Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»
Дисциплина «Объектно-ориентированное программирование»

Лабораторная работа №8

Тема: Асинхронное программирование

Студент: Инютин М. А.
Группа: М8О-207Б-19
Преподаватель: Чернышев 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:
    1. Вывод информации о фигурах в буфере на экран;
    2. 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 2. Квадрат, прямоугольник, трапеция.

2. Описание программы

Используем классы фигур и *TFactory* из предыдущей лабораторной работы. Создадим отдельный класс *TPubSubMQ*, который будет моделировать очередь сообщений согласно шаблону *«Publishe and Subscribe»*. Так как очередь используется несколькими потоками, то нужно защищать все операции с очередью мьютексом *std::mutex*. Для асинхронной обработки испольуем *std::thread*, который будет обращаться к общей очереди сообщений. Поток обработки будет активно ждать сообщения, а основной поток будет ждать, пока сообщение не будет прочитано. Функции печати фигур на экран и в файл реализованы с помощью лямбда-функций и хранятся в в контейнере *std::vector*. В задании требуется сохранять каждый буфер фигур в уникальный файл, поэтому будем случайно генерировать название файла с помощью функции *genFileName*. Если в конце работы программы буфер имеет размер меньше заданного, то он всё равно обрабатывается потоком.

3. Набор тестов

Программа запускается с аргументом командной строки — размером буффера для асинхронной обработки. Программа обрабатывает строки до конца ввода. На каждой строке располагается тип фигуры и её данные.

Тест 1.
```
3 0 0 10 6 3
1 1 1 4
2 5 5 2 3
3 -1 6 4 8 1
1 -1 -1 2
2 -2 -6 6 2
```

Тест 2.
```
3 2 2 10 4 2
1 -3 -4 5
2 -6 0 4 4
```

Тест 3.
```
3 -4 -2 4 2 2
1 4 4 1
2 -4 4 2 4
2 -4 4 2 4
2 8 9 1 2
```

4. Результат выполнения программы

Программа обрабатывает каждый буфер и выводит на экран данные о фигуре и создаёт уникальный файл с фигурами. Так как фигур может быть довольно много, то программа создаёт файлы в папке. Программа запускалась с размером буфера 2.

Тест 1.
Square {(1, 1), (1, 5), (5, 5), (5, 1)}
Trapeze {(0, 0), (2, 3), (8, 3), (10, 0)}
Trapeze {(-1, 6), (1, 7), (5, 7), (7, 6)}
Rectangle {(5, 5), (5, 7), (8, 7), (8, 5)}
Rectangle {(-2, -6), (-2, 0), (0, 0), (0, -6)}
Square {(-1, -1), (-1, 1), (1, 1), (1, -1)}

3 файла

Тест 2.
Square {(-3, -4), (-3, 1), (2, 1), (2, -4)}
Trapeze {(2, 2), (5, 4), (9, 4), (12, 2)}
Rectangle {(-6, 0), (-6, 4), (-2, 4), (-2, 0)}

2 файла

Тест 3.
Square {(4, 4), (4, 5), (5, 5), (5, 4)}
Trapeze {(-4, -2), (-3, 0), (-1, 0), (0, -2)}
Rectangle {(-4, 4), (-4, 6), (0, 6), (0, 4)}
Rectangle {(-4, 4), (-4, 6), (0, 6), (0, 4)}
Rectangle {(8, 9), (8, 10), (10, 10), (10, 9)}

3 файла

5. Листинг программы

Программа разделена на файлы figure.hpp, square.hpp, rectangle.hpp, trapeze.hpp, factory.hpp, pub_sub.hpp, main.cpp. В каждом файле находится реализация соответствующего класса, а в main.cpp работа с потоком.

