**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |
| --- | --- |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ОТЧЁТ ПО МИКРОПРОЕКТУ № 2**  **ПО ДИСЦИПЛИНЕ**  **«АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ»**  **ВАРИАНТ 15** | | |
|  |  | |
| **Исполнитель**  студент группы БПИ191  Н. К. Игумнов | |
|  | | |
|  | |  |

**Москва 2020**

ЗАДАНИЕ

15. *Задача о гостинице - 3 (дамы и джентльмены).* В гостинице 10 номеров рассчитаны на одного человека и 15 номеров рассчитаны на двух человек. В гостиницу приходят клиенты дамы и клиенты джентльмены, и конечно они могут провести ночь в номере только с представителем своего пола. Если для клиента не находится подходящего номера, он уходит искать ночлег в другое место. Создать многопоточное приложение, моделирующее работу гостиницы.

ИНТЕРПРЕТАЦИЯ

Вводится число – число клиентов. Затем генерируются случайных клиентов (дам или джентельменов), для каждого из которых выделен свой поток. Изначально заполняются двухместные номера, затем одноместные.

РЕШЕНИЕ

В данной задаче я применяю модель **взаимодействующих равных**.

Ниже представлены функции в программе:

# Read()

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – функция Read

Функция Read() считывает входные данные.

Если включён GENERATE (строка 23 раскомменчена), то число потоков генерируется случайно, а выходной файл задаётся по умолчанию (answer.txt).

Иначе пользователь вводит – число потоков (в ). Если число не попадает в указанные границы, то пользователю об этом сообщается. Если включён FILE\_OUT, то пользователь также вводит название выходного файла.

Далее инициализируется семафор write\_semaphore.

# Compute()

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 - функция Compute

Функция Compute() находит ответ на задачу, генерируя случайного человека и вызывая потоков, отвечающих за них. Каждый из потоков описывается функцией SpendTheNight(). Возвращает функция количество человек, которым не хватило места в отеле.

# SpendTheNight()

Функция SpendTheNight() обрабатывает каждого клиента. Выводится информация о том, что человек встал в очередь ({name} #{id} came to your hotel) и информация после обработки человека (либо, что он зарегистрировался в отеле ({name} #{id} checked in to the {single / double} room #{id\_room}, thread: {cur\_thread.get\_id()}), либо, что все номера в отеле были заняты ({name} #{id} could not find a room, thread: {cur\_thread.get\_id()})).

# Print()

Функция Print() выводит в консоль / выходной файл (в зависимости от FILE\_OUT) полученное расположение людей в отеле и количество людей, которые не получили номер.

ТЕСТИРОВАНИЕ ПРОГРАММЫ

1. Некорректный ввод ()

Изображение выглядит как текст

Автоматически созданное описаниеПрограмма отработала успешно.2. Некорректный ввод ()

Изображение выглядит как текст

Автоматически созданное описаниеПрограмма отработала успешно.

3.

Изображение выглядит как текст

Автоматически созданное описание

Программа отработала успешно.

4. (см. tests/answer.txt)

Изображение выглядит как текст

Автоматически созданное описаниеПрограмма отработала успешно.

5. (см. tests/answer2.txt)

Изображение выглядит как текст

Автоматически созданное описание

Программа отработала успешно

6. (см. tests/answer3.txt)

Изображение выглядит как текст

Автоматически созданное описание

Программа отработала успешно

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ошибки многопоточности в C++ [Электронный ресурс]. Режим доступа: https://otus.ru/nest/post/145/, свободный. (дата обращения: 17.11.2020)
2. Simple example of threading in C++ [Электронный ресурс]. Режим доступа: https://stackoverflow.com/questions/266168/simple-example-of-threading-in-c, свободный. (дата обращения: 17.11.2020)
3. std::thread::hardware\_concurrency [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/thread/thread/hardware\_concurrency, свободный. (дата обращения: 17.11.2020)
4. Measuring execution time of a fuction in C++ [Электронный ресурс]. Режим доступа: https://stackoverflow.com/questions/22387586/measuring-execution-time-of-a-function-in-c, свободный. (дата обращения: 17.11.2020)
5. Практические приёмы построения многопоточных приложений [Электронный ресурс]. Режим доступа: http://www.softcraft.ru/edu/comparch/tasks/t03/, свободный. (дата обращения: 17.11.2020)
6. Choosing the number of threads at runtime [Электронный ресурс]. Режим доступа: https://livebook.manning.com/book/c-plus-plus-concurrency-in-action/chapter-2/92/, свободный. (дата обращения: 17.11.2020)

КОД ПРОГРАММЫ

Смотрите multiThread.cpp

#include <iostream>  
#include <optional>  
#include <thread>  
#include <chrono>  
#include <random>  
#include <fstream>  
#include <list>  
#include <vector>  
#include <mutex>  
  
