся ресурсы программы:

* адресное пространство
* потоки
* открытые файлы
* дочерние процессы
* и т.д.

**Поток** — это просто контейнер, в котором находятся:

* Счётчик команд
* Регистры
* Стек

**Отличие процесса от потока -** процесс рассматривается ОС, как заявка на все виды ресурсов (память, файлы и пр.), кроме одного — процессорного времени. Поток — это заявка на процессорное время. Процесс — это всего лишь способ группировать взаимосвязанные данные и ресурсы, а потоки — это единицы выполнения (unit of execution), которые выполняются на процессоре.

### Словесное описание

Есть транспортные корабли, которые подплывают к туннели и далее плывут к причалам для погрузки разного рода товара. Они проходят через узкий туннель, где одновременно могут находиться только 5 кораблей. Необходимо правильным образом организовать работу кораблей, причалов, тоннеля (рис.1.).

Изначально свою работу начинают “Генератор кораблей” и “Причалы”. Они параллельно начинают свою работу.

Далее идет процесс отправки корабля в тоннель и далее к причалу

* Сгенерированный корабль отправляется в тоннель(если тоннель не заполнен).
* Каждый причал обращается к тоннелю и ищет корабль нужного типа(если причал не занят и корабль нужного типа имеется в тоннель корабль отправляется на погрузку).
* После завершения погрузки корабль отправляется в свободное плавание.

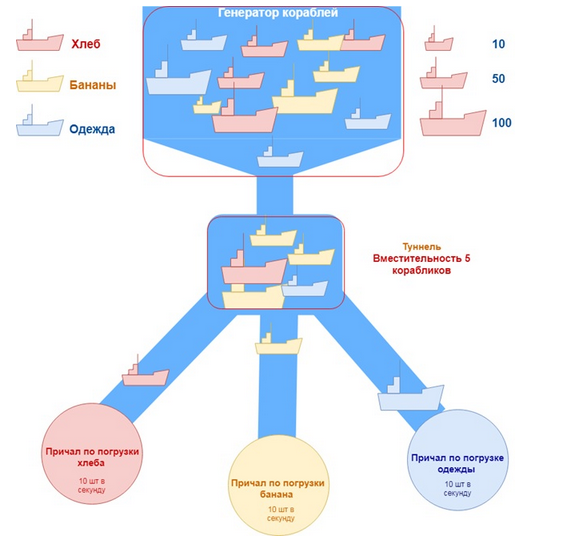


рис.1. Принцип работы приложения

## **Глава 2. Практическая часть**

### **Способ реализации**

Изначально пользователь выбирает число кораблей, которое необходимо заполнить грузом. Далее создаются потоки для генерирования и отправки кораблей к причалам. Генерация и отправка происходит одновременно. При создании корабля поток генерации пытается добавить его в тоннель, если там есть свободное место, и параллельно с этим причалы ищут в тоннеле корабли, с определенным типом продовольствия, чтобы сообщить кораблю, что причал свободен и можно начинать погрузку.

После окончания работы генератора, по истечению 10 секунд, причалы перестают функционировать.

### **Структура консольного приложения**

Для решения поставленной задачи были созданы такие классы как: “Enums”, “Ship”, ”Tunnel”, “ShipGenerator”, “PierLoader”.

* “Enums”-содержит в себе все необходимые перечисления.
* “Ship”-описывает корабль.
* “Tunnel”-регулирует работу тоннеля.
* “Ship generator”- создает определенное количество кораблей и помещает их в тоннель.
* “PierLoader”-бесконечно пытается забрать корабли из тоннеля.

## 

## **Глава 3. Документация**

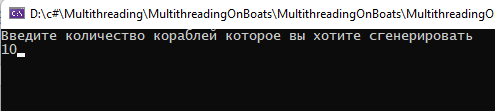
### Руководство системного администратора

Приложение реализовано в среде Visual Studio 2022 на языке C#(.NET Framework 4.6.1). Для запуска приложения требуется Windows 7-10. Особых требований к объему памяти и производительности нет. Для работы приложения надо запустить файл MultithreadingOnShips.exe

### Руководство пользователя

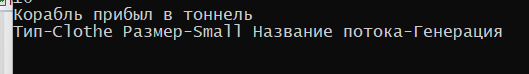
**Консольное приложение**

Для запуска консольного приложения необходимо запустить MultithreadingOnShips.exe. Далее в консоле вам необходимо ввести количество кораблей, которое вам необходимо заполнить товаром.

