

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Кафедра программной инженерии

Домашнее задание №4

по дисциплине «Архитектура вычислительных систем»

Цель: «изучение применения OpenMP для разработки многопоточных приложений»

Вариант **27**

Пояснительная записка

Исполнитель: студент 2 курса,

Шалаева Марина Андреевна

Группа БПИ191

Москва 2020

СОДЕРЖАНИЕ

СТРУКТУРА РАБОТЫ	3
ТЕКСТ ЗАДАНИЯ	4
МОДЕЛЬ ВЫЧИСЛЕНИЯ.....	5
ТЕСТИРОВАНИЕ.....	10
ПРИЛОЖЕНИЕ 1.....	11
ИСТОЧНИКИ.....	15

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 восемь .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]. Данный диапазон был выбран для однозначного декодирования: каждой букве соответствует одно неиспользованное до этого двузначное число. Случайные числа присваиваются значениям map'a 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 (например, кириллицу, цифры, специальные символы (: ; , ? & и т.д.)).

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>».

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.

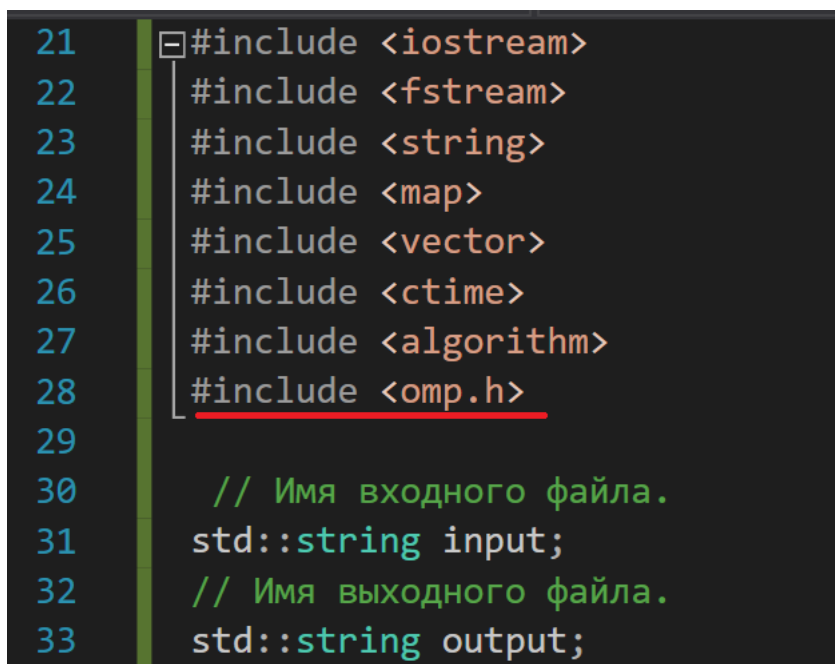
То есть поток получает задачу из портфеля и пока задача остается не выполненной, поток ее решает, а затем снова получает задачу из портфеля.^[1]

Также в данной работе важным пунктом было изучение работы с OpenMP.

OpenMP (Open Multi-Processing) — открытый стандарт для распараллеливания программ на языках C, C++ и Фортран. Дает описание совокупности директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью.^[3]

Существует множество разновидностей параллельных вычислительных систем — многоядерные/многопроцессорные компьютеры, кластеры, системы на видеокартах, программируемые интегральные схемы и т.д. Библиотека OpenMP подходит только для программирования систем с общей памятью, при этом используется параллелизм потоков. Потоки создаются в рамках единственного процесса и имеют свою собственную память. Кроме того, все потоки имеют доступ к памяти процесса.

Для использования библиотеки OpenMP было необходимо подключить заголовочный файл <omp.h> (см. рисунок 1), а также добавить опцию сборки -fopenmp (для компилятора gcc (например, компилятор среды разработки CLion), см. рисунок 2) или установить соответствующий флажок в настройках проекта (для Visual Studio, рисунок 3).



```
21 #include <iostream>
22 #include <fstream>
23 #include <string>
24 #include <map>
25 #include <vector>
26 #include <ctime>
27 #include <algorithm>
28 #include <omp.h>
29
30 // Имя входного файла.
31 std::string input;
32 // Имя выходного файла.
33 std::string output;
```

Рисунок 1 – Подключение <omp.h>

```

1 cmake_minimum_required(VERSION 3.17)
2 project(ABC_OpenMP)
3
4 # added -fopenmp
5 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -fopenmp")
6 set(CMAKE_CXX_STANDARD 14)
7
8 set(SOURCE_FILES main.cpp)
9 add_executable(ABC_OpenMP ${SOURCE_FILES})

