Национальный исследовательский университет
"Высшая школа экономики"
Факультет компьютерных наук
Программная инженерия

Микропроект 2

Вариант 1

Задача о парикмахере

Агроскин Александр Викторович

БПИ 199

Постановка задания

Задача о парикмахере: В тихом городке есть парикмахерская. Салон парикмахерской мал, ходить там может только парикмахер и один посетитель. Парикмахер всю жизнь обслуживает посетителей. Когда в салоне никого нет, он спит в кресле. Когда посетитель приходит и видит спящего парикмахера, он будет его, садится в кресло и спит, пока парикмахер занят стрижкой. Если посетитель приходит, а парикмахер занят, то он встает в очередь и засыпает. После стрижки парикмахер сам провожает посетителя. Если есть ожидающие посетители, то парикмахер будит одного из них и ждет пока тот сядет в кресло парикмахера и начинает стрижку. Если никого нет, он снова садится в свое кресло и засыпает до прихода посетителя. Создать многопоточное приложение, моделирующее рабочий день парикмахерской.

В моей немного вольной интерпретации посетители приходят не стричься, а бриться.

Ограничение на входные и выходные данные

При запуске программа запрашивает у пользователя одно число – количество посетителей парикмахерской.

Выходные данные представляют собой описание событий в парикмахерской на английском языке.

Пример вывода:

```
Please enter the number of clients: 3
A new day begins. The barber is fast asleep, for now
Client George walks in
The barber wakes up!
The barber serves client George. This will take 3 seconds
Client Roger walks in
Finished shaving client George
The barber serves client Roger. This will take 2 seconds
Client Thomas walks in
Finished shaving client Roger
The barber serves client Thomas. This will take 2 seconds
Finished shaving client Thomas
The barber goes back to sleep
The barber has finished for the day
```

Логика работы программы

Логика работы программы основана на двух типах потоков: поток парикмахера (**barber_**) и поток клиента (**Client::thread**). Объект-клиент является совокупностью имени клиента, потока клиента и мьютекса, с помощью которого клиент “засыпает” и “просыпается”. Объекты-клиенты хранятся в очереди **queue_** в порядке появления. Парикмахер обслуживает посетителей, по очереди запуская потоки, лежащие в **queue_**, открывая мьютекс соответствующего пользователя. После того, как поток клиента завершил свою работу, поток парикмахера запускает следующего клиента из очереди, или, если очередь пуста, засыпает и ждет появления новых клиентов. После того, как было обслужено максимальное число посетителей, поток парикмахера завершает свою работу.

Реализация логики программы

Посетитель парикмахерской представлен структурой **Client**, которая содержит три поля: имя клиента, поток с функцией логики клиента (сама функция является методом структуры) и мьютексом, с помощью которого парикмахер запускает поток клиента. Для того, чтобы поток клиента не запустился раньше времени, мьютекс создается закрытым.

Структура клиента:

```
struct Client {
    explicit Client(std::string name) : name(std::move(name)) {
        mutex.lock();
        thread = std::thread(&Client::ClientFunc, this);
    }

    std::string name;
    std::thread thread;
    std::mutex mutex;

    void ClientFunc() {
        mutex.lock();
        int barber_time = rand() % 3 + 1;
        std::cout << "The barber serves client " << name << ". This will take " << barber_time
                  << " seconds\n";
        std::this_thread::sleep_for(std::chrono::seconds(barber_time));
        std::cout << "Finished shaving client " << name << "\n";
    }
};
```

Метод **ClientFunc** отвечает за процесс обслуживания клиента. Обслуживание длится случайное количество секунд от 1 до 3.

Логика парикмахерской реализована в классе **BarberShop**. В конструкторе класса инициализируется максимальное количество клиентов (передается параметром конструктора) и запускается главный поток парикмахера.