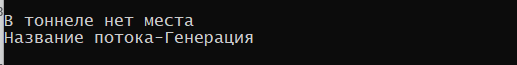


Далее программа начнет свое выполнение и в окне вывода вы сможете увидеть процесс ее работы.

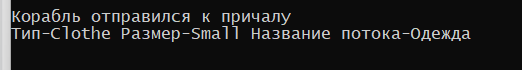
“Корабль прибыл в тоннель” - сгенерированный корабль смог попасть в тоннель.



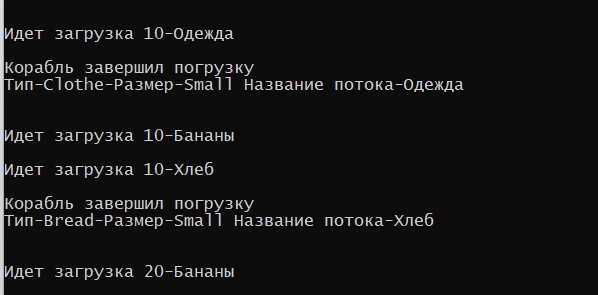
“В тоннеле нет места” - сгенерированный корабль не смог попасть в тоннель, так как он был заполнен.



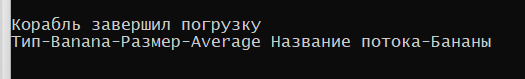
“Корабль отправился к причалу” - корабль отправился к нужному причалу для погрузки .



“Идет загрузка” - корабль приплывший к причалу начал пополняться припасами.



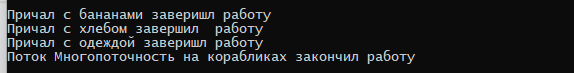
“Корабль завершил погрузку” - корабль полностью заполнен припасами и готов к отправке.



“Генератор закончил работу” - необходимое количество кораблей было сгенерировано и генератор завершает свою работу.



Все корабли были заполнены и причалы завершают свою работу.



### 

### Руководство программиста

*Класс Enums.cs*

public enum Types

{

Banana,

Cloth,

Bread

}

public enum Sizes

{

Small = 10,

Average = 50,

Huge = 100

}

*Содержит в себе два перечисления(Enum).*

Types (три возможных типа корабля) - с бананами, хлебом и одеждой

Sizes (три возможных размера корабля) -10кг, 50кг, 100кг (вместимость)

*Класс Ship.cs*

Свойства:

Count - сколько кг продовольствия в данный момент на корабле.

Type - тип корабля.

Size - размер корабля.

Методы:

Add - добавляет count кг товара в свойство Count.

CheckCount - проверка на завершение загрузки.

*Класс Tunnel.cs*

Свойства:

block - объект заглушка для синхронизации потоков.

tunnel - лист кораблей имитирующий туннель.

MinShipsInTunnel - минимально возможное количество кораблей в тоннеле.

MaxShipsInTunnel - максимально возможное количество кораблей в тоннеле.

shipCounter - показывает сколько кораблей на данный момент находится в тоннеле.

Методы:

AddShip - функция позволяющая добавить корабль в тоннель.

public bool AddShip(Ship ship)

{

try

{

lock (block) // весь код блокируется и становится недоступным для других потоков[4]

{

var th=Thread.CurrentThread;

if (shipCounter < MaxShipsInTunnel)

{

Console.WriteLine($"Корабль прибыл в тоннель\nТип-{ship.Type} Размер-{ship.Size} Название потока-{th.Name}\n\n");

Monitor.PulseAll(block);//уведомляет все потоки из очереди ожидания, что текущий поток освободил объект[4]

tunnel.Add(ship);

shipCounter += 1;

}

else

{

Console.WriteLine($"В тоннеле нет места\nНазвание потока-{th.Name}\n\n");

Monitor.Wait(block);//освобождает блокировку объекта и переводит поток в очередь ожидания[4]

return false;

}

}

}

catch

{

}

return true;

}

GetShip - функция, позволяющая получить корабль определенного типа(параметр type) из тоннеля.