```

Рисунок 2 – Добавление опции сборки -fopenmp в CLion

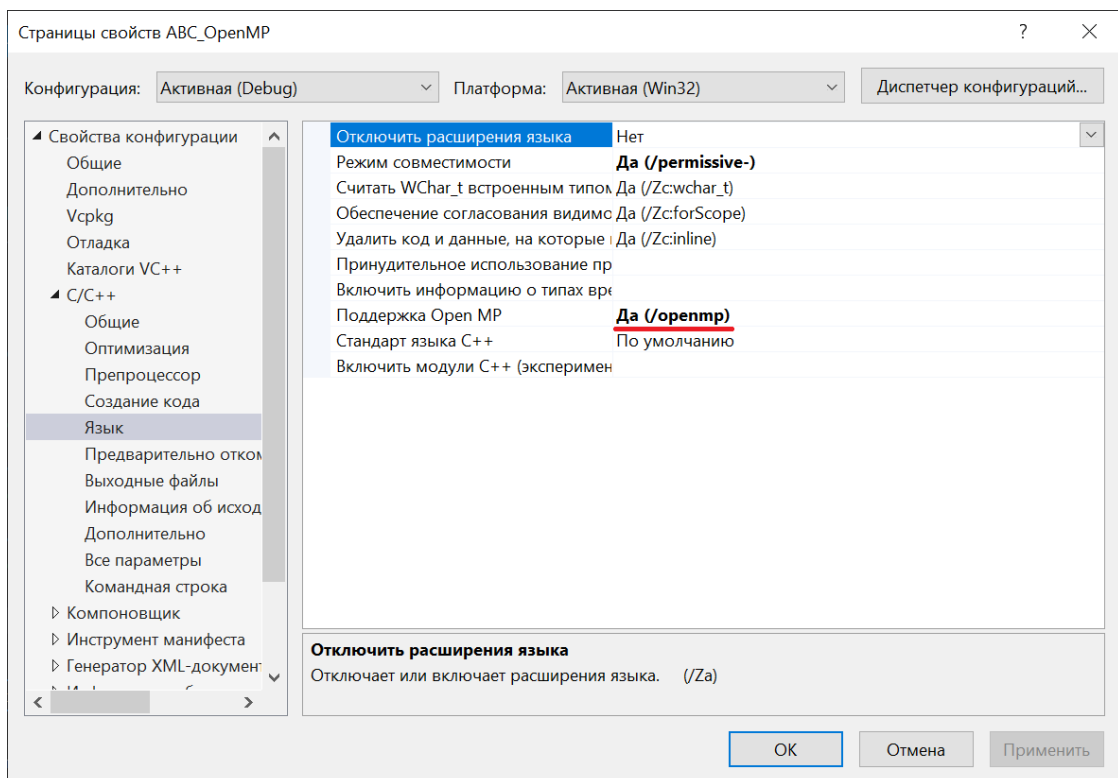


Рисунок 3 - Добавление опции в Visual Studio

После запуска программы создается единственный процесс, который начинает выполняться, как и обычная последовательная программа. Встретив параллельную область (задаваемую директивой `#pragma omp parallel`) процесс порождает ряд потоков (их число можно задать явно, однако по умолчанию будет создано столько потоков, сколько в вашей системе вычислительных ядер). Границы параллельной области выделяются фигурными скобками, в конце области потоки уничтожаются. Все потоки создаются одним (главным) потоком, который существует все время работы процесса. Такой поток в OpenMP называется `master`, все остальные потоки многократно создаются и уничтожаются. Стоит

отметить, что директивы `parallel` могут быть вложенными, при этом в зависимости от настроек могут создаваться вложенные потоки.

Все переменные, созданные до директивы `parallel`, являются общими для всех потоков. Переменные, созданные внутри потока, являются локальными и доступны только текущему потоку. При изменении общей переменной одновременно несколькими потоками возникает состояние гонок (мы не можем гарантировать какой-либо конкретный порядок записи и, следовательно, результат) — это проблема и допускать такое нельзя. Такая же проблема возникает, когда один поток пытается читать переменную в то время, как другой ее изменяет.^[4]

Для решения проблемы существует директива `critical`. В критической секции в один момент времени может находиться только один поток, остальные ожидают ее освобождения. Правилom хорошего тона считается, если критическая секция содержит обращения только к одному разделяемому ресурсу.