//#define GENERATE  
//#define FILE\_OUT  
  
constexpr int64\_t MIN\_VALUE = 1000;  
constexpr int64\_t MAX\_VALUE = 999'999'999ll;  
  
std::mutex mut;  
  
bool Read(int \*n, int \*thread\_number, std::string \*file\_name) { // NOLINT  
#ifdef GENERATE  
 std::mt19937 gen(std::chrono::high\_resolution\_clock::now().time\_since\_epoch().count());  
 std::uniform\_int\_distribution<int> dist\_n(2, 9);  
 \*n = dist\_n(gen);  
 std::uniform\_int\_distribution<int64\_t> dist\_t(1, MAX\_VALUE - MIN\_VALUE + 1);  
 \*thread\_number = dist\_t(gen);  
 \*file\_name = "answer.txt";  
 std::cout << "Number n: " << \*n << '\n';  
 std::cout << "Amount of threads: " << \*thread\_number << '\n';  
#else  
 std::cout << "Input your number n (in (1, 10)): ";  
 std::cin >> \*n;  
 if (\*n <= 1 || \*n >= 10) {  
 std::cout << "Incorrect number!\n";  
 return false;  
 }  
 // see https://livebook.manning.com/book/c-plus-plus-concurrency-in-action/chapter-2/92 for details  
 unsigned int hardware\_threads = std::thread::hardware\_concurrency();  
 std::cout << "Optimal number of threads for your machine: " << (hardware\_threads != 0 ? hardware\_threads : 2)  
 << '\n';  
 std::cout << "Input amount of threads (in [1, 10000]): ";  
 std::cin >> \*thread\_number;  
 if (\*thread\_number < 1 || \*thread\_number > 10000) {  
 std::cout << "Incorrect amount of threads!\n";  
 return false;  
 }  
#endif  
#ifdef FILE\_OUT  
 std::cout << "Input name of output file: ";  
 std::cin >> \*file\_name;  
 std::ofstream out(\*file\_name);  
 if (!out.is\_open()) {  
 std::cout << "Incorrect file name!\n";  
 out.close();  
 return false;  
 }  
 out.close();  
#endif  
 return true;  
}  
  
unsigned int Fun(int64\_t x) {  
 unsigned int y = 0; // [0, 1024)  
 while (x != 0) {  
 y |= 1u << unsigned(x % 10);  
 x /= 10;  
 }  
 return y;  
}  
  
void ComputeThread(int n, int64\_t from, int64\_t to, std::list<int64\_t> &numbers) {  
 for (int64\_t i = from; i <= to; ++i) {  
 if (Fun(i) == Fun(i \* n)) {  
 mut.lock();  
 numbers.push\_back(i);  
 mut.unlock();  
 }  
 }  
}  
  
void Compute(int n, int64\_t thread\_number, std::list<int64\_t> &numbers) {  
 int64\_t loop\_size = (MAX\_VALUE - MIN\_VALUE + 1) / thread\_number;  
 std::vector<std::thread> thr(thread\_number);  
 for (int64\_t i = 0; i < thread\_number; ++i) {  
 if (i != thread\_number - 1) {  
 thr[i] = std::thread{ComputeThread, n, MIN\_VALUE + loop\_size \* i, MIN\_VALUE + loop\_size \* (i + 1) - 1,  
 std::ref(numbers)};  
 } else {  
 thr[i] = std::thread{ComputeThread, n, MIN\_VALUE + loop\_size \* i, MAX\_VALUE, std::ref(numbers)};  
 }  
 }  
 for (std::thread &Thread : thr) {  
 Thread.join();  
 }  
}  
  
void Print(int n, int threads\_num, const std::string &file\_name, const std::list<int64\_t> &numbers) {  
#ifdef FILE\_OUT  
 std::ofstream out(file\_name);  
 out << "Number n: " << n << '\n';  
 out << "Amount of threads: " << threads\_num << '\n';  
 for (int64\_t x : numbers) {  
 out << x << '\n';  
 }  
 out.close();  
#else  
 for (int64\_t x : numbers) {  
 std::cout << x << '\n';  
 }  
#endif  
}  
  
int main() {  
 std::ios\_base::sync\_with\_stdio(false);  
 int n, thread\_number;  
 std::string file\_name;  
 if (!Read(&n, &thread\_number, &file\_name)) {  
 return 1;  
 }  
  
 std::list<int64\_t> numbers;  
 auto begin = std::chrono::steady\_clock::now();  
  
 Compute(n, thread\_number, numbers);  
  
 auto end = std::chrono::steady\_clock::now();  
 auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);  
  
 Print(n, thread\_number, file\_name, numbers);  
 std::cout << "Time (computation): " << elapsed\_ms.count() / 1000. << " sec\n";  
 return 0;  
}