ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Кафедра программной инженерии

Домашнее задание №3

по дисциплине «Архитектура вычислительных систем»

Тема: «практические приемы построения многопоточных приложений»

Пояснительная записка

Исполнитель: студент 2 курса,

Шалаева Марина Андреевна

Группа БПИ191

Москва 2020

СОДЕРЖАНИЕ

СТРУКТУРА РАБОТЫ

1) Папка «Code», содержащая в себе .cpp файл с исходным кодом программы (который также представлен в приложении 1)
2) Папка «Work», содержащая в себе папку «files» с входными и выходными данными, .exe файл и пять .bat файлов
3) Пояснительная записка.pdf – отчет о проделанной работе в формате pdf
4) Пояснительная записка.docx – отчет о проделанной работе в формате docx

ТЕКСТ ЗАДАНИЯ

Тема №27:

Пляшущие человечки. На тайном собрании глав преступного мира города Лондона председатель собрания профессор Мориарти постановил: отныне вся переписка между преступниками должна вестись тайнописью. В качестве стандарта были выбраны "пляшущие человечки", шифр, в котором каждой букве латинского алфавита соответствует хитроумный значок. Реализовать многопоточное приложение, шифрующее исходный текст (в качестве ключа используется кодовая таблица, устанавливающая однозначное соответствие между каждой буквой и каким-нибудь числом). Каждый поток шифрует свои кусочки текста. При решении использовать парадигму портфеля задач.

МОДЕЛЬ ВЫЧИСЛЕНИЯ

Для хранения кодовой таблицы было решено использовать контейнер **std::map< char, short>**, в котором ключами являются строчные буквы латинского алфавита, а значениями – случайные числа в диапазоне [10, 99]. Случайные числа присваются в значения map'а с помощью метода **void fillAlphabet()**. Поскольку программа кодирует только строчные буквы латинского алфавита, каждую обработанную букву она приводит к нижнему регистру, а все неподходящие значения оставляет в неизменном виде.

Для работы с входными и выходными данными была использована файловая система. Все входные файлы находятся в папке «files\input\testN.txt», выходные – в папках «files\output\answerN.txt» (закодированный текст), «files\output_alphabet\alphabetN.txt» (алфавит и кодировки всех букв), где N – номер теста.

Формат взаимодействия с программой – ввод данных в консоль в формате « <имя .exe файла> <путь к входному файлу> < путь к выходному файлу > <путь к алфавиту> <количество потоков> ».

В зависимости от количества потоков и количества символов во входном тексте формируется количество символов, которое обрабатывает один поток. В случае, если потоков больше, чем символов в тексте, количество потоков автоматически заменяется на количество символов в тексте.

При разработке приложения была использована парадигма параллельного программирования взаимодействующие равные и парадигма портфеля задач.

Взаимодействующие равные – модель, в которой исключен не занимающийся непосредственными вычислениями управляющий поток.

Распределение работ в таком приложении либо фиксировано заранее, либо динамически определяется во время выполнения. Одним из распространенных способов динамического распределения работ является «портфель задач». Портфель задач, как правило, реализуется с помощью разделяемой переменной, доступ к которой в один момент времени имеет только один процесс.

Вычислительная задача делится на конечное число подзадач. Как правило, каждая подзадача должна выполнить однотипные действия над разными данными. Подзадачи нумеруются, и каждому номеру определяется функция, которая однозначно отражает номер задачи на соответствующий ему набор данных. Создается переменная, которую следует выполнять следующей. Каждый поток сначала обращается к портфелю задач для

выяснения текущего номера задачи, после этого увеличивает его, потом берет соответствующие данные и выполняет задачу, затем обращается к портфелю задач для выяснения следующего номера задачи.

То есть поток получает задачу из портфеля и пока задача остается не выполненной, поток ее решает, а затем снова получает задачу из портфеля.[1]

В случае данного приложения портфелем задач является вектор **std::vector<std::thread> threads(numberOfThreads)**; каждой подзадачей является кодирование определенного кусочка текста. Во избежание кодирования одной и той же части текста различными потоками, предусмотрена переменная **index**, увеличивающаяся после того, как поток «взял» подзадачу. Закодированный под определенным индексом символ будет помещен в другой массив (с выходными данными) под тем же индексом, что позволяет избежать накладок.

ТЕСТИРОВАНИЕ

В качестве входных аргументов программа принимает четыре аргумента. Первый вводимый аргумент – путь к входному файлу или его имя, второй аргумент – коэффициент путь к выходному файлу или его имя, третья переменная – путь к выходному файлу для букв и их числовых значений или его имя, четвертая переменная – количество исполняемых потоков. Предъявляемые требования: первые три параметра должны содержать пути к файлам или их названия, четвертый параметр должен быть натуральным числом. В случае ввода некорректных данных программа выводит пользователю сообщение о том, в каком формате должны вводиться входные данные и завершает свою работу.