Для ряда операций более эффективно использовать директиву `atomic`, чем критическую секцию. Она ведет себя также, но работает чуть быстрее. Применять ее можно для операций префиксного/постфиксного инкремента/декремента и операции типа `X BINOP = EXPR`, где `BINOP` представляет собой не перегруженный оператор `+`, `*`, `-`, `/`, `&`, `^`, `|`, `<<`, `>>`.

В случае разработанного мной приложения, было решено воспользоваться директивой `parallel for` (см. рисунок 4). Представленная на рисунке 4 функция `void encode()` представляет из себя секцию в программе, в которой происходит разделение задач между потоками путем того, что каждый порожденный поток «забирает» себе одну итерацию цикла. Поскольку разные потоки завершают свою работу в разное время, что не всегда

```
139 void encode() {
140
141     #pragma omp parallel for
142
143     for (int i = 0; i < text.size(); ++i) {
144
145         char letter;
146         std::map<char, short>::iterator it;
147         letter = text[i];
148         it = alphabet.find(letter);
149
150         if (it == alphabet.end())
151             encoded_text[i] = (char)(letter);
152         else
153             encoded_text[i] = std::to_string(it->second);
154     }
155 }
```

Рисунок 4 – Функция `encode()`

происходит последовательно, мной было принято решение использовать динамический массив **std::string* encoded_text** для хранения закодированных символов. Таким образом символ, соответствующий определенному индексу в тексте, кодируется и записывается в массив **encoded_text** под тем же индексом. После завершения цикла **for** происходит запись в выходной файл.

В данном случае использование директивы **parallel for** OpenMP уже подразумевает использование «портфеля задач», поскольку есть один исполняемый поток, порождающий новые потоки, забирающие подзадачи из переменной **std::string text**, в которой хранится считанный из файла текст. Подзадачей является кодирование определенного символа текста. После завершения работы одного потока индекс **i** инкрементируется, и, следовательно, следующий поток не имеет доступа к уже закодированному символу, соответствующему индексу **i**, 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 – путь к выходному файлу или его имя, третья переменная – путь к выходному файлу для букв и их числовых значений или его имя. Предъявляемые требования: параметры должны содержать пути к файлам или их названия. 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. Для работы с .bat файлами в операционной системе Windows необходимо открыть командную строку, изменить директорию на «...\Shalaeva_ABC_HW4\Work», «перетащить» нужный .bat файл в консоль и нажать Enter.

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 <algorithm>
28. #include <omp.h>
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.
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. /// <summary>
47. /// Функция, дополняющая название файла до
48. /// полного пути к нему.
49. /// </summary>
50. void addingFullPath() {
51.
52.     if (input.find("files\\input\\") == std::string::npos)
53.         input = "files\\input\\" + input;
54.     if (output.find("files\\output\\") == std::string::npos)
55.         output = "files\\output\\" + output;
56.     if (output_alphabet.find("files\\output_alphabet\\") == std::string::npos)
57.         output_alphabet = "files\\output_alphabet\\" + output_alphabet;
58. }
59.
60. /// <summary>
61. /// Функция для пересоздания выходного
62. /// файла, на случай если тот уже создан.
63. /// </summary>

```

```

64. void createNewOutputFile() {
65.     std::ofstream out;
66.     out.open(output, std::ios::out);
67.     out << "";
68.     out.close();
69. }
70.
71. /// <summary>
72. /// Функция для заполнения файла
73. /// данными из словаря (карты).
74. /// </summary>
75. void fillAlphabetFiles() {
76.
77.     std::ofstream out;
78.     out.open(output_alphabet, std::ios::out);
79.
80.     out << "-----\n";
81.     for (auto& item : alphabet) {
82.         out << " " << item.first << " | " << item.second
83.             << "\n-----\n";
84.     }
85.     out.close();
86. }
87.
88. /// <summary>
89. /// Функция для заполнения словаря (карты)
90. /// строчными буквами латинского алфавита
91. /// (ключи) и случайными числами от 10
92. /// до 99 (значения).
93. /// </summary>
94. void fillAlphabet() {
95.
96.     for (size_t i = 0; i < 26; i++) {
97.
98.         srand((unsigned int)time(nullptr));
99.
100.         int index_of_numbers_array = rand() % 90;
101.
102.         if (numbers[index_of_numbers_array] == 0) {
103.             alphabet[(char)(i + 97)] = index_of_numbers_array + 10;
104.             numbers[index_of_numbers_array] = 1;
105.         }
106.         else
107.             i--;
108.     }
109.     fillAlphabetFiles();
110. }
111.
112. /// <summary>
113. /// Функция для считывания входных
114. /// данных из файла.
115. /// </summary>
116. void readFromFile() {
117.     std::ifstream in;
118.     in.open(input, std::ios::in);
119.
120.     if (!in.is_open()) {
121.         std::cout << "Opening of the file failed!\n";
122.     }
123.     else {
124.         char x;
125.         text = "";
126.         while ((x = in.get()) != EOF) {
127.             text += tolower(x);
128.         }
129.     }
130.     in.close();
131. }

```

