Факултет по Математика и Информатика
СУ “Св. Климент Охридски”

Проект за избираема дисциплина:
"Семинар по Системно Програмиране”

Тема 2:

# Система за таймери

Автор: Румен Георгиев Азманов
Специалност: Компютърни науки, Курс: 3, Група: 3
Факултетен номер: 82176

гр. София
07.2023г.

# Увод

## Описание и цели

Проектът трябва да реализира система за организация на таймери. Всеки таймер притежава функция, която се изпълнява след точно зададено време или когато е възможно за системата да го изпълни. Таймерите могат да бъдат с единично изпълнение, с многократно повтаряне и с точен брой повторения. Системата трябва да поддържа възможност да се изтриват (отменят) таймери и то за многонишкови програми.

# Описание на изискванията

## Функционални изисквания

- Създаване на системата
  - Създава се с параметър брой използвани нишки.
- Създаване на единичен таймер
  - Създава се чрез параметри период до изпълнение и функция, която да се изпълни. Връща инстанция на манипулатор, който се използва за изтриване (отменяне) на таймер.
- Създаване на повтарящ се таймер
  - Създава се чрез параметри интервал за изпълнение и функция, която да се изпълни. Връща инстанция на манипулатор, който се използва за изтриване (отменяне) на таймер.
- Създаване на повтарящ се таймер с краен брой повторения
  - Създава се чрез параметри интервал за изпълнение, брой повторения и функция, която да се изпълни. Връща инстанция на манипулатор, който се използва за изтриване (отменяне) на таймер.
- Отменяне на таймер
  - Използва манипулатор на таймер за изтриването. Ако е възможно, то се връща истина, в противен случай лъжа.
- Изтриване на системата
  - Деактивират се всички бъдещи таймери и се освобождават заделените ресурси.

## Нефункционални и технически изисквания

- Език на реализация - C++
- Възможност за използване в многонишков режим
  - Системата трябва да може да се ползва от повече от една нишки без това да предизвиква проблем
- Зададен интерфейс за употреба
- Конфигурационни параметри
  - Точен брой нишки за работа

- Прилагат се основни ООП принципи
- Прилагат се научени от курса по ССП методи и знания
- Оптимално използвани ресурси

# Архитектура

Основните елементи на проекта са таймер, контейнер за таймери и организатор на таймерите. Таймерите са от клас *Timer* и се създават при извикване на съответния метод от класа за организиране - *TimerScheduler* подобен на *thread pool.* По време на работа таймерите се съхраняват в *TimersList* (разширена приоритетна опашка), където се пренареждат по бъдещия си ред на изпълнение. В *TimerScheduler* постоянно проверява дали следващият таймер трябва да се изпълни. Съответно в този момент предава на нишка изпълнението на таймера и неговата функция. Всяка нишка се грижи да вземе съответния таймер и да го отбележи като изпълняващ се. Съответно след приключване на работа да го върне или изтрие.

# Проектиране

## Клас *Timer*

Таймерът е основна структурна единица в проекта. Всеки таймер има по условие времето си на изчакване, функция и параметри за възможност да бъде повторен. За целите на използването са добавени времето за изпълнение и указател към следващ таймер. Методите са публични, но се приема, че се ползват само от класовете, организиращи таймера. Освен методите аксесори на членовете се използват и следните:

- *void setExecuteTime()* – Сложност: O(C)
    - За опресняване на времето на таймера. Използва се текущото време и се добавя времето на изчакване (или интервал ако е повтарящ се).
- *void noRepeat()* – Сложност: O(C)
    - Таймерът се девалидира и се премахват свойствата му да бъде изпълнен повторно.
- *void reduceRepeatCount()* – Сложност: O(C)
    - Намалява се броят изпълнения на таймера, тоест ако са били краен брой се е изпълнил веднъж.

## Клас *TimersList*

Този клас се използва главно за контейнер на таймерите под форма на свързан списък, където самите таймери притежават указател към следващ. Те не се създават вътре в класа, а се подават по указател. Има стандартен интерфейс и сложност на едносвързан списък с указатели към първи и последен елемент (без допълнителен елемент на първо място). Специфично реализираните методи са:

- *void enqueueTimer(Timer* newTimer)* – Сложност: O(N)

    - Добавя дадената инстанция на таймера в контейнера като спазва наредба таймерите да са подредени по време на изпълнение.
- *Timer* detachFirst()* – Сложност: O(C)
    - Премахва първия таймер от списъка и го връща под формата на указател.
- *bool detachTimer(Timer* remTimer)* – Сложност: O(N)
    - Премахва подадения като параметър таймер от контейнера чрез последователно търсене. Ако таймерът не е намерен връща *false*.

