Московский авиационный институт
(Национальный исследовательский университет)

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

Лабораторная работа № 8

Тема: Асинхронное программирование

Студент: Савченко Илья Владимирович
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

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. 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 1:

Треугольник, квадрат, прямоугольник

2. Описание программы

main.cpp	
int main(... args)	Драйвер код
figures.h	
struct Figure	Абстрактный класс фигур
void Figure::Print(); void Figure::Write(ostream&)	Виртуальные функции записи фигуры в потоки
class Triangle<T> : Figure class Square<T> : Figure class Rectangle<T> : Figure	Реализации фигур T — тип коорд.
factory	
std::shared_ptr<Figure> CreateFigure(char, T)	Паттерн простой фабрики для создания объектов-фигур
handler.h	
class Handler	Класс для обработки операций в нескольких потоках, хранит вектор фигур, а также вектор функций для мультитредового исполнения
void Handler::AddFunction(std::function&)	Добавление функции
void Handler::Push(FigPtr)	Добавление фигуры (FigPtr = shared_ptr)
static void Printing(Handler*)	Функция, вызываемая из тредов, выписывает фигуры в потоки

3. Набор тестов

На вход подаются лишь фигуры и их параметры (стороны фигур)
Также можно запустить программу с аргументом числом — размером буфера

Когда буфер заполняется фигурами, эти фигуры записываются через треды в стд. вывод и файл.

Содержимое файлов идентично выводу в стд.

4. Результаты выполнения тестов

```
h - this help text
x - exit
t - add triangle
q - add square
r - add reactangle (2 sides)
>  t 6
Added triangle
> q 3
Added square
> r 9 12
Added rectangle
> q 5
Tri: {0, 0}  {0, 6}  {6, 0}
Sqr: {0, 0}  {3, 0}  {3, 0}  {3, 3}
Rct: {0, 0}  {9, 0}  {9, 12}  {0, 12}
Sqr: {0, 0}  {5, 0}  {5, 0}  {5, 5}
Added square
> t 9
Added triangle
> r 2 9
Added rectangle
> t 6
Added triangle
> x
Stop
Tri: {0, 0}  {0, 9}  {9, 0}
Rct: {0, 0}  {2, 0}  {2, 9}  {0, 9}
Tri: {0, 0}  {0, 6}  {6, 0}
```

