

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 8: «Асинхронное программирование»

| | |
|----------------|----------------------------|
| Группа: | М8О-208Б-18, №2 |
| Студент: | Алексеева Мария Алексеевна |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: | |
| Дата: | 29.12.2019 |

Москва, 2019

1. **Тема:** Асинхронное программирование
2. **Цель работы:** Знакомство с асинхронным программированием;
Получение точечных навыков в параллельной обработке данных;
Получение практических навыков в синхронизации потоков.
3. **Задание (вариант № 2):**
Фигуры — прямоугольник, трапеция, квадрат.
4. **Адрес репозитория на GitHub**
https://github.com/PowerMasha/oop_exercise_08
5. **Код программы на C++**
main.cpp

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

int main(int argc, char** argv){
    int SizeVector = std::atoi(argv[1]); //размер вектора
    std::vector<std::shared_ptr<figure>> Figure; //вектор-буфер для хранения фигур
    std::condition_variable k1; // примитивы синхронизации
    std::condition_variable k2;
    std::mutex mutex;

    factory Factory; // фабрика создания фигур

    bool done = false;
    char cmd;
    int in = 1;
    std::vector<std::shared_ptr<Subscriber>> subs; // вектор с обработчиками
    subs.push_back(std::make_shared<Consol>());
    subs.push_back(std::make_shared<File>());
    std::thread subscriber([&](){
        std::unique_lock<std::mutex> subscriber_lock(mutex); // универсальная оболочка для
        владения мьютексом, поток-обработчиков
        while(!done) {
            k1.wait(subscriber_lock); // блокирует текущий поток до тех пор, пока переменная не
            будет пробуждена

            if (done) {
```

```

        k2.notify_all(); // уведомляет все потоки ожидающие k2
        break;
    }
    for (unsigned int i = 0; i < subs.size(); ++i) {
        subs[i]->output(Figure);
    }
    in++;
    Figure.resize(0);
    k2.notify_all();
}
});
while(cmd != 'q') {
    std::cout << "'q'-quit, 'c'-continue , Figures: square, trapez, rectangle" << std::endl;
    std::cin >> cmd;
    if (cmd != 'q') {
        std::unique_lock<std::mutex> main_lock(mutex); // главный поток
        for (int i = 0; i < SizeVector; i++) {
            Figure.push_back(Factory.FigureCreate(std::cin));
            std::cout << "Added" << std::endl;
        }
        k1.notify_all();
        k2.wait(main_lock);
    }
}
done = true;
k1.notify_all();
subscriber.join(); //Блокирует текущий поток до тех пор, пока поток, обозначенный *this,
не завершит свое выполнение
return 0;
}
subscriber.h

```

```

#ifndef SUBSCRIBERS_H
#define SUBSCRIBERS_H
#include <fstream>

```

```

class Sub{
public:
    virtual void output(std::vector<std::shared_ptr<figure>>& Vec) = 0;
    virtual ~Sub() = default;
};

```

```

class Consol : public Sub {
public:
    void output(std::vector<std::shared_ptr<figure>>& Vec) override {
        for (auto& figure : Vec) {
            figure->print(std::cout);
        }
    }
}

```

```
};
```

```
class File : public Sub{
public:
    File() : in(1) {}
    void output(std::vector<std::shared_ptr<figure>>& Vec) override {
        std::string filename;
        filename = std::to_string(in);
        filename += ".txt";
        std::ofstream file;
        file.open(filename);
        for (auto &figure : Vec) {
            figure->print(file);
        }
        in++;
    }
private :
    int in;
};
```

```
#endif
```

```
trapez.h
```

```
#ifndef OOP_TRAPEZ_H
#define OOP_TRAPEZ_H
#include <cmath>
#include <iostream>
#include "point.h"
#include "figure.h"
```

```
struct Trapez : figure{

    point a1,a2,a3,a4;

    point center() const {
        double x,y;
        x = (a1.x + a2.x + a3.x + a4.x ) / 4;
        y = (a1.y + a2.y + a3.y + a4.y ) / 4;
        point p(x,y);
        return p;
    }
}
```

