**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: Киреев Александр Константинович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Вариант 8. (8-угольник, треугольник, квадрат)

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.

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 выводит данные на экран и запишет в файл.

1. **Описание программы**

Классы фигур сделаны также, как и в прошлых лабораторных работах, за тем лишь исключением, что в них добавился метод печати фигуры в файл. Шаблон Publish-Subscribe реализован при помощи класса EventChannel, который выполняет роль соединяющей шины между двумя потоками. В данную шину мы помещаем объекты структуры Event как только наш буфер готов к обработке. Объекты класса Event содержат поле типа ивента, данные, которые необходимо обработать, файл, в который мы будем записывать результат, а также указатель на функцию-обработчик. Чтобы сделать программу более расширяемой сделаем родительский класс обработчик, от которого и будем наследовать остальные обработчики. Данное решение позволит нам хранить в Event лишь указатель на родительский класс, не заботясь о реализации обработчика. В функции mainLoop файла main.cpp создается новый поток, в котором, параллельно основному потоку, работает поток - обработчик, который ждет, пока в шине сообщений что-либо появится. Для избежания состояния “гонки” будем использовать мьютексы для блокировки множественного доступа к разделяемым данным - шине сообщений. В mainLoop, пока возможно, будем заполнять буфер, а как только он заполнится – отдавать на обработку в другой поток через шину.

1. **Набор тестов**

Тест 1:

1 2

0 0 1 1

1 3

0 0 0 1

1 1

0 0 0 2

3

Тест 1 проверяет работоспособность программы c буфером размера 1. В данном тесте мы получаем 3 файла, в каждом из которых будет записан один вектор из одной фигуры. На экран аналогично выводятся 3 разных вектора.

Тест 2:

1 2

0 0 1 1

1 3

0 0 0 1

1 1

0 0 0 2

3

Тест 2 аналогичен тесту 1 по содержанию, однако для его проверки запустим программу с ключом 2, что сделает вместимость нашего буфера равной 2. В этом случае мы уже получим 2 файла, в первом будет вектор из двух фигур, а во втором файле – вектор из одной фигуры. На экране будет аналогичная ситуация.

1. **Результаты выполнения тестов**

Тест 1:

MacBook-Air-K:Build AK$ ./lr8 1 < ../../test/test01.t

Command -- Description

1 <type> -- Add new figure (1 -- Octagon, 2 -- Square, 3 -- Triangle)

2 -- Display help

3 -- End program

[ (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000), (1.000, -1.000) ]

[ (0.000, 1.000), (-0.866, -0.500), (0.866, -0.500) ]

[ (0.000, 2.000), (-1.414, 1.414), (-2.000, 0.000), (-1.414, -1.414), (-0.000, -2.000), (1.414, -1.414), (2.000, -0.000), (1.414, 1.414) ]

MacBook-Air-K:Build AK$ ls

CMakeCache.txt Makefile file\_1 file\_3

CMakeFiles cmake\_install.cmake file\_2 lr8

Тест 2:

MacBook-Air-K:Build AK$ ./lr8 2 < ../../test/test02.t

Command -- Description

1 <type> -- Add new figure (1 -- Octagon, 2 -- Square, 3 -- Triangle)

2 -- Display help

3 -- End program

[ (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000), (1.000, -1.000) ]

[ (0.000, 1.000), (-0.866, -0.500), (0.866, -0.500) ]

[ (0.000, 2.000), (-1.414, 1.414), (-2.000, 0.000), (-1.414, -1.414), (-0.000, -2.000), (1.414, -1.414), (2.000, -0.000), (1.414, 1.414) ]

MacBook-Air-K:Build AK$ ls

CMakeCache.txt Makefile file\_1

CMakeFiles cmake\_install.cmake file\_2 lr8

1. **Листинг программы**

**main.cpp**

// Киреев Александр Константинович 206

#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <memory>

#include "pubsub.hpp"

// мануал

void help() {

std::cout << "Command -- Description\n" <<

"1 <type> -- Add new figure (1 -- Octagon, 2 -- Square, 3 -- Triangle)\n" <<

"2 -- Display help\n" <<

"3 -- End program" << std::endl;

}

void mainLoop(int max\_cap) {

std::vector<std::shared\_ptr<Figure>> buf;

int command = 0, type = 0;

std::string s;

std::string filename;

EventChannel channel;

Event event;

int curr\_file\_count = 1;

std::thread routine(HandleLoop, std::ref(channel));

while (std::cin >> s) {

if (s.length() > 1) {

std::cout << "Invalid command." << std::endl;

continue;

}

std::stringstream ss(s);

ss >> command;

if (command == 3) {

if (buf.size() != 0) {

channel.push({EventCode::file,

buf,

"file\_" + std::to\_string(curr\_file\_count),

std::shared\_ptr<Handler>(new PrinterInFile)});

channel.push({EventCode::screen,

buf,

"",

std::shared\_ptr<Handler>(new PrinterOnScreen)});

curr\_file\_count++;

buf.clear();