*Класс ShipGenerator.cs*

Свойства:

block - объект заглушка.

Tunnel - тоннель, в который созданные корабли должны приплывать.

ShipCount - количество кораблей, которые нужно сгенерировать.

ShipsWasCreated - счетчик, отслеживающий сколько кораблей было создано.

Методы:

Start - функция запускается в отдельном потоке. Генерирует корабли и пытается отправить их в тоннель.

static public Sizes CreateSize()

{

var sizes = Enum.GetValues(typeof(Sizes));

return (Sizes)sizes.GetValue(new Random().Next(sizes.Length));

}

static public Types CreateType()

{

var types = Enum.GetValues(typeof(Types));

return (Types)types.GetValue(new Random().Next(types.Length));

}

CreateSize - функция возвращает случайное значение из Enum-”Sizes”.

CreateType - функция возвращает случайное значение из Enum-”Types”.

Рандомное создание корабля.

*Класс PierLoader.cs*

Свойства:

Tunnel - туннель, из которого PierLoader пытается достать корабль.

ShipType - поле, которое указывает на тип корабля, который может принимать у себя причал для погрузки товара.

IsAlive - булевая переменная отвечающая за работу причала.

Методы:

public void Start()

{

while (IsAlive)

{

try

{

Thread.Sleep(1000);

Ship ship = Tunnel.GetShip(ShipType);

var th = Thread.CurrentThread;

if (ship != null)

{

while (ship.CheckCount())

{

Thread.Sleep(100);

ship.Add(10);

Console.WriteLine($"Идет загрузка {ship.Count}-{th.Name}\n");

}

Console.WriteLine($"Корабль завершил погрузку\nТип-{ship.Type}-Размер-{ship.Size} Название потока-{th.Name}\n\n");

}

}

catch

{

}

}

}

Start - функция запускается в отдельном потоке. Причал пытается найти корабль(типа ShipType) и начать его погрузку.

Break - функция останавливающая работу причала.

*Класс Program.cs*

Методы:

Main - точка входа в программу. Создается отдельный поток для работы кораблей, причалов и генератора посредством вызова функции Start.

Start - функция создает все необходимые потоки (каждый поток имеют свое название) и запускает их. По мере завершения работы потоков на консоль выводятся соответствующие текстовые сообщения.

StopPierLoaders - функция останавливает всю работу потоков. Вызывается по истечению 10 секунд после завершения работы Генератора.

Библиотеки, используемые в проекте

* using System.Collections.Generic
* using System.Threading
* using System

## 

## Заключение

В ходе выполнения работы было написано консольное приложение, которое позволило на практике использовать все прелести многопоточного программирования. Были разобраны такие понятия как многозадачность, многопоточность, процессы и потоки. Безусловно остались некоторые пробелы в знаниях (синхронизация потоков), поскольку знаний и навыков студента первого курса пока что не хватает для полноценного осознания данной тематики. Надеюсь, что в будущем эти пробелы будут восполнены. В дальнейшем хотелось бы разработать WinForm приложение в виде игры, которое бы могло помочь начинающим программистам разобраться с такой непростой темой, как многопоточность.

## 

## Список литературы

1. Многопоточность на корабликах. URL: <https://habr.com/ru/post/352374/> (дата обращения: 23.05.2022)
2. Документация Microsoft C#:

<https://docs.microsoft.com/ru-ru/dotnet/api/system.threading?view=net-6.0> (дата обращения: 23.05.2022)

1. Основная теоретическая часть:

<https://habr.com/ru/company/otus/blog/549814/>

(дата обращения: 23.05.2022)

1. Примеры использования System.Threading:

<https://metanit.com/sharp/tutorial/11.1.php> (дата обращения:23.05.2022)

1. Видео материл про многопоточность (75-80) <https://www.youtube.com/playlist?list=PLH3y3SWteZd2kB92hHtqMnLNZrKQVo9Up>

(дата обращения: 23.05.2022)