Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

III семестр

Задание 8: «Асинхронное программирование»

|  |  |
| --- | --- |
| Группа: | М8О-206Б-18, №14 |
| Студент: | Орозбакиев Э.Д. |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: |  |

Москва, 2019

1. **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.

1. **Адрес репозитория на GitHub**

https://github.com/p0kemo4ik/oop\_exercise\_08

1. **Код программы на С++**

main.cpp

#include **<iostream>**  
#include **<memory>**  
#include **<vector>**  
#include **<thread>**  
  
#include **"factory.hpp"**  
#include **"figure.hpp"**  
#include **"subscriber.hpp"**  
  
**void** help() {  
 std::cout << **"help - print this menu\n"**  
 **"add <square, rectangle or trapezoid> <vertices> - add a figure\n"**  
 **"quit\n"**;  
}  
  
**int** main(**int** argc,**char**\* argv[]) {  
 **if** (argc != 2) {  
 std::cout << **"2 arguments needed\n"**;  
 **return** 1;  
 }  
  
 **int** buffer\_size = std::stoi(argv[1]);  
 std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer = std::make\_shared<std::vector<std::shared\_ptr<figure>>>();  
 buffer->reserve(buffer\_size);  
 factory factory;  
 std::string cmd;  
 subscriber sub;  
 sub.processors.push\_back(std::make\_shared<stream\_processor>());  
 sub.processors.push\_back(std::make\_shared<file\_processor>());  
 std::thread sub\_thread(std::ref(sub));  
  
 **while** (**true**) {  
 std::unique\_lock<std::mutex> locker(sub.mtx);  
 std::cin >> cmd;  
 **if** (cmd == **"help"**) {  
 help();  
 } **else if** (cmd == **"add"**) {  
 **try** {  
 buffer->push\_back(factory.FigureCreate(std::cin));  
 } **catch** (std::logic\_error &e) {  
 std::cout << e.what() << **'\n'**;  
 **continue**;  
 }  
 **if** (buffer->size() == buffer\_size) {  
 std::cout << **"You've reached the limit\n"**;  
 sub.buffer = buffer;  
 sub.cond\_var.notify\_all();  
 sub.cond\_var.wait(locker, [&](){ **return** sub.buffer == **nullptr**;});  
 buffer->clear();  
 }  
 } **else if** (cmd == **"quit"**) {  
 **break**;  
 } **else** {  
 std::cout << **"Wrong command\n"**;  
 }  
 }./  
 sub.stop = **true**;  
 sub.cond\_var.notify\_all();  
 sub\_thread.join();  
 **return** 0;  
}

*processor.cpp*

#include **"processor.hpp"****void** stream\_processor::process(std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer) { **for** (**auto** figure : \*buffer) { figure->print(std::cout); }}**void** file\_processor::process(std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer) { std::ofstream fout; fout.open(std::to\_string(cnt) + **".txt"**); cnt++; **if** (!fout.is\_open()) { std::cout << **"can't open\n"**; **return**; } **for** (**auto** figure : \*buffer) { figure->print(fout); }}

*subscriber.cpp*

#include **"subscriber.hpp"****void** subscriber::**operator**()() { **for**(;;) { std::unique\_lock<std::mutex>lock(mtx); cond\_var.wait(lock,[&]{ **return** (buffer != **nullptr** || stop);}); **if** (stop) { **break**; } **for** (**auto** elem: processors) { elem->process(buffer); } buffer = **nullptr**; cond\_var.notify\_all(); }}

1. **Результаты выполнения тестов**

*test\_01.txt*

add pentagon 0 0 0 0 0 0 0 0 0 0  
add hexagon 1 1 1 1 1 1 1 1 1 1 1 1  
add octagon 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
quit

*test\_result\_01.txt*

You've reached the limit  
0 0 0 0 0 0 0 0 0 0   
Pentagon  
Center: 0 0 Area: 0  
  
1 1 1 1 1 1 1 1 1 1 1 1   
Hexagon  
Center:1 1 Area: 0  
  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2   
Octagon  
Center: 2 2 Area: 0

1. **Объяснение результатов работы программы**

При запуске программы пользователь задаёт размер буфера, в который помещаются задаваемые им фигуры. Когда буфер становится полным, в терминал выводится вся информация о фигурах, а буфер очищается.

1. **Вывод**

При разработке программ очень редко задействуется один поток или процесс, так как очень выгодно содержать большое кол-во синхронизированных потоков, на которых происходят вычисления. Для этого в стандартных библиотеках языка С++ присутствуют специальные классы потоков, критических переменных, мьютексов и т.д. для реализации потоков и их синхронизации. Каждый программист должен уметь пользоваться ими и писать многопоточные программы.