}

channel.push({EventCode::quit,

buf,

"",

NULL});

break;

}

if (command == 1) {

if (buf.size() < max\_cap) {

std::cin >> type;

if (type == 1) {

buf.push\_back(std::shared\_ptr<Figure>(new Octagon(std::cin)));

} else if (type == 2) {

buf.push\_back(std::shared\_ptr<Figure>(new Square(std::cin)));

} else if (type == 3) {

buf.push\_back(std::shared\_ptr<Figure>(new Triangle(std::cin)));

} else {

std::cout << "Invalid type." << std::endl;

}

}

if (buf.size() == max\_cap) {

channel.push({EventCode::file,

buf,

"file\_" + std::to\_string(curr\_file\_count),

std::shared\_ptr<Handler>(new PrinterInFile)});

channel.push({EventCode::screen,

buf,

"",

std::shared\_ptr<Handler>(new PrinterOnScreen)});

curr\_file\_count++;

buf.clear();

}

} else if (command == 2) {

help();

} else {

std::cout << "Invalid command!" << std::endl;

}

}

routine.join();

}

int main(int argc, char\* argv[]) {

if (argc != 2) {

std::cout << "Usage: ./lr8 <buffer\_capacity>\n";

return 0;

}

int max\_cap = std::atoi(argv[1]);

help();

mainLoop(max\_cap);

return 0;

}

**pubsub.cpp**

// М8О-206Б-19 Киреев А.К.

#include "pubsub.hpp"

void PrinterOnScreen::handle(const std::vector<std::shared\_ptr<Figure>> fv, std::string filename) {

for (auto& f: fv) {

f->print(std::cout);

std::cout << "\n";

}

}

void PrinterInFile::handle(const std::vector<std::shared\_ptr<Figure>> fv, std::string filename) {

std::ofstream ofs;

ofs.open(filename);

for (auto& f: fv) {

f->fprint(ofs);

ofs << "\n";

}

ofs.close();

}

void EventChannel::push(Event e) {

std::lock\_guard<std::mutex> lock(mx);

buf.push(e);

condv.notify\_one();

}

Event EventChannel::front() {

std::lock\_guard<std::mutex> lock(mx);

return buf.front();

}

void EventChannel::pop() {

std::lock\_guard<std::mutex> lock(mx);

buf.pop();

}

bool EventChannel::is\_empty() {

std::lock\_guard<std::mutex> lock(mx);

return buf.empty();

}

void HandleLoop(EventChannel& channel) {

bool is\_over = false;

while(!is\_over) {

std::unique\_lock<std::mutex> lock(channel.condmx);

channel.condv.wait(lock, [&channel]() { return !channel.is\_empty(); });

Event next\_event = channel.front();

channel.pop();

switch (next\_event.code) {

case EventCode::screen:

next\_event.handler->handle(next\_event.data, "");

break;

case EventCode::file:

next\_event.handler->handle(next\_event.data, next\_event.filename);

break;

case EventCode::quit:

is\_over = true;

break;

default:

break;

}

}

}

**pubsub.hpp**

#include <vector>

#include <queue>

#include <memory>

#include <iostream>

#include <fstream>

#include <string>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include "triangle.hpp"

#include "square.hpp"

#include "octagon.hpp"

class Handler {

public:

virtual void handle(const std::vector<std::shared\_ptr<Figure>> fv, std::string filename) = 0;

virtual ~Handler() = default;

};

class PrinterOnScreen: public Handler {

public:

void handle(const std::vector<std::shared\_ptr<Figure>> fv, std::string filename) override;

};

class PrinterInFile: public Handler {

public:

void handle(const std::vector<std::shared\_ptr<Figure>> fv, std::string filename) override;

};

enum class EventCode {

screen = 0,

file,

quit

};

struct Event {

EventCode code;

std::vector<std::shared\_ptr<Figure>> data;

std::string filename;

std::shared\_ptr<Handler> handler;

};

class EventChannel {

private:

std::queue<Event> buf;

std::mutex mx;

public:

std::mutex condmx;

std::condition\_variable condv;

EventChannel() = default;

~EventChannel() = default;

void push(Event e);

Event front();

void pop();

bool is\_empty();

};

void HandleLoop(EventChannel& channel);

1. **Вывод**

Многопоточное и асинхронное программирование – быстро развивающееся направление в программировании, которое может ускорить работу серверов и сервисов, имеющих многоядерные процессоры, способные вычислять выражения параллельно. Для работы с асинхронными моделями в C++ представлен большой арсенал функций и объектов: мьютексы, функции ожидания и создания потоков, condition variables и прочее.

**Список литературы**

1. “Язык программирования C++. Краткий курс”, Бьярне Страуструп, 2-е издание, (перевод Игоря Красикова), бумага офсетная-белая, твердый переплет, 320 стр., ISBN 978-5-907144-12-5.