```

void print(std::ostream& os) const {
    os << "trapez " << a1 << " " << a2 << " " << a3 << " " << a4 << "\n";
}

void printFile(std::ofstream &of) const {
    of << "trapez " << a1 << " " << a2 << " " << a3 << " " << a4 << "\n";
}

double area() const {
    return (-0.5) * ((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a1.y) - ( a1.y*a2.x
+ a2.y*a3.x + a3.y*a4.x + a4.y*a1.x ));
}

Trapez(std::istream& is) {
    is >> a1 >> a2 >> a3 >> a4 ;
}

Trapez(std::ifstream& is) {
    is >> a1 >> a2 >> a3 >> a4 ;
}
};

#endif //OOP_TRAPEZ_H

#ifndef OOP_RECTANGLE_H
#define OOP_RECTANGLE_H

#include <cmath>
#include "point.h"
#include "figure.h"

struct Rectangle : figure {

    point a1, a2, a3, a4;

    point center() const {
        double x, y;
        x = (a1.x + a2.x + a3.x + a4.x) / 4;
        y = (a1.y + a2.y + a3.y + a4.y) / 4;
        point p(x, y);
        return p;
    }

    void print(std::ostream &os) const {

```

```

        os << "rectangle " << a1 << " " << a2 << " " << a3 << " " << a4 << "\n";
    }

    void printFile(std::ofstream &of) const {
        of << "rectangle " << a1 << " " << a2 << " " << a3 << " " << a4 << "\n";
    }

    double area() const {
        return (-0.5) * ((a1.x * a2.y + a2.x * a3.y + a3.x * a4.y + a4.x * a1.y) -
            (a1.y * a2.x + a2.y * a3.x + a3.y * a4.x + a4.y * a1.x));
    }

    Rectangle(std::istream &is) {
        is >> a1 >> a2 >> a3 >> a4;
    }

    Rectangle(std::ifstream &is) {
        is >> a1 >> a2 >> a3 >> a4;
    }
};

#endif //OOP_RECTANGLE_H

```

square.h

```

#ifndef OOP_SQUARE_H
#define OOP_SQUARE_H

#include <cmath>
#include "point.h"
#include "figure.h"

struct Square : figure {
public:
    point a1, a2, a3, a4;

    point center() const {
        double x, y;
        x = (a1.x + a2.x + a3.x + a4.x) / 4;
        y = (a1.y + a2.y + a3.y + a4.y) / 4;
        point p(x, y);
        return p;
    }
}

```

```

void print(std::ostream &os) const {
    os << "square " << a1 << " " << a2 << " " << a3 << " " << a4 << "\n";
}
void printFile(std::ofstream &of) const {
    of << "square " << a1 << " " << a2 << " " << a3 << " " << a4 << "\n";
}

double area() const {
    return (-0.5) * ((a1.x * a2.y + a2.x * a3.y + a3.x * a4.y + a4.x * a1.y) -
        (a1.y * a2.x + a2.y * a3.x + a3.y * a4.x + a4.y * a1.x));
}

Square(std::istream &is) {
    is >> a1 >> a2 >> a3 >> a4;
}

Square(std::ifstream &is) {
    is >> a1 >> a2 >> a3 >> a4;
}
};
#endif //OOP_SQUARE_H

```

factory.h

```

#ifndef OOP_FACTORY_H
#define OOP_FACTORY_H

```

```

#include <memory>
#include <iostream>
#include <fstream>
#include "square.h"
#include "rectangle.h"
#include "trapez.h"
#include <string>

```

```

struct factory {

    std::shared_ptr<figure> FigureCreate(std::istream &is) {
        std::string name;
        is >> name;
        if ( name == "rectangle" ) {
            return std::shared_ptr<figure> ( new Rectangle(is));
        } else if ( name == "trapez" ) {
            return std::shared_ptr<figure> ( new Trapez(is));
        } else if ( name == "square" ) {

```