Для ускорения тестирования в папке «Work» находятся .bat файлы для тестирования корректных и некорректных данных.

ПРИЛОЖЕНИЕ 1

Текст программы

```
1.  /* Вариант 27:
2.   *
3.   * Пляшущие человечки.
4.   * На тайном собрании глав преступного мира города
5.   * Лондона председатель собрания профессор Мориарти
6.   * постановил: отныне вся переписка между преступниками
7.   * должна вестись тайнописью. В качестве стандарта
8.   * были выбраны "пляшущие человечки", шифр, в котором
9.   * каждой букве латинского алфавита соответствует
10.  * хитроумный значок. Реализовать многопоточное при-
11.  * ложение, шифрующее исходный текст (в качестве ключа
12.  * используется кодовая таблица, устанавливающая одноз-
13.  * начное соответствие между каждой буквой и каким-нибудь
14.  * числом). Каждый поток шифрует свои кусочки текста.
15.  * При решении использовать парадигму портфеля задач.
16.  *
17.  * ФИО: Шалаева Марина Андреевна
18.  * Группа: БПИ191
19.  */
20.
21. #include <iostream>
22. #include <fstream>
23. #include <string>
24. #include <map>
25. #include <vector>
26. #include <ctime>
27. #include <thread>
28. #include <algorithm>
29.
30.  // Имя входного файла.
31. std::string input;
32. // Имя выходного файла.
33. std::string output;
34. // Имя выходного файла для букв и их числовых значений.
35. std::string output_alphabet;
36.
37. // Переменная для считанного из файла текста (наше портфолио).
38. std::string text;
39. // Строковый массив для записи закодированных переменных.
40. std::string* encoded_text{};
41. // Словарь (карта) для хранения строчных букв латинского алфавита и их численных
    значений.
42. std::map<char, short> alphabet;
43. // Вспомогательный массив, индексы элементов которого служат значениями букв.
44. short numbers[90] = { 0 };
45.
46. // Индекс для работы потоков.
47. int index = 0;
48. // Количество потоков.
49. int numberOfThreads;
50. // Количество символов, которое кодирует один поток.
51. int numberOfLetters;
52.
53.
54. /// <summary>
55. /// Функция, проверяющая, является ли
56. /// входной параметр натуральным числом.
57. /// </summary>
58. /// <param name="s"> - входная строка</param>
59. /// <returns>является ли строка числом</returns>
60. bool isNumber(const std::string& s) {
61.     return !s.empty() && std::find_if(s.begin(),
62.         s.end(),
63.         [](char c) { return !std::isdigit(c); })
```

```
        == s.end();
}

/// <summary>
/// Функция, дополняющая название файла до
/// полного пути к нему.
/// </summary>
void addingFullPath() {

    if (input.find("files\\input\\") == std::string::npos)
        input = "files\\input\\" + input;
    if (output.find("files\\output\\") == std::string::npos)
        output = "files\\output\\" + output;
    if (output_alphabet.find("files\\output_alphabet\\") == std::string::npos)
        output_alphabet = "files\\output_alphabet\\" + output_alphabet;

}

/// <summary>
/// Фукция для пересоздания выходного
/// файла, на случай если тот уже создан.
/// </summary>
void createNewOutputFile() {
    std::ofstream out;
    out.open(output, std::ios::out);
    out << "";
    out.close();
}

/// <summary>
/// Функция для заполнения файла
/// данными из словаря (карты).
/// </summary>
void fillAlphabetFiles() {

    std::ofstream out;
      out.open(output_alphabet, std::ios::out);

      out << "--------\n";
      for (auto& item : alphabet) {
          out << " " << item.first << " | " << item.second << "\n--------\n";
      }
      out.close();
  }

  /// <summary>
  /// Функция для заполнения словаря (карты)
  /// строчными буквами латинского алфавита
  /// (ключи) и случайными числами от 10
  /// до 99 (значения).
  /// </summary>
  void fillAlphabet() {

      for (size_t i = 0; i < 26; i++) {

          srand((unsigned int)time(NULL));

          int index_of_numbers_array = rand() % 90;

          if (numbers[index_of_numbers_array] == 0) {
              alphabet[(char)(i + 97)] = index_of_numbers_array + 10;
              numbers[index_of_numbers_array] = 1;
          }
          else
              i--;
      }
      fillAlphabetFiles();
  }
```

```
132.
133.  /// <summary>
134.  /// Функция для считывания входных
135.  /// данных из файла.
136.  /// </summary>
137.  void readFromFile() {
138.      std::ifstream in;
139.      in.open(input, std::ios::in);
140.
141.      if (!in.is_open()) {
142.          std::cout << "Opening of the file failed!\n";
143.      }
144.      else {
145.          char x;
146.          text = "";
147.          while ((x = in.get()) != EOF) {
148.              text += tolower(x);