figure.hpp

```
#ifndef FIGURE_HPP
#define FIGURE_HPP

#include <iostream>
#include <tuple>

class IFigure {
public:
     virtual void Print() = 0;
     virtual void Write(FILE* out) = 0;
     virtual ~IFigure() {}
};

template<class T1, class T2>
std::ostream & operator << (std::ostream & out, const
std::pair<T1, T2> & p) {
     out << "(" << p.first << ", " << p.second << ")";
     return out;
}

#endif /* FIGURE_HPP */
```

pub_sub.hpp

```
#ifndef PUB_SUB_HPP
#define PUB_SUB_HPP

#include <mutex>
#include <queue>

template<class T>
class TPubSubMQ {
private:
	std::queue<T> MessageQueue;
	std::mutex MQMutex;
public:
	explicit TPubSubMQ() noexcept : MessageQueue(), MQMutex() {}
	~TPubSubMQ() {}

	bool Empty() {
		MQMutex.lock();
		bool res = MessageQueue.empty();
		MQMutex.unlock();
		return res;
	}

	T Front() {
		MQMutex.lock();
		T elem = MessageQueue.front();
		MQMutex.unlock();
		return elem;
	}

	void Pop() {
		MQMutex.lock();
		MessageQueue.pop();
		MQMutex.unlock();
	}

	void Push(const T & message) {
		MQMutex.lock();
		MessageQueue.push(message);
		MQMutex.unlock();
	}
};

#endif /* PUB_SUB_HPP */
```

square.hpp

```
#ifndef SQUARE_HPP
#define SQUARE_HPP

#include "figure.hpp"

const unsigned int SQUARE_TYPE_ID = 1;

template<class T>
class TSquare : public IFigure {
private:
	/* Cords of left bottom corner and square side length */
	std::pair<T, T> Cords;
	T Side;
public:
	TSquare() : Cords(), Side() {}
	TSquare(const std::pair<T, T> & xy, const T & l) : Cords(xy),
Side(l) {}

	void Write(FILE* out) override {
		fwrite(&SQUARE_TYPE_ID, sizeof(unsigned int), 1, out);
		fwrite(&Cords.first, sizeof(T), 1, out);
		fwrite(&Cords.second, sizeof(T), 1, out);
		fwrite(&Side, sizeof(T), 1, out);
	}

	void Print() override {
		std::cout << *this << std::endl;
	}

	template<class U>
	friend std::ostream & operator << (std::ostream & of, const
TSquare<U> & sq) {
		of << "Square {";
		of << std::pair<U, U>(sq.Cords.first, sq.Cords.second) <<
", ";
		of << std::pair<U, U>(sq.Cords.first, sq.Cords.second +
sq.Side) << ", ";
		of << std::pair<U, U>(sq.Cords.first + sq.Side,
sq.Cords.second + sq.Side) << ", ";
		of << std::pair<U, U>(sq.Cords.first + sq.Side,
sq.Cords.second);
		of << "}";
		return of;
	}
};

#endif /* SQUARE_HPP */
```

rectangle.hpp

```
#ifndef RECTANGLE_HPP
#define RECTANGLE_HPP

#include "figure.hpp"

const unsigned int RECTANGLE_TYPE_ID = 2;

template<class T>
class TRectangle : public IFigure {
private:
	/* Cords of left bottom corner, height and width */
	std::pair<T, T> Cords;
	T Height, Width;
public:
	TRectangle() : Cords(), Height(), Width() {}
	TRectangle(const std::pair<T, T> & xy, const T & h, const T &
w) : Cords(xy), Height(h), Width(w) {}

	void Print() override {
		std::cout << *this << std::endl;
	}

	void Write(FILE* out) override {
		fwrite(&RECTANGLE_TYPE_ID, sizeof(unsigned int), 1, out);
		fwrite(&Cords.first, sizeof(T), 1, out);
		fwrite(&Cords.second, sizeof(T), 1, out);
		fwrite(&Height, sizeof(T), 1, out);
		fwrite(&Width, sizeof(T), 1, out);
	}

	template<class U>
	friend std::ostream & operator << (std::ostream & of, const
TRectangle<U> & rect) {
		of << "Rectangle {";
		of << std::pair<U, U>(rect.Cords.first,
rect.Cords.second) << ", ";
		of << std::pair<U, U>(rect.Cords.first, rect.Cords.second
+ rect.Height) << ", ";
		of << std::pair<U, U>(rect.Cords.first + rect.Width,
rect.Cords.second + rect.Height) << ", ";
		of << std::pair<U, U>(rect.Cords.first + rect.Width,
rect.Cords.second);
		of << "}";
		return of;
	}
};

#endif /* RECTANGLE_HPP */
```