```

        return std::shared_ptr<figure> ( new Square(is));
    } else {
        throw std::logic_error("There is no such figure\n");
    }
}

};

```

```

#endif //OOP_FACTORY_H

```

figure.h

```

#ifndef OOP_FIGURE_H
#define OOP_FIGURE_H
#include <iostream>
#include "point.h"
#include <fstream>

```

```

struct figure {
    virtual point center() const = 0;
    virtual void print(std::ostream&) const = 0 ;
    virtual void printFile(std::ofstream&) const = 0 ;
    virtual double area() const = 0;
    virtual ~figure() = default;
};

```

```

#endif //OOP_FIGURE_H

```

point.h

```

#ifndef OOP_POINT_H
#define OOP_POINT_H

```

```

#include <iostream>

```

```

struct point {
    double x, y;
    point (double a,double b) { x = a, y = b;};
    point() = default;

```

```

};
//std::istream& operator >> (std::istream& is,point& p );
//std::ostream& operator << (std::ostream& os,const point& p);

```



```

std::istream& operator >> (std::istream& is, point& p ) {
    return is >> p.x >> p.y;
}

std::ostream& operator << (std::ostream& os, const point& p) {
    return os << p.x << ' ' << p.y;
}
#endif

```

CmakeLists.txt

```

cmake_minimum_required(VERSION 3.10.2)
project(oop_exercise_08)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -g3 -Wextra -
pthread")
add_executable(oop_exercise_08
    main.cpp
    point.h
    trapez.h
    figure.h
    rectangle.h
    square.h
    factory.h
    subscriber.h)

```

6. Результаты выполнения тестов

masha@masha-VirtualBox:~/2kurs/oop_exercise_08/tmp\$./oop_exercise_08 3

Input 'q' for quit, or 'r' to continue

r

trapez 0 0 1 1 2 1 3 0

rectangle 0 0 0 0 0 0 0 0

square 0 0 0 1 1 1 1 0

r

rectangle 1 1 1 1 1 1 1 1

square 0 0 0 2 2 2 2 0

rectangle 2 2 2 2 2 2 2 2

q

Added

Added

Added

trapez 0 0 1 1 2 1 3 0

rectangle 0 0 0 0 0 0 0 0

square 0 0 0 1 1 1 1 0

Input 'q' for quit, or 'r' to continue

```
Added
Added
Added
rectangle 1 1 1 1 1 1 1 1
square 0 0 0 2 2 2 2 0
rectangle 2 2 2 2 2 2 2 2
Input 'q' for quit, or 'r' to continue
```

```
masha@masha-VirtualBox:~/2kurs/oop_exercise_08/tmp$ ./oop_exercise_08 5
< ../test_02.txt
```

```
Input 'q' for quit, or 'r' to continue
```

```
Added
```

```
Added
```

```
Added
```

```
Added
```

```
Added
```

```
square 0 0 0 1 1 1 1 0
```

```
square 0 0 0 1 1 1 1 0
```

```
square 0 0 0 1 1 1 1 0
```

```
square 0 0 0 1 1 1 1 0
```

```
square 0 0 0 1 1 1 1 0
```

```
Input 'q' for quit, or 'r' to continue
```

```
Added
```

```
Added
```

```
Added
```

```
Added
```

```
Added
```

```
square 0 0 0 1 1 1 1 0
```

```
square 0 0 0 1 1 1 1 0
```

```
square 0 0 0 1 1 1 1 0
```

```
square 0 0 0 1 1 1 1 0
```

```
square 0 0 0 1 1 1 1 0
```

```
Input 'q' for quit, or 'r' to continue
```

7. Объяснение результатов работы программы - вывод

В `subscriber.h` реализованы два подписчика — обработчика `Consol` и `File`. Один осуществляет вывод данных на консоль, другой в текстовый файл. Синхронизация процессов осуществляется посредством двух условных переменных и мьютекса.

В ходе выполнения лабораторной работы были приобретены начальные навыки работы с асинхронным программированием, получены некоторые навыки в параллельной обработке данных, получены практические навыки в синхронизации потоков.