Созданы файлы figs_11.txt и figs_17.txt

5. Листинг программы

```
////////////// main.cpp
/**
 * Савченко И.В.
 * М8О-208Б-19
 * https://github.com/ShyFly46/oop_exercise_08
 *
 * Вариант 1:
 * "Асинхронное программирование"
 * Треугольник, квадрат, прямоугольник
 **/

#include <pthread.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <string>
#include <vector>
#include <sstream>    // stringstream

#include "factory.h"
#include "handler.h"

using namespace std;

using FigPtr = shared_ptr<Figure>;

void help() {
      cout
       //<< "t q r\n"
       << "h - this help text\n"
       << "x - exit\n"
       << "t - add triangle\n"
       << "q - add square\n"
       << "r - add reactangle (2 sides)\n";
}


int main(int argc, char** argv) {
      size_t bufferSize = 3;

      if (argc == 2) {
      stringstream argc;
      argc << argv[1];
      argc >> bufferSize;
      }

      Handler handler(bufferSize);

      handler.AddFuction([](const vector<FigPtr>& figs) {
      for (auto& fig : figs) {
            fig->Print();
      }
      });

      handler.AddFuction([](const vector<FigPtr>& figs) {
      int fileNumber = rand() % 30;
```

```
      string fileName = "figs_" + to_string(fileNumber) + ".txt";
      ofstream file(fileName);
      for (auto items : figs) {
            items->Write(file);
      }
      file.close();
      });

      help();
    char cmd;
    cout << "> ";
      while(cin >> cmd){
      switch(cmd){
      case 't':
       case 'q':
       case 'r':
             {
                   double side;
                   cin >> side;
                   handler.Push(CreateFigure(cmd, side));
             }
             cout << "Added ";
             if   (cmd == 't') cout << "triangle\n";
             else if(cmd == 'q') cout << "square\n";
             else       /*'r'*/cout << "rectangle\n";
            break;
       case 'h':
             help();
             break;
      case 'x':
            cout << "Stop\n";
            return 0;
      default:
            cout << "???\n";
            break;
      }
      cout << "> ";
      }
}
////////////// handler.h
#pragma once
#include <condition_variable>
#include <mutex>
#include <shared_mutex>
#include <thread>
#include <list>
#include <functional>
#include <atomic>
#include <vector>
#include <iostream>

#include "figures.h"

class Handler {
    using FigPtr = std::shared_ptr<Figure>;
      std::mutex               mutex;
      std::thread              thread;
      std::condition_variable  condVar;
```

```
	std::vector<FigPtr>	figures;
	std::vector <std::function <void(std::vector<FigPtr>&)> >
				handlers;

	size_t max = 0;

public:
	bool running;

	Handler(size_t size = 3)
	: max(size),
	running(true),
	 thread(std::thread(Printing, this))
    {}

	~Handler() {
	running = false;
	condVar.notify_one();
	thread.join();
	}

	void AddFuction(std::function <void(const std::vector<FigPtr>&)>&&
func) {
	handlers.push_back(func);
	}

	void Push(FigPtr fig) {
	std::unique_lock<std::mutex> uLock(mutex);
	figures.push_back(fig);
	if (IsFull()) {
		condVar.notify_one();
		condVar.wait(uLock, [this]() {
		return figures.empty();
		});
	}
	}

	bool IsFull() {
	return figures.size() >= max;
	}

	static void Printing(Handler* t) {
	while (t->running) {
		std::unique_lock<std::mutex> lock(t->mutex);
		t->condVar.wait(lock, [t]() {
		return t->IsFull()||!t->running;
		});
		for (auto& item : t->handlers) {
		item(t->figures);
		}

		t->figures.clear();
		lock.unlock();
		t->condVar.notify_one();
	}
	}

};
```

```
////////////// figures.h
#pragma once
#include <iostream>
#include <fstream>

using namespace std;

struct Figure {
    virtual void Print(){
       Write(cout);
    }
    virtual void Write(ostream& file) = 0;
};

template<typename T>
class Square : public Figure {
    using coords = pair<T, T>;
    coords a, b, c, d;

public:
    Square(T s = 1){
       a.first  = 0;
       a.second = 0;

       b.first  = s;
       b.second = 0;

       c.first  = s;
       c.second = 0;

       d.first  = s;
       d.second = s;
    }

    void Write(ostream& file) override {
       file << "Sqr: "
            << "{" << a.first << ", " << a.second << "}  "
            << "{" << b.first << ", " << b.second << "}  "
            << "{" << c.first << ", " << c.second << "}  "
            << "{" << d.first << ", " << d.second << "}\n";
    }
};

template<typename T>
class Triangle : public Figure {
    using coords = pair<T, T>;
    coords a, b, c;

public:
    Triangle(T s = 1) {
       a.first = a.second  = 0;

       b.first  = 0;
       b.second = s;

       c.first  = s;
       c.second = 0;
    }
```

```
    void Write(ostream& file) override {
       file << "Tri: "
            << "{" << a.first << ", " << a.second << "}  "
            << "{" << b.first << ", " << b.second << "}  "
            << "{" << c.first << ", " << c.second << "}\n";
    }
};

template<typename T>
class Rectangle : public Figure {
    using coords = pair<T, T>;
    coords a, b, c, d;

public:
    Rectangle(T s = 1, T h = 1){
       a.first = a.second = b.second = d.first = 0;
       b.first = c.first = s;
       c.second = d.second = h;
    }

    void Write(ostream& file) override {
       file << "Rct: "
            << "{" << a.first << ", " << a.second << "}  "
            << "{" << b.first << ", " << b.second << "}  "
            << "{" << c.first << ", " << c.second << "}  "
            << "{" << d.first << ", " << d.second << "}\n";
    }
};
////////////// factory.h
#pragma once
#include <memory>
#include "figures.h"

using namespace std;

template<typename T>
std::shared_ptr<Figure> CreateFigure(char type, T side){
    std::shared_ptr<Figure> fig;
    switch(type){
       default:
       case 'q':
             return make_shared<Square<T>>(side);
       case 't':
             return make_shared<Triangle<T>>(side);
       case 'r':
             {
                   T height;
                   cin >> height;
                   return make_shared<Rectangle<T>>(side, height);
             }
    }
}
```

Список литературы

1. Добро пожаловать в параллельный мир. Часть 1: Мир многопоточный [Электронный ресурс].
   URL: http://scrutator.me/post/2012/04/04/parallel-world-p1.aspx
   (Дата обращения: 20.04.2021).
2. Потоки [Электронный ресурс].
   URL: https://habr.com/ru/post/279653/
   (Дата обращения: 20.04.2021).