**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: Шараковский Юрий Дмитриевич

Группа: 80-206

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.
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;

* 1. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.

1. Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.
2. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.
3. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков.
4. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.
5. 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.

Вариант 25: треугольник, квадрат, прямоугольник.

1. Описание программы

Программа реализует многопоточную систему обработки ввода и связана с проблемой producer-consumer (в моей программе один поток — производитель и один поток — потребитель). Используется паттерн проектирования Publish-Subscribe. Основной поток в функции main считывает ввод и добавляет фигуры в буфер, при полном заполнении которого все подписчики уведомляются о произошедшем событии посредством установки специального поля класса подписчика в значение логической истины и вызова метода ::notify\_one условной переменной. В цикле потока-обработчика происходит ожидание этой условной переменной с помощью метода ::wait, далее для каждого готового подписчика вызывается его функция-обработчик буфера (лямбда).

1. Набор тестов

Описание команд:

1. create triangle <center> <vertex-0>
2. create square <center> <vertex-0>
3. create rectangle <center> <vertex-0>

test01.txt — проверяет возможность добавить элементы, работу обработчиков, отсутствие проблем с завершением программы (запускать с ключем 2, результат в файле 0).

1 1 1 1 1

2 2 2 2 2

test02.txt — дополнительная проверка на отсутствие взаимоблокировок, а также на отсутствие вывода неполного буфера в файл (запускать с ключем 3, вывод только в файле 0).

1 1 1 1 1

2 2 2 2 2

3 3 3 3 3

1 0 0 -1 -1

2 -1 2 3 4

1. Результаты выполнения тестов

$ cat answer01:

1. triangle

2. square

3. rectangle

WAITING

triangle: [x: 1 y: 1] [x: 1 y: 1] [x: 1 y: 1]

square: [x: 2 y: 2] [x: 2 y: 2]

Goodbye!

$ cat 0

triangle: [x: 1 y: 1] [x: 1 y: 1] [x: 1 y: 1]

square: [x: 2 y: 2] [x: 2 y: 2]

$ cat answer02:

1. triangle

2. square

3. rectangle

WAITING

triangle: [x: 1 y: 1] [x: 1 y: 1] [x: 1 y: 1]

square: [x: 2 y: 2] [x: 2 y: 2]

rectangle: [x: 3 y: 3] [x: 3 y: 3]

WAITING

Goodbye!

$ cat 0

triangle: [x: 1 y: 1] [x: 1 y: 1] [x: 1 y: 1]

square: [x: 2 y: 2] [x: 2 y: 2]

rectangle: [x: 3 y: 3] [x: 3 y: 3]

1. Листинг программы

main.cpp

// Шараковский Юрий Дмитриевич. М8О-206Б-19, МАИ. 12.2020.

// Лабораторная работа №7.

#include <fstream>

#include <iostream>

#include <iomanip>

#include <cstdio>

#include <cstdlib>

#include <string>

#include <vector>

#include <list>

#include <array>

#include <mutex>

#include <condition\_variable>

#include <thread>

#include <functional>

#include <algorithm>

#include "figure.hpp"

using FigureBuffer = std::list<std::shared\_ptr<Figure>>;

class Subscriber {

public:

bool isActive = true;

bool isAwaken = false;

void operator()(FigureBuffer figures) {

this->Handler(figures);

}

std::function<void(FigureBuffer)> Handler;

};

class Publisher {

public:

Publisher(uint32\_t max\_size) {

MAX\_SIZE = max\_size;

}

~Publisher() {

Release();

}

Publisher(const Publisher& other) = delete;

Publisher& operator=(const Publisher& other) = delete;

Publisher(Publisher&& other) = delete;

Publisher& operator=(Publisher&& other) = delete;

void Subscribe(Subscriber& sub) {

this->subscribers.emplace\_back(&sub);

}

void Release() {

std::unique\_lock<std::mutex> lock(mx\_release);

for (auto& sub : subscribers) {

sub->isActive = false;

