НИУ “ВШЭ”

ОТЧЕТ

Домашнее задание 4

Вариант 12

Выполнил студент

группы БПИ195

Кан Алексей

Москва, 2020

Задание

Задача о больнице. В больнице два врача принимают пациентов, выслушивают их жалобы и отправляют их или к стоматологу или к хирургу или к терапевту. Стоматолог, хирург и терапевт лечат пациента. Каждый врач может принять только одного пациента за раз. Пациенты стоят в очереди к врачам и никогда их не покидают. Создать многопоточное приложение, моделирующее рабочий день клиники.

О программе

Работа выполнялась на языке C++. Для разработки использовалась среда разработки Visual Studio 2019.

Входные данные

Пользователь ввод количество пациентов. Предусмотрен повторный ввод при неверных входных данных.

Выходные данные

В течение определенного времени выводится информация о работе больницы. Для вывода реализован безопасный (синхронизированный) вывод.

Алгоритм

Потоки реализованы при помощи стандартной библиотеки C++.

Потоки пациентов запускаются в цикле (количество вводит пользователь).

Врач принимает одного пациента, а остальные ждут. Когда доктор заканчивает работу с пациентом, он принимает следующего.

Работа потоков синхронизирована при помощи mutex.

Комментарии

В коде программы также присутствуют комментарии, поясняющие все действия (на английском языке)

Код программы

```
#include <iostream>
#include <vector>
#include <string>
#include <chrono>
#include <mutex>
#include <sstream>
#include <thread>
#include <random>
#include <condition_variable>

using namespace std;

// End of doctors job.
bool finish = false;
// Flags.
bool ready = false, exch1 = false, exch2 = false, answer = false;
// Mutexes for synch.
mutex m_ready, m_exch1, m_exch2, m_answer;
// Safe output mutexes.
mutex m_write;
condition_variable cv_ready, cv_exch1, cv_exch2, cv_answer;

// Safe output.
void write(stringstream& message) {
    m_write.lock();
    cout << message.str();
```

```
    message.str("");
    m_write.unlock();
}

// Get doctor in string.
string getDoctor(int index) {
    switch (index) {
    case 1:
        return "Dentist";
    case 2:
        return "Surgeon";
    case 3:
        return "Therapist";
    default:
        return "";
    }
}

// Input with repeat.
int getCountOfPatients() {
    // While correct input.
    while (true) {
        cout << "Enter count of patients: ";
        int a;
        cin >> a;

        // Check input.
        if (cin.fail() || a <= 0) { // if incorrect
            cin.clear(); // clear
            cin.ignore(32767, '\n');
            cout << "Oops, that input is invalid.  Please try again.\n";
        }
        else {
```

```
            cin.ignore(32767, '\n'); // remove trash
            return a;
        }
    }
}

// Patient info struct.
struct {
    int id; // id of patient
    int doctor; // doctor's type
}info;

// Sleep thread for 'ms'.
void sleep(int ms) {
    this_thread::sleep_for(chrono::milliseconds(ms));
}

void doctor() {
    unique_lock<mutex> ul_exch1(m_exch1), ul_exch2(m_exch2);

    stringstream ss;

    while (!finish) {
        // Doctor nut busy.
        ready = true;
        cv_ready.notify_one();

        // Wait for patient.
        while (!exch1) { cv_exch1.wait(ul_exch1); }

        exch1 = false;
```

```
        ss << getDoctor(info.doctor) << " started to heal patient " <<
info.id << "." << endl;
        write(ss);

        // Simulation of healing.
        sleep(1000 + rand() % 10000);

        ss << getDoctor(info.doctor) << " finished healing patient " <<
info.id << "." << endl;
        write(ss);

        // Patient cured.
        answer = true;
        cv_answer.notify_one();

        // Wait patient to leave.
        while (!exch2) { cv_exch2.wait(ul_exch2); }

        exch2 = false;
        sleep(150);
    }
}

void patient(int index, int doctor) {
    unique_lock<mutex> ul_ready(m_ready), ul_answer(m_answer);

    srand(time(nullptr));

    stringstream ss;

    sleep(1000 + rand() % 10000);

    ss << "Patient " << index << " going to the " << getDoctor(doctor)
<< "." << endl;
```

```
    write(ss);

    sleep(1000 + rand() % 10000);

    // Wait for doctor.
    while (!ready) { cv_ready.wait(ul_ready); }
    ready = false;

    ss << "Patient " << index << " came to the " << getDoctor(doctor) <<
"." << endl;
    write(ss);

    // Doctor get info about patient.
    info.id = index;
    info.doctor = doctor;

    exch1 = true;
    cv_exch1.notify_one();

    // Healing.
    while (!answer) {
        cv_answer.wait(ul_answer);
    }

    answer = false;
    ss << "Patient " << info.id << " cured." << endl;
    write(ss);

    // Patient leaving.
    exch2 = true;
    cv_exch2.notify_one();
    ss << "Patient " << index << " left hospital." << endl;
    write(ss);
```

```
}

int main() {
    int count = getCountOfPatients();

    // Doctor's thread.
    thread doctor(doctor);

    // Patients threads.
    vector<thread> patients;

    // Run <count> threads of patients.
    for (int i = 0; i < count; i++) {
        cout << "Patient " << i + 1 << " came to the hospital. " <<
endl;
        patients.push_back(thread(patient, i + 1, rand() % 3));
    }

    // Wait for all threads.
    for (int i = 0; i < count; i++) {
        patients.at(i).join();
    }
    doctor.join();

    // The end.
    return 0;
}
```