## Клас *TimerScheduler*

Основен клас за организация на таймерите. Съдържа динамичен масив от нишки, който се ползват за изпълнението, контейнер от тип *TimersList* за таймери, динамичен масив за указатели към текущо изпълняващи се таймери и примитиви за синхронизация и паралелизиране. По-важни атрибути са:

- *std::thread* mainWorker*
    - Нишка, която непрестанно проверява дали следващият таймер в списъка трябва да бъде изпълнен. Започва изпълнението си със създаване на инстанция на класа. Допълнително необходима за работата на програмата (не е от броя подадени на конструктора).
- *std::vector<std::thread> threads*
    - Динамичен масив от заделените нишки.
- *std::vector<Timer*> runningTimers*
    - Динамичен масив от таймери, над които работят в текущия момент нишките. Ползва се за проверка ако трябва да се изтрие таймер.
- *std::atomic<int> freeThreads*
    - Брой готови за работа нишки. На практика се увеличава/намалява с всяка нишка, която е готова да вземе или вече е взела таймер за изпълнение
- *std::mutex mtxQueue*
    - Основен *mutex* за синхронизация на методите.
- *std::condition_variable cndvarNotifyTimer*
    - Условна променлива за нотификация от *mainWorker* към една от нишките, която е готова за изпълнение.

Специфични методи на класа са:

- *void executeTimerThread(int threadID)*
    - Функция, която се изпълнява за всяка от нишките. Подава се на нишка съответстващия ѝ пореден номер за съхранението на указател към текущия таймер на същото място в масива *runningTimers*.
- *void TimerScheduler::gearTrainExecutor()*
    - Функция, изпълнявана от основната нишка, чрез която се изчаква следващия таймер и се сигнализира на останалите нишки.

# Реализация

## Елементи на реализацията, алгоритми

**Управление на таймерите (създаване/унищожаване)**

Таймерите се създават директно от класа *TimerScheduler* при извикване на съответния метод. За различните видове таймери се използват и съответните параметри: за единичен таймер (*repeatCount* = 1, *repeat* = false), периодичен (*repeatCount* = 0, *repeat* = true) и такъв с N повторения (*repeatCount* = N, *repeat* = false). Таймерите се унищожават или от нишката, която ги е изпълнила последно, или от опашката, ако ѝ бъде зададено от потребителя да изтрие някой таймер.

**Инварианта за изпълнение последователно по време и за многократно своевременно изпълнение**

Таймерите се преподреждат в опашката спрямо момента, в който ще бъдат изпълнени, тоест се вмъкват на съответното място в свързания списък. Ако някой таймер има повече от едно изпълнение, то той все пак съществува само като един възел в опашката, който трябва да бъде изваден, за да се изпълни. Тоест ако периодът на повторение е по-малък от времето за повторение няма как да се наруши зададеното правило за изчакване, понеже няма как друга нишка да вземе същия таймер.

**Инварианта за изпълнение на таймери**

Таймерите не се изтриват ако не са изпълнени, тоест единствените начини да не се изпълни функция на таймер е или да не съществува свободна нишка (приемаме това за техническо ограничение), или *TimerScheduler* да бъде унищожен преди да се изпълни таймерът.

**Инварианта за многонишково изпълнение**

Организацията на проекта изисква да се използват множество изучавани системни примитиви за синхронизация и съответно те да предотвратят пресичането на операции на различни нишки.

**Възможност за отмяна на таймер**

Ако таймер трябва да бъде отменен и той се изпълнява в момента, то се покрива минималното изискване той да бъде невалиден, тоест в следващото си изпълнение да бъде премахнат. Ако се намира в опашката, то чрез последователно търсене ще бъде намерен и изтрит. Не съществуват други места, където да се съхраняват таймери и няма как да бъде погрешно изтрит таймер, понеже се идентифицира уникално с указател към заделената си памет.

## Качествен контрол

Към проекта са приложени само няколко кратки теста за валидността на основните методи, които трябва да бъдат реализирани в задачата. Използваната

библиотека *catch2* също е приложена. При основното изпълнение на проекта в *main* функцията е реализиран пример, при който многонишково (на 2 нишки) се добавят няколко таймера в създаден с 4 нишки *TimerScheduler* и след кратко изчакване (очаква се да се изпълнят за това време) програмата се прекратява.