}

//this->subscribers.clear();

this->End = true;

cv\_process.notify\_one();

}

void Append(Figure\* f) {

std::unique\_lock<std::mutex> lock(mx\_buffer);

cv\_append.wait(lock, [&]() { return buffer.size() < MAX\_SIZE; });

buffer.emplace\_back(f);

if (buffer.size() == MAX\_SIZE) {

for (auto& sub : subscribers) {

sub->isAwaken = true;

}

cv\_process.notify\_one();

}

}

void Clear() {

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

buffer.clear();

cv\_append.notify\_all();

}

void Wait() {

std::unique\_lock<std::mutex> lock(mx\_release);

cv\_process.wait(lock, [&]() { return buffer.size() == MAX\_SIZE || this->End; });

}

FigureBuffer CopyBuffer() {

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

return buffer;

}

private:

uint32\_t MAX\_SIZE;

FigureBuffer buffer;

std::condition\_variable cv\_append;

std::condition\_variable cv\_process;

std::mutex mx\_buffer;

std::mutex mx\_release;

bool End = false;

std::vector<Subscriber\*> subscribers;

};

void doProcessing(Publisher& pub, std::array<Subscriber, 2>& sub) {

bool alive;

bool doneWork;

do {

alive = false;

doneWork = false;

//std::cout << "\tWAITING" << std::endl;

pub.Wait();

for (auto& subscr : sub) {

if (subscr.isAwaken) {

doneWork = true;

subscr.isAwaken = false;

subscr(pub.CopyBuffer());

}

}

if (doneWork) {

pub.Clear();

}

alive = std::any\_of(sub.begin(), sub.end(), [](Subscriber& elem) { return elem.isActive; });

} while (alive);

}

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

if (argc < 2) {

std::terminate();

}

Publisher pub(std::atoi(argv[1]));

std::array<Subscriber, 2> sub;

uint32\_t counter = 0;

sub.at(0).Handler = [](FigureBuffer buffer) {

for (auto& elem : buffer) {

std::cout << \*(elem.get()) << std::endl;

}

};

pub.Subscribe(sub.at(0));

sub.at(1).Handler = [&counter](FigureBuffer buffer) {

std::fstream file;

file.open(std::to\_string(counter).c\_str(), std::fstream::out);

for (auto& elem : buffer) {

file << \*(elem.get()) << std::endl;

}

file.close();

++counter;

};

pub.Subscribe(sub.at(1));

std::cout << "1. triangle\n2. square\n3. rectangle" << std::endl;

std::thread consumers\_thread(doProcessing, std::ref(pub), std::ref(sub));

uint32\_t option;

while (std::cin >> option) {

Figure\* fig;

if (option == 1) {

fig = new Triangle();

}

else if (option == 2) {

fig = new Square();

}

else if (option == 3) {

fig = new Rectangle();

}

else {

throw std::runtime\_error("Wrong figure type!");

}

std::cin >> \*fig;

pub.Append(fig);

}

pub.Release();

consumers\_thread.join();

std::cout << "Goodbye!" << std::endl;

return 0;

}

Файлы figure.hpp и figure.cpp соответствуют таким же файлам из ЛР3.

1. Вывод

Умение эффективно использовать средства синхронизации и многопоточного программирования является важным качеством хорошего программиста, так как в нынешнее время мы видим тренд на параллелизацию вычислений и облачные вычисления. В стандартной библиотеки C++ много способов организовать многопоточное программирование: std::thread, std::jthread, std::async, std::promise, и.т.д.

Список литературы

1. C++ Primer (5th Edition) Stanley B. Lippman, Josee Lajoie, E. Moo. – Addison-Wesley, 2019. ISBN-13: 978-0321714114, ISBN-10: 9780321714114.

2. Effective Modern C++, Scott Meyers. ISBN-13: 978-1491903995

ISBN-10: 9781491903995