Главный поток парикмахера исполняет функцию **Barber**, которая по очереди в цикле обслуживает клиентов. При каждой итерации цикла программа закрывает мьютекс на запись в очередь, проверяет, ждут ли в очереди клиенты (с помощью std::atomic<bool> флага **client_waits_**), и если ждут, то запускает следующего в очереди клиента, открывая его мьютекс. После того, как клиент завершил свою работу, парикмахер снова проверяет наличие клиентов в очереди и засыпает при их отсутствии.

Конструктор класса и функция **Barber:**

```
explicit BarberShop(int max_clients)
    : max_clients_(max_clients), barber_(std::thread(&BarberShop::Barber, this)) {
    std::lock_guard<std::mutex> lock( &queue_mutex_);
    std::cout << "A new day begins. The barber is fast asleep, for now\n";
}

void Barber() {
    while (max_clients_ > 0) {
        std::lock_guard<std::mutex> lock( &queue_mutex_);
        if (client_waits_ && !queue_.empty()) {
            queue_.front().mutex.unlock();
            queue_.front().thread.join();
            queue_.pop();
            if (queue_.empty() && !client_in_queue_) {
                std::cout << "The barber goes back to sleep\n";
                client_waits_ = false;
            }
            max_clients_--;
        }
    }
}
```

Добавление нового клиента осуществляется с помощью функции **ClientWalksIn**. При вызове функции как параметр передается имя нового посетителя. Функция выводит информацию о том, что в парикмахерскую зашел новый пользователь (используется stringstream для атомарности вывода). После этого функция закрывает мьютекс на доступ к очереди посетителей, создает новый объект **Client** и добавляет его в очередь. Также функция обновляет значение **client_waits_** и “будит” парикмахера если требуется.

Функция **ClientWalksIn**:

```
void ClientWalksIn(const std::string& name) {
    std::stringstream stream;
    client_in_queue_ = true;
    stream << "Client " << name << " walks in\n";
    std::cout << stream.str();
    std::lock_guard<std::mutex> lock( &: queue_mutex_);
    queue_.emplace(name);
    client_in_queue_ = false;
    if (!client_waits_) {
        std::cout << "The barber wakes up!\n";
        client_waits_ = true;
    }
}
```

Также в классе **BarberShop** реализована функция **WaitForEvening**, являющаяся оберткой над методом **join** потока **barber_**. Функция присоединяет поток парикмахера, дожидаясь окончания его работы, и, если в очереди остались клиенты (в данной реализации такой ситуации не возникает), очищает очередь **queue_**.

Функция **WaitForEvening**:

```
void WaitForEvening() {
    barber_.join();
    std::cout << "The barber has finished for the day\n";
    std::lock_guard<std::mutex> lock( &: queue_mutex_);
    if (!queue_.empty()) {
        std::cout << "The rest of the clients go home\n";
        while (!queue_.empty()) {
            queue_.pop();
        }
    }
}
```

Во входной точке программы запрашивается число клиентов, создается объект парикмахерской и, со случайным интервалом до 199 миллисекунд, посетители со случайными именами входят в парикмахерскую.

Тестирование программы

Результат работы программы для 10 посетителей:

```
Please enter the number of clients: 10
A new day begins. The barber is fast asleep, for now
Client Tim walks in
The barber wakes up!
The barber serves client Tim. This will take 1 seconds
Client Larry walks in
Finished shaving client Tim
The barber serves client Larry. This will take 2 seconds
Client Matthew walks in
Finished shaving client Larry
The barber serves client Matthew. This will take 1 seconds
Client Fred walks in
Finished shaving client Matthew
The barber serves client Fred. This will take 3 seconds
Client Otto walks in
Finished shaving client Fred
The barber serves client Otto. This will take 1 seconds
Client Roger walks in
Finished shaving client Otto
The barber serves client Roger. This will take 2 seconds
Client Paul walks in
Finished shaving client Roger
The barber serves client Paul. This will take 3 seconds
Client Hal walks in
Finished shaving client Paul
The barber serves client Hal. This will take 2 seconds
Client Tim walks in
Finished shaving client Hal
The barber serves client Tim. This will take 2 seconds
Client Thomas walks in
Finished shaving client Tim
The barber serves client Thomas. This will take 2 seconds
Finished shaving client Thomas
The barber goes back to sleep
The barber has finished for the day
```

