Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 8

Тема: Ассинхронное программирование

Студент: Пермяков Никита Александрович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Цель:

Знакомство с асинхронным программированием;

Получение практических навыков в параллельной обработке данных;

Получение практических навыков в синхронизации потоков;

Создать приложение, которое будет считывать из стандартного ввода данные фигур и выводить их характеристики на экран, записывать в файл.

Программа должна:

1. Осуществлять ввод из стандартного ввода данных фигур
2. Программа должна создавать классы, соответствующие введенным данных фигур;
3. Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задается параметром командной строки.
4. При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;
5. Обработка должна производиться в отдельном потоке;
6. Реализовать два обработчика, которые должны обрабатывать данные буфера:
    a. Вывод информации о фигурах в буфере на экран;
    b. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.
7. Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.
8. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.
9. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;
10. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.
11. Реализовать в основном потоке (main) ожидание обработки буфера в потоке-обработчике. Т.е. после отправки буфера на обработку основной поток должен ждать, пока поток обработчик выведет данные на экран и запишет в файл.

Вариант 3:

- Трапеция
- Прямоугольник
- Ромб

2. Описание программы

Программа состоит из 4 файлов:
1) figures.h - содержит реализацию фигур и все операции связанные с ними.
2) factory.h - содержит класс для создания графических примитиве фигур.
3) subscriber.h - реализация класса, необходимого для передачи в поток как функтора, который необходим для выполнения обработки на отдельном потоке.
4) main.cpp - файл с взаимодействием с пользователем.

В программе имеются два потока, в поток subscriber_thread передаем функтор класса Subscriber, который после заполнения буфера будет выводить информацию о фигурах из буфера в файл и на экран.
При неверном вводе параметров фигуры будет происходить исключения.

В main.cpp содержится меню, позволяющее работать с вектором, содержащим в себе общие указатели на абстрактный класс, тип точек которого int. Перед входом в меню создается отдельный поток обработчиков для печати буфера. После создания, поток блокируется и ожидает, пока не придет сигнал из main о заполненности буфера (т.е. вектора), после чего он печатает содержимое и записывает его в файл.

3. Набор тестов

Пояснение:

Пользователь при запуске указывает размер буфера фигур - количество фигур, которое вмещает буфер. В программе пользователь может полностью заполнить буфер различными фигурами, после чего будет показана содержимое буфера на экран и экспорт буфера в файл с уникальным именем. После экспорта буфер очистится.

Test 1

```
add
1
0 0  5 0 5 10  0 10
menu
add
2
0 0  4 0  3 2  1 2
exit
```

Test 2

```
add
2
0 0  4 0  3 2  1 2
add
1
0 0  5 0 5 10  0 10
add
2
0 0  4 0  3 2  1 2
exit
```