149.          }
150.      }
151.      in.close();
152.  }
153.
154.
155.  /// <summary>
156.  /// Функция для работы одного потока,
157.  /// кодирующая часть текста, соответствующую
158.  /// переданному индексу.
159.  /// </summary>
160.  /// <param name="i"> - индекс потока</param>
161.  /// <returns>закодированный текст</returns>
162.  void encode(int i) {
163.
164.      for (int j = i * numberOfLetters; j < i * numberOfLetters + numberOfLetters; ++j) {
165.          if (j >= text.size())
166.              break;
167.
168.          std::map<char, short>::iterator it;
169.          char letter = text[j];
170.          it = alphabet.find(letter);
171.
172.          if (it == alphabet.end())
173.              encoded_text[j] = (char)(letter);
174.          else
175.              encoded_text[j] = std::to_string(it->second);
176.      }
177.
178.  }
179.
180.  /// <summary>
181.  /// Функция для распределения задач по
182.  /// различным потокам.
183.  /// </summary>
184.  void launchPortfolio() {
185.
186.      numberOfLetters = ((int)text.size() / numberOfThreads) + 1;
187.      if (numberOfThreads > text.size()) {
188.          /* Если введенное количество потоков
189.           * меньше количества символов в тексте,
190.           * то приравниваем количество потоков
191.           * к размеру текста. */
192.          std::cout << "The number of threads is bigger than number of\n"
193.              "letters in the text! Program will use "
194.              << text.size() << "\nthreads instead of "
195.              << numberOfThreads;
196.          numberOfLetters = 1;
197.          numberOfThreads = text.size();
198.      }
199.      /* Инициализируем динамический массив,
```

```
200.       * в который мы запишем закодированные
201.       * буквы, используя соответствующие
202.       * индексы. */
203.      encoded_text = new std::string[text.size()];
204.
205.      // Инициализируем вектор потоков.
206.      std::vector<std::thread> threads(numberOfThreads);
207.      /* Цикл, вызывающий функцию для работы каждого
208.       * потока. В качестве аргумента подается index,
209.       * увеличивающий свое значение на каждой итерации. */
210.      for (int i = 0; i != numberOfThreads; ++index, ++i) {
211.          threads[i] = std::thread(encode, index);
212.      }
213.      for (int i = 0; i != numberOfThreads; ++i) {
214.          threads[i].join();
215.      }
216.
217.      std::ofstream out;
218.      out.open(output, std::ios::app);
219.      // Запись задокированного текста в файл.
220.      for (int i = 0; i < text.size(); ++i) {
221.          out << encoded_text[i];
222.      }
223.      out.close();
224.
225.      // Удаление динамического массива.
226.      delete[]encoded_text;
227.  }
228.
229.  int main(int argc, char* argv[]) {
230.
231.      /* Если количество введенных параметров меньше
232.      * нужного количества, сообщаем об этом пользователю
233.      * и завершаем работу приложения. */
234.      if (argc <= 4) {
235.          std::cout << "An invalid format of input params!\n"
236.              "Use the name of input file as the first value,\n"
237.              "the name of output file as the second one, the\n"
238.              "name of output file for letters and its codes as\n"
239.              "the third one and number of threads as the forth one.\n";
240.          exit(0);
241.      }
242.
243.      if (!isNumber(argv[4])) {
244.          std::cout << "An invalid format of input params!\n"
245.              "Number of threads should be integer "
246.              "number!";
247.          exit(0);
248.      }
249.
250.      if (atoi(argv[4]) <= 0) {
251.          std::cout << "An invalid format of input params!\n"
252.              "Number of threads should be integer "
253.              "number!";
254.          exit(0);
255.      }
256.
257.      input = argv[1];
258.      output = argv[2];
259.      output_alphabet = argv[3];
260.      numberOfThreads = atoi(argv[4]);
261.
262.      addingFullPath();
263.      createNewOutputFile();
264.      fillAlphabet();
265.      readFromFile();
266.      launchPortfolio();
267.
```

```
268.      return 0;
269.  }
```

ИСТОЧНИКИ

1) Парадигмы параллельного программирования [Электронный ресурс] //URL: https://pro-prof.com/forums/topic/parallel-programming-paradigms (Дата обращения: 16.11.2020, режим доступа: свободный).
2) Многопоточное программирование [Электронный ресурс] //URL: http://softcraft.ru/edu/comparch/practice/thread/02-sync/ (Дата обращения: 15.11.2020, режим доступа: свободный).