Приложение 1

Список используемых источников

1. https://www.justsoftwaresolutions.co.uk/threading/implementing-a-thread-safe-queue-using-condition-variables.html
2. https://nrecursions.blogspot.com/2014/08/mutex-tutorial-and-example.html

Приложение 2

Код программы

```
#include <iostream>
#include <queue>
#include <mutex>
#include <thread>
#include <utility>
#include <sstream>

struct Client {
   explicit Client(std::string name) : name(std::move(name)) {
       mutex.lock();
       thread = std::thread(&Client::ClientFunc, this);
   }

   std::string name;
   std::thread thread;
   std::mutex mutex;

   void ClientFunc() {
       mutex.lock();
       int barber_time = rand() % 3 + 1;
       std::cout << "The barber serves client " << name << ". This will take " <<
barber_time
                 << " seconds\n";
       std::this_thread::sleep_for(std::chrono::seconds(barber_time));
       std::cout << "Finished shaving client " << name << "\n";
   }
};

class BarberShop {
public:
   explicit BarberShop(int max_clients)
       : max_clients_(max_clients), barber_(std::thread(&BarberShop::Barber, this))
{
       std::lock_guard<std::mutex> lock(queue_mutex_);
       std::cout << "A new day begins. The barber is fast asleep, for now\n";
   }

   void Barber() {
```

```
        while (max_clients_ > 0) {
            std::lock_guard<std::mutex> lock(queue_mutex_);
            if (client_waits_ && !queue_.empty()) {
                queue_.front().mutex.unlock();
                queue_.front().thread.join();
                queue_.pop();
                if (queue_.empty() && !client_in_queue_) {
                    std::cout << "The barber goes back to sleep\n";
                    client_waits_ = false;
                }
                max_clients_--;
            }
        }
    }

    void ClientWalksIn(const std::string& name) {
        std::stringstream stream;
        client_in_queue_ = true;
        stream << "Client " << name << " walks in\n";
        std::cout << stream.str();
        std::lock_guard<std::mutex> lock(queue_mutex_);
        queue_.emplace(name);
        client_in_queue_ = false;
        if (!client_waits_) {
            std::cout << "The barber wakes up!\n";
            client_waits_ = true;
        }
    }

    void WaitForEvening() {
        barber_.join();
        std::cout << "The barber has finished for the day\n";
        std::lock_guard<std::mutex> lock(queue_mutex_);
        if (!queue_.empty()) {
            std::cout << "The rest of the clients go home\n";
            while (!queue_.empty()) {
                queue_.pop();
            }
        }
    }

private:
    std::queue<Client> queue_;
    std::mutex queue_mutex_;
    std::atomic<int> max_clients_;
    std::thread barber_;
    std::atomic<bool> client_waits_ = false, client_in_queue_ = false;
};

int main() {
    std::cout << "Please enter the number of clients: ";
    int client_num = -1;
    std::cin >> client_num;
    srand(time(nullptr));
    std::vector<std::string> names{
        "Adam",  "Alex",  "Aaron",  "Ben",     "Carl", "Dan",    "David",  "Edward",
        "Fred",  "Frank", "George", "Hal",     "Hank", "Ike",    "John",   "Jack",
```

```
        "Joe",   "Larry", "Monte",  "Matthew", "Mark", "Nathan", "Otto",   "Paul",
        "Peter", "Roger", "Steve",  "Thomas",  "Tim",  "Ty",     "Victor",
"Walter"};
    BarberShop shop(client_num);
    for (int i = 0; i < client_num; ++i) {
        shop.ClientWalksIn(names[rand() % names.size()]);
        std::this_thread::sleep_for(std::chrono::milliseconds(rand() % 200));
    }
    shop.WaitForEvening();
    return 0;
}
```