1. Результаты выполнения тестов

Test 1

<menu> - menu

<add> Add figure

<exit> Exit

add

1

0 0 5 0 5 10 0 10

menu

add

2

0 0 4 0 3 2

figure type

1 - rectangle

2 - rhombus

3 - trapezoid

figure type

1 - rectangle

2 - rhombus

3 - trapezoid

1 2

exit**Test 2**

<menu> - menu

<add> Add figure

<exit> Exit

add

1

0 0 5 0 5 10 0 10

menu

add

2

0 0 4 0 3 2

figure type

1 - rectangle

2 - rhombus

3 - trapezoid

figure type

1 - rectangle

2 - rhombus

3 - trapezoid

figure type

1 - rectangle

2 - rhombus

3 - trapezoid

figure type

1 - rectangle

2 - rhombus

3 - trapezoid

figure type

1 - rectangle

2 - rhombus

3 - trapezoid

1 2

exit

5. Листинг программы

main.cpp

```
#include <condition_variable>
#include <fstream>
#include <iostream>
#include <memory>
#include <mutex>
#include <string>
#include <thread>
#include <vector>
#include <cstdlib>
#include "factory.h"
#include "figures.h"
#include "subscriber.h"

void menu() {
	std::cout << std::endl;
	std::cout << "<menu> - menu" << std::endl;
   std::cout << "<add> Add figure" << std::endl;
   std::cout << "<exit> Exit" << std::endl;
}

int main(int argc, char *argv[]) {
	if (argc != 2) {
		std::cout << "need define size" << std::endl;
		return 1;
	}
	uint32_t vector_size = std::atoi(argv[1]);
	Factory factory;
	Subscriber subscriber;
	subscriber.buffer.reserve(vector_size);
	subscriber.processes.push_back(std::make_shared<ProcessConsole>());
	subscriber.processes.push_back(std::make_shared<ProcessFile>());
```

```
	std::thread subscriber_thread(std::ref(subscriber));
	std::string cmd;
	std::cout << "add - exit" << std::endl;
	while (std::cin >> cmd) {
		std::unique_lock<std::mutex> main_lock(subscriber.mtx);
		if (cmd == "exit" || cmd == "q" || cmd == "e") {
			subscriber.end = true;
			subscriber.cv.notify_all();
			break;
		} else if (cmd == "add" || cmd == "a") {
			int figure_type;

			for (uint32_t id = 0; id < vector_size; ++id) {
				std::cout << "figure type" << std::endl
				<< "\t1 - rectangle" << std::endl
				<< "\t2 - rhombus" << std::endl
				<< "\t3 - trapezoid" << std::endl;
				std::cin >> figure_type;
				switch (figure_type)
				{
				case 1:{
					std::pair<double, double> *vertices = new std::pair<double,
double>[4];
					for (int i = 0; i < 4; ++i)
						std::cin >> vertices[i].first >> vertices[i].second;

					try {
						subscriber.buffer.push_back(factory.FigureCreate(rec,
vertices, id));
					} catch (std::logic_error &e) {
						std::cout << e.what() << std::endl;
						--id;
					}
					break;
```

```
                }
                case 2: {
                        std::pair<double, double> *vertices = new std::pair<double,
double>[4];
                        for (int i = 0; i < 4; ++i)
                                std::cin >> vertices[i].first >> vertices[i].second;

                        try {

        subscriber.buffer.push_back(factory.FigureCreate(rhomb, vertices, id));
                        } catch (std::logic_error &e) {
                                std::cout << e.what() << std::endl;
                                id--;
                        }
                        break;
                }
                case 3: {
                        std::pair<double, double> *vertices = new std::pair<double,
double>[4];
                        for (int i = 0; i < 4; i++)
                                std::cin >> vertices[i].first >> vertices[i].second;

                        try {

        subscriber.buffer.push_back(factory.FigureCreate(trap, vertices, id));
                        } catch (std::logic_error &e) {
                                std::cout << e.what() << std::endl;
                                id--;
                        }
                        break;
                }
                default:
                        break;
                }
        }
```

```
                        if (subscriber.buffer.size() == vector_size) {
                                //main_lock.unlock();
                                subscriber.cv.notify_all();
                                subscriber.cv.wait(main_lock, [&subscriber]() {
                                        return subscriber.success == true;
                                });
                                subscriber.success = false;
                        }
                }
        }
        subscriber_thread.join();
        return 0;
}
```

subsriber.h

```
#pragma once
#include<vector>
#include<string>
#include<iostream>
#include<fstream>
#include <thread>
#include <mutex>

struct ProcessSubscribers {
        virtual void Process(std::vector<std::shared_ptr<Figure>> &buffer) = 0;
        virtual ~ProcessSubscribers() = default;
};

struct ProcessConsole : ProcessSubscribers {
        void Process(std::vector<std::shared_ptr<Figure>> &buffer) override {
                for (const auto figure : buffer) {
                        figure->Print(std::cout);
                }
        }
```

```
};

struct ProcessFile : ProcessSubscribers {
	void Process(std::vector<std::shared_ptr<Figure>> &buffer) override {
		std::ofstream os(std::to_string(this->name));
		for (const auto figure : buffer) {
			figure->Print(os);
		}
		++this->name;
	}

private:
	uint16_t name = 0;
};

struct Subscriber {
	void operator()() {
		for(;;) {
			std::unique_lock<std::mutex> guard(mtx);
			cv.wait(guard, [&](){
				return buffer.size() == buffer.capacity() || end;
			});

			if (end)
				break;

			for (uint16_t i = 0; i < processes.size(); ++i)
				processes[i]->Process(buffer);

			buffer.clear();
			success = true;
			cv.notify_all();
		}
	}
```

```
	bool end = false;
	bool success = false;
	std::vector<std::shared_ptr<Figure>> buffer;
	std::vector<std::shared_ptr<ProcessSubscribers>> processes;
	std::condition_variable cv;
	std::mutex mtx;
};
```

factory.h

```
#pragma once
#include "figures.h"
#include "Rectangle.h"
#include "Rhombus.h"
#include "Trapezoid.h"

class Factory {
public:
	std::shared_ptr<Figure> FigureCreate(FigureType type) const {
		std::shared_ptr<Figure> res;
		if (type == rec) {
			res = std::make_shared<Rectangle>();
		} else if (type == rhomb) {
			res = std::make_shared<Rhombus>();
		} else if (type == trap) {
			res = std::make_shared<Trapezoid>();
		}
		return res;
	}

	std::shared_ptr<Figure> FigureCreate(FigureType type, std::pair<double, double> *vertices,
int id) const {
		std::shared_ptr<Figure> res;
		if (type == rec) {
```

```
                res = std::make_shared<Rectangle>(vertices[0], vertices[1], vertices[2],
vertices[3], id);
            } else if (type == rhomb) {
                res = std::make_shared<Rhombus>(vertices[0], vertices[1], vertices[2],
vertices[3], id);
            } else if (type == trap) {
                res = std::make_shared<Trapezoid>(vertices[0], vertices[1], vertices[2],
vertices[3], id);
            }
            return res;
        }
};
```