# Допълнения

## Бъдещи планове за проекта

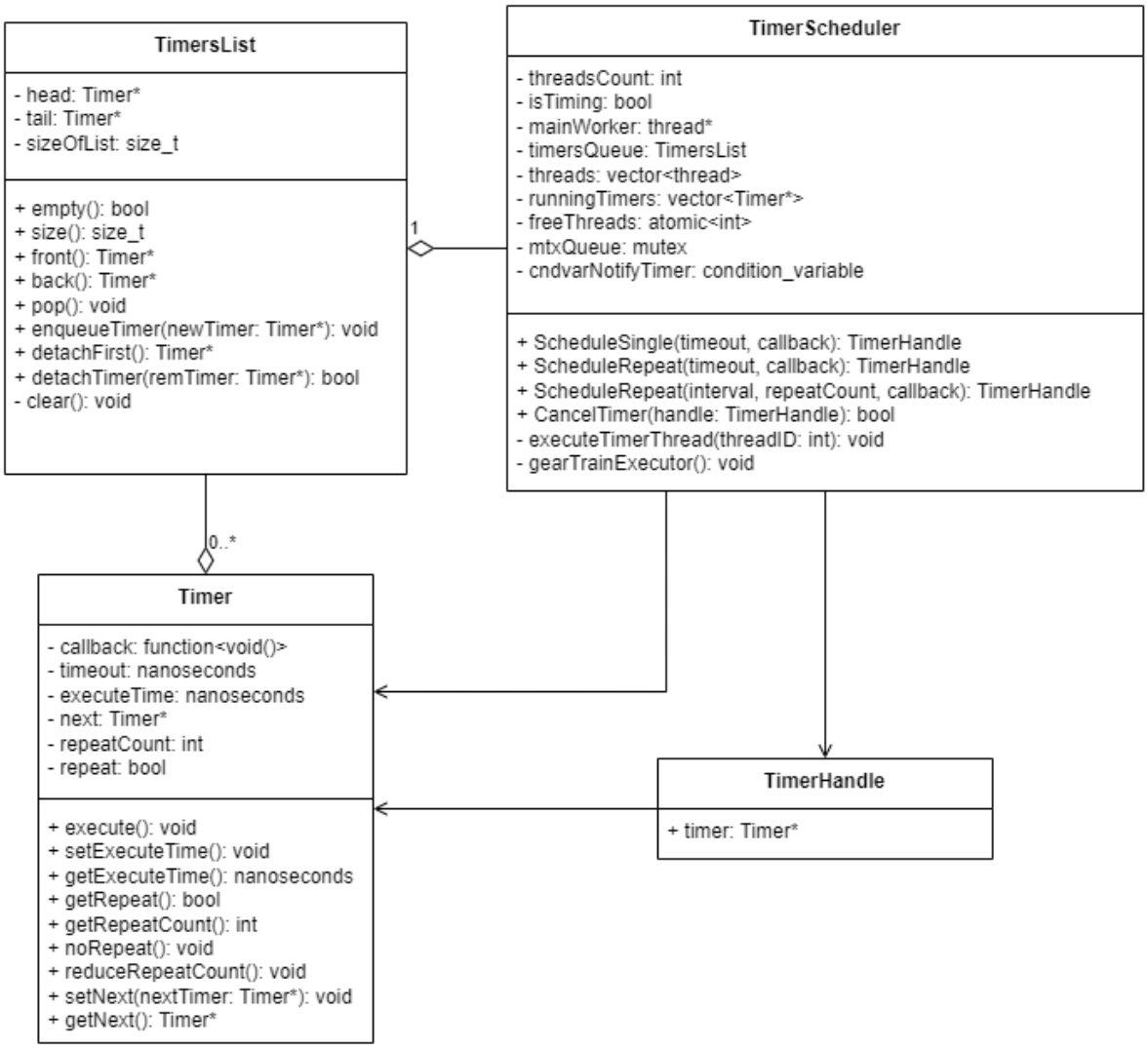
- Проучване на други възможно по-ефективни структури и модели на организация.
- Възможност за използване на допълнителна “спяща” нишка, която да предотврати постоянния цикъл на основната чрез изчакване в спящо състояние до необходимото време на следващия възможен таймер или събуждане при добавяне на нов таймер от потребител.
- Повече възможности за имплементация на таймери с допълнителни опции (напр. таймер с име, таймер с висок приоритет, таймер с повече от една функция, таймер с изпълнение в определен момент).
- Използване на по-ефективна структура за контейнер от тип приоритетна опашка.

## Източници


- Лекции и семинари по ССП, летен семестър 2022-2023г., СУ ФМИ
- stackoverflow.com
- en.cppreference.com
- www.cplusplus.com
- Wikipedia

# Приложение

## UML диаграма на класовете в проекта



# UML диаграма на основните последователности между нишките