factory.hpp

```
#ifndef FACTORY_HPP
#define FACTORY_HPP

#include <memory>

#include "rectangle.hpp"
#include "square.hpp"
#include "trapeze.hpp"

template<class T, class FIGURE>
class TFactory;

template<class T>
class TFactory< T, TSquare<T> > {
public:
     static std::shared_ptr<IFigure> CreateFigure() {
          std::pair<T, T> curCords;
          T curSide;
          std::cout << "Input square as follows: x y a" <<
std::endl;
          std::cout << "x, y is a left bottom corner cords" <<
std::endl;
          std::cout << "a is square side" << std::endl;
          std::cin >> curCords.first >> curCords.second >> curSide;
          TSquare<T>* sq = new TSquare<T>(curCords, curSide);
          return std::shared_ptr<IFigure>(sq);
     }
};

template<class T>
class TFactory< T, TRectangle<T> > {
public:
     static std::shared_ptr<IFigure> CreateFigure() {
          std::pair<T, T> curCords;
          T curHeight, curWidth;
          std::cout << "Input rectangle as follows: x y a b" <<
std::endl;
          std::cout << "x, y is a left bottom corner cords" <<
std::endl;
          std::cout << "a and b are width and heigth" << std::endl;
          std::cin >> curCords.first >> curCords.second >>
curHeight >> curWidth;
          TRectangle<T>* rect = new TRectangle<T>(curCords,
curHeight, curWidth);
          return std::shared_ptr<IFigure>(rect);
     }
};

template<class T>
class TFactory< T, TTrapeze<T> > {
```

```
public:
     static std::shared_ptr<IFigure> CreateFigure() {
          std::pair<T, T> curCords;
          T curGreaterBase, curSmallerBase, curHeight;
          std::cout << "Input trapeze as follows: x y a b c" <<
std::endl;
          std::cout << "x, y is a left bottom corner cords" <<
std::endl;
          std::cout << "a, b and c are larger, smaller base and
height" << std::endl;
          std::cin >> curCords.first >> curCords.second >>
curGreaterBase >> curSmallerBase >> curHeight;
          TTrapeze<T>* trap = new TTrapeze<T>(curCords,
curGreaterBase, curSmallerBase, curHeight);
          return std::shared_ptr<IFigure>(trap);
     }
};

#endif /* FACTORY_HPP */
```

trapeze.hpp

```
#ifndef TRAPEZE_HPP
#define TRAPEZE_HPP

#include "figure.hpp"

const unsigned int TRAPEZE_TYPE_ID = 3;

template<class T>
class TTrapeze : public IFigure {
private:
     /* Cords of left bottom corner, greater and smaller base,
heigth */
     std::pair<T, T> Cords;
     T GreaterBase, SmallerBase, Height;
public:
     TTrapeze() : Cords(), GreaterBase(), SmallerBase(), Height()
{}
     TTrapeze(const std::pair<T, T> & xy, const T & gb, const T &
sb, const T & h) : Cords(xy), GreaterBase(gb), SmallerBase(sb),
Height(h) {
          if (SmallerBase > GreaterBase) {
               std::swap(SmallerBase, GreaterBase);
          }
     }

     void Print() override {
          std::cout << *this << std::endl;
```

```
    }

    void Write(FILE* out) override {
        fwrite(&TRAPEZE_TYPE_ID, sizeof(unsigned int), 1, out);
        fwrite(&Cords.first, sizeof(T), 1, out);
        fwrite(&Cords.second, sizeof(T), 1, out);
        fwrite(&SmallerBase, sizeof(T), 1, out);
        fwrite(&GreaterBase, sizeof(T), 1, out);
        fwrite(&Height, sizeof(T), 1, out);
    }

    template<class U>
    friend std::ostream & operator << (std::ostream & out, const
TTrapeze<U> & trapeze) {
        T d = (trapeze.GreaterBase - trapeze.SmallerBase) / 2.0;
        out << "Trapeze {";
        out << std::pair<T, T>(trapeze.Cords.first,
trapeze.Cords.second) << ", ";
        out << std::pair<T, T>(trapeze.Cords.first + d,
trapeze.Cords.second + trapeze.Height) << ", ";
        out << std::pair<T, T>(trapeze.Cords.first +
trapeze.SmallerBase + d, trapeze.Cords.second + trapeze.Height) <<
", ";
        out << std::pair<T, T>(trapeze.Cords.first +
trapeze.GreaterBase, trapeze.Cords.second);
        out << "}";
        return out;
    }
};

#endif /* TRAPEZE_HPP */
```

