Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Лабораторной работе № 08
по курсу «Объектно-ориентированное программирование»

Тема:
«Асинхронное программирование »

Студент:	Пшеницын А. А.
Группа:	М80-208Б-18
Преподаватель:	Журавлев А. А.
Вариант:	17
Оценка:	
Дата:	

Москва
2019

Цель:

Цель:
• Знакомство с асинхронным программированием;
• Получение практических навыков в параллельной обработке данных;
• Получение практических навыков в синхронизации потоков;

Задание

Создать приложение, которое будет считывать из стандартного ввода данные фигур, согласно варианту задания, выводить их характеристики на экран и записывать в файл. Фигуры могут задаваться как своими вершинами, так и другими характеристиками (например, координата центра, количество точек и радиус). Программа должна:
1. Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;
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. Например, для буфера размером 10 фигур:
oop_exercise_08 10
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. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;
9. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.
10. 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.

Код

main.cpp

```
#include <iostream>
#include <memory>
#include <vector>
#include <condition_variable>
#include <thread>
#include <mutex>
#include "factory.h"
#include "figure.h"
#include "trigon.h"
#include "rectangle.h"
#include "quadrate.h"

struct processor {
   virtual void process(std::shared_ptr<std::vector<std::shared_ptr<fig>>> buffer) = 0;

};

struct stream_processor : processor {
```

```
    void process(std::shared_ptr<std::vector<std::shared_ptr<fig>>> buffer) override{
        for (const auto& figure : *buffer) {
            figure -> print(std::cout);
        }
    }
};

struct file_processor : processor {
    void process(std::shared_ptr<std::vector<std::shared_ptr<fig>>> buffer) override{
        std::ofstream fout;
        fout.open(std::to_string(counter) + ".txt");
        ++counter;
        if (!fout.is_open()) {
            std::cout << "File not opened\n";
            return;
        }
        for (const auto& figure : *buffer) {
            figure -> print(fout);
        }

    }
private:
    int counter = 0;
};

struct executor {
    void operator()(){
        while(1) {
            std::unique_lock<std::mutex> lock(mtx);
            cv.wait(lock,[&]{ return (buffer != nullptr || flag);});
            if (flag) {
                break;
            }
            for (const auto& processor_elem: processors) {
                processor_elem->process(buffer);
            }
            buffer = nullptr;
            cv.notify_all();
        }
    }

    std::mutex& get_mtx(){
        return mtx;
    }

    void push_buf(std::shared_ptr<std::vector<std::shared_ptr<fig>>> buf){
         buffer = buf;
    }

    std::shared_ptr<std::vector<std::shared_ptr<fig>>> get_buf(){
        return buffer;
    }
```

```
    bool empty_buf(){
        return buffer == nullptr;
    }

    void notify_all(){
        cv.notify_all();
    }

    void flag_true(){
        flag = true;
    }

    std::vector<std::shared_ptr<processor>> processors;
    std::shared_ptr<std::vector<std::shared_ptr<fig>>> buffer;
    std::mutex mtx;
    std::condition_variable cv;
    bool flag = false;

};



int main(int argc,char* argv[]) {
    if (argc != 2) {
        std::cout << "ERROR";
        return 1;
    }
    int buffer_size = std::stoi(argv[1]);
    std::shared_ptr<std::vector<std::shared_ptr<fig>>> buffer;
    buffer = std::make_shared<std::vector<std::shared_ptr<fig>>>();
    buffer -> reserve(buffer_size);
    factory fact;
    executor sub;
    sub.processors.push_back(std::make_shared<stream_processor>());
    sub.processors.push_back(std::make_shared<file_processor>());
    std::thread sthread(std::ref(sub));

    while(1) {
        std::string comm;
        std::unique_lock<std::mutex> mut(sub.get_mtx());
        std::cin >> comm;
        if (comm == "add") {
            std::shared_ptr<fig> fig = fact.fig_create(std::cin);
            if(fig == nullptr){
                continue;
            }
            buffer -> push_back(fig);
            if (buffer -> size() == buffer_size) {
                sub.push_buf(buffer);
                sub.notify_all();
                sub.cv.wait(mut, [&](){ return sub.empty_buf();});
```

```
            buffer -> clear();
          }
        } else if (comm == "exit"){
          break;
        } else {
          std::cout << "unknown command\n";
        }
    }

    sub.flag_true();
    sub.notify_all();
    sthread.join();


    return 0;
}
```

Ссылка на репозиторий на GitHub

https://github.com/AlexPshen/oop_exercise_06.git

Тесты

test_01.txt

```
add trigon 0 0 0 1 1 0
add rectangle 0 0 1 1 1 0 0 1
add quadrate 0 0 0 1 1 0 1 1
add trigon 0 0 0 2 2 0
exit
```

res_01.txt

```
add trigon 0 0 0 1 1 0
add rectangle 0 0 1 1 1 0 0 1
trigon: (0, 0) (0, 1) (1, 0)
rectangle: (0, 0) (1, 1) (1, 0) (0, 1)
add quadrate 0 0 0 1 1 0 1 1
add trigon 0 0 0 2 2 0
quadrate: (0, 0) (0, 1) (1, 0) (1, 1)
trigon: (0, 0) (0, 2) (2, 0)
exit
```

test_02.txt

```
add trigon 0 0 0 1 1 0
add rectangle 0 0 0 2 2 0 2 2
add quadrate 0 0 1 0 0 1 1 1
exit
```

res_02.txt

```
add trigon 0 0 0 1 1 0
```

add rectangle 0 0 0 2 2 0 2 2
add quadrate 0 0 1 0 0 1 1 1
trigon: (0, 0) (0, 1) (1, 0)
rectangle: (0, 0) (0, 2) (2, 0) (2, 2)
quadrate: (0, 0) (1, 0) (0, 1) (1, 1)
exit

Объяснение результатов работы программы

В данной лабораторной работе работают два потока, один из которых служит для считывания и сохранения фигуры в буфер, а другой, когда буфер заполняется на размер, заданный пользователем, печатает координаты данных фигур в файл и на консоль.

Вывод

В данной лабораторной работе мы познакомились с асинхронным программированием. В синхронном коде каждая операция ожидает окончания предыдущей. Поэтому вся программа может зависнуть, если одна из команд выполняется очень долго. Асинхронный код убирает блокирующую операцию из основного потока программы, так что она продолжает выполняться, но где-то в другом месте, а обработчик может идти дальше.