figures.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <utility>
#include <memory>
#include <cmath>
#include <stdexcept>


enum FigureType {
    rec,
    rhomb,
    trap,
};

class Figure {
    public:
    virtual double Area() const = 0;
    virtual std::pair<double, double> Center() const = 0;
    virtual std::ostream& Print(std::ostream& out) const = 0;
```

```
    virtual void Serialize(std::ofstream& os) const = 0;
    virtual void Deserialize(std::ifstream& is) = 0;
    virtual int getId() const = 0;
    virtual ~Figure() = default;
};

std::pair<double, double> getCenter(
    const std::pair<double, double> *vertices,
    uint16_t& n
    ) {
    double x = 0, y = 0;
    for (uint16_t i = 0; i < n; ++i) {
        x += vertices[i].first;
        y += vertices[i].second;
    }
    return {x / n, y / n};
}

std::pair<double, double> operator- (
    const std::pair<double, double> &p1,
    const std::pair<double, double> &p2
    ) {
    return {p1.first - p2.first, p1.second - p2.second};
}

bool collinear(
    const std::pair<double, double> &a,
    const std::pair<double, double> &b,
    const std::pair<double, double> &c,
    const std::pair<double, double> &d
    ){
    return (b.second-a.second)*(d.first-c.first) - (d.second-c.second)*(b.first-a.first) <= 1e-9;
}
```

```
bool perpendicular(
    const std::pair<double, double> &a,
    const std::pair<double, double> &b,
    const std::pair<double, double> &c,
    const std::pair<double, double> &d
    ){
    std::pair<double, double> AC = c - a;
    std::pair<double, double> BD = d - b;

    double normaAC = sqrt(pow(AC.second, 2) + pow(AC.first, 2));
    double normaDB = sqrt(pow(BD.second, 2) + pow(BD.first, 2));

    double hightAC = AC.second / normaAC;
    double widthAC = AC.first / normaAC;
    double hightBD = BD.second / normaDB;
    double widthBD = BD.first / normaDB;

    double cos_theta = hightAC * hightBD + widthAC * widthBD;

    return abs(cos_theta) < 1e-9;
}

double dist(
    const std::pair<double, double> &a,
    const std::pair<double, double> &b
    ){
    return sqrt(((b.first - a.first) * (b.first - a.first)) + ((b.second - a.second) * (b.second - a.second)));
}

bool operator==(
    const std::pair<double, double> &a,
    const std::pair<double, double> &b
    ){
```

```
    return (a.first == b.first) && (a.second == b.second);
}

std::ostream& operator<<(std::ostream &o, const std::pair<double, double> &p){
    o << "<" << p.first << ", " << p.second << ">";
    return o;
}

std::istream& operator>>(std::istream &is, std::pair<double, double> &p){
    std::string checker;
    p.first = static_cast<double>(std::stod(checker));
    p.second = static_cast<double>(std::stod(checker));
    return is;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(lab8)
add_executable(lab8 main.cpp)
set_property(TARGET lab8 PROPERTY CXX_STANDARD 11)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -g")
```

6. Ссылка на репозиторий

https://github.com/nikit34/oop_exercise_08

7. Объяснение результатов работы программы

Программа создает два потока выполнения, которые не могут исполнятся паралелльно, но последовательность может быть сразу для нескольких потоков, с разделением одного адресного пространства для избежания ошибок и блокировок. Один поток уведомляет второй, при заполнении буффера и ждет ответа от него. Второй поток вызывает вывод в консоль и файл. Происходит очистка буффера с уведомлением от второго потока.

8. Вывод

В процессе работы, я вывел общее определение синхронного и асинхронного выполнения:

Асинхронность используется везде, где нужно ждать. Чтобы не ждать ответа, а выполнять другие операции, и продолжить выполнение по готовности
Примеры:

- События интерфейса
- Длинные сложные вычисления.

Список литературы

Проектирование классов C++ [Электронный ресурс]. URL: https://metanit.com/cpp/tutorial/5.14.php

(дата обращения: 19.12.2020).

2. Академическое программирование C++ [Электронный ресурс]. URL:http://www.c-cpp.ru/books/akkadem.pdf

(дата обращения: 19.12.2020).