main.cpp

```
#include <ctime>
#include <functional>
#include <thread>
#include <vector>

#include "factory.hpp"
#include "pub_sub.hpp"

/*
 * Инютин М А М8О-207Б-19
 * 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:
 *    1. Вывод информации о фигурах в буфере на экран;
 *    2. 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.
 */
```

```
TPubSubMQ< std::vector< std::shared_ptr<IFigure> > > mq;
const std::string FOLDER = "files/";
FILE* file = NULL;

std::string genFileName(size_t n) {
	std::string res;
	for (size_t i = 0; i < n; ++i) {
		res.push_back('a' + std::rand() % 26);
	}
	return res;
}

void ThreadFunc() {
	using functionType =
std::function<void(std::shared_ptr<IFigure> fig)>;
	functionType PrintToStdout = [](std::shared_ptr<IFigure> fig)
{
		fig->Print();
	};
	functionType WriteToFile = [](std::shared_ptr<IFigure> fig) {
		fig->Write(file);
	};
	std::vector<functionType> funcs({PrintToStdout,
WriteToFile});
	bool awake = true;
	while (awake) {
		if (!mq.Empty()) {
			std::vector< std::shared_ptr< IFigure > > message =
mq.Front();
			if (message.empty()) {
				awake = false;
				break;
			}
			file = fopen((FOLDER + genFileName(16)).c_str(),
"wb");
			while (!message.empty()) {
				std::shared_ptr< IFigure > figPtr =
message.back();
				message.pop_back();
				for (auto func : funcs) {
					func(figPtr);
				}
			}
			fclose(file);
			mq.Pop();
		}
	}
}

using SCALAR_TYPE = int;
```

```
int main(int argc, char** argv) {
     if (argc < 2) {
          std::cout << "Missing argument!" << std::endl;
          return -1;
     }
     size_t bufferSize;
     try {
          bufferSize = std::stoi(std::string(argv[1]));
     } catch (std::exception & ex) {
          std::cout << ex.what() << std::endl;
          return -1;
     }
     std::srand(time(NULL));
     std::thread myThread(ThreadFunc);
     std::vector< std::shared_ptr<IFigure> > figures;
     unsigned int type;
     while (std::cin >> type) {
          if (type == SQUARE_TYPE_ID) {
               figures.push_back(TFactory<SCALAR_TYPE,
TSquare<SCALAR_TYPE> >::CreateFigure());
          } else if (type == RECTANGLE_TYPE_ID) {
               figures.push_back(TFactory<SCALAR_TYPE,
TRectangle<SCALAR_TYPE> >::CreateFigure());
          } else if (type == TRAPEZE_TYPE_ID) {
               figures.push_back(TFactory<SCALAR_TYPE,
TTrapeze<SCALAR_TYPE> >::CreateFigure());
          }
          if (figures.size() == bufferSize) {
               mq.Push(figures);
               bool threadWorkDone = false;
               while (!threadWorkDone) {
                    if (mq.Empty()) {
                         threadWorkDone = true;
                    }
               }
               figures.clear();
          }
     }
     mq.Push(figures);
     figures.clear();
     mq.Push(figures);
     myThread.join();
}
```

6. Выводы

В ходе выполнения лабораторной работы я познакомился со стандартными средствами асинхронного программирования на C++, узнал, что STL предоставляет объектно-ориентированную оболочку над системными вызовами POSIX, реализовал просту программу по асинхронной обработке данных. Вряд ли получится загрузить современный процессор однопоточной программой на сто процентов, поэтому большинство приложений используют несколько потоков для ускорения вычислений.

Список литературы

1. Таненбаум Э., Бос Х. *Современные операционные системы.* — 4-е изд. — СПб.: Издательский дом «Питер», 2018. — 1120 с.
   (ISBN 978-5-496-01395-6 «Питер»)
2. std::thread — cppreference.com
   URL: https://en.cppreference.com/w/cpp/thread/thread
   (дата обращения 15.12.2020)
3. std::mutex — cppreference.com
   URL: https://en.cppreference.com/w/cpp/thread/mutex
   (дата обращения 15.12.2020)