```

132.
133. /// <summary>
134. /// Функция для кодирования текста
135. /// с помощью нескольких потоков
136. /// с использованием OpenMP.
137. /// </summary>
138. void encode() {
139.
140. #pragma omp parallel for
141.
142.     for (int i = 0; i < text.size(); ++i) {
143.
144.         char letter;
145.         std::map<char, short>::iterator it;
146.         letter = text[i];
147.         it = alphabet.find(letter);
148.
149.         if (it == alphabet.end())
150.             encoded_text[i] = (char)(letter);
151.         else
152.             encoded_text[i] = std::to_string(it->second);
153.     }
154. }
155.
156. /// <summary>
157. /// Функция для распределения задач по
158. /// различным потокам.
159. /// </summary>
160. void launchPortfolio() {
161.
162.     /* Инициализируем динамический массив,
163.      * в который мы запишем закодированные
164.      * буквы, используя соответствующие
165.      * индексы. */
166.     encoded_text = new std::string[text.size()];
167.
168.     encode();
169.
170.     std::ofstream out;
171.     out.open(output, std::ios::app);
172.     // Запись задокированного текста в файл.
173.     for (int i = 0; i < text.size(); ++i) {
174.         out << encoded_text[i];
175.     }
176.     out.close();
177.
178.     // Удаление динамического массива.
179.     delete[] encoded_text;
180. }
181.
182. /// <summary>
183. /// Функция, проверяющая валидность входных
184. /// параметров и присваивающая необходимые
185. /// значения полям, если входные параметры
186. /// верные.
187. /// </summary>
188. /// <param name="argc"> - количество входных параметров</param>
189. /// <param name="argv"> - массив входных параметров</param>
190. void workingWithInputValues(const int& argc, char* argv[]) {
191.
192.     /* Если количество введенных параметров не равно
193.      * нужному количеству, сообщаем об этом пользователю
194.      * и завершаем работу приложения. */
195.     if (argc != 4) {
196.         std::cout << "An invalid format of input params!\n"
197.                     "Use the name of input file as the first value,\n"
198.                     "the name of output file as the second one and the\n"
199.                     "name of output file for letters and its codes as\n"

```

```

200.         "the third one.\n";
201.         exit(0);
202.     }
203.
204.     input = argv[1];
205.     output = argv[2];
206.     output_alphabet = argv[3];
207.
208.     addingFullPath();
209.     createNewOutputFile();
210. }
211.
212. int main(int argc, char* argv[]) {
213.
214.     workingWithInputValues(argc, argv);
215.     fillAlphabet();
216.     readFromFile();
217.     launchPortfolio();
218.
219.     return 0;
220. }

```

ИСТОЧНИКИ

- 1) Парадигмы параллельного программирования [Электронный ресурс] //URL: <https://pro-prof.com/forums/topic/parallel-programming-paradigms> (Дата обращения: 17.11.2020, режим доступа: свободный).
- 2) Многопоточное программирование [Электронный ресурс] //URL: <http://softcraft.ru/edu/comparch/practice/thread/02-sync/> (Дата обращения: 17.11.2020, режим доступа: свободный).
- 3) OpenMP [Электронный ресурс] //URL: <https://ru.wikipedia.org/wiki/OpenMP> (Дата обращения: 28.11.2020, режим доступа: свободный).
- 4) Учебник по OpenMP [Электронный ресурс] //URL: <https://pro-prof.com/archives/4335> (Дата обращения: 28.11.2020, режим доступа: свободный).
- 5) Getting Started with OpenMP [Электронный ресурс] //URL: <https://software.intel.com/content/www/us/en/develop/articles/getting-started-with-openmp.html> (Дата обращения: 29.11.2020, режим доступа: свободный).
- 6) Эффективное распределение нагрузки между потоками с помощью OpenMP [Электронный ресурс] //URL: <https://software.intel.com/content/www/ru/ru/develop/articles/more-work-sharing-with-openmp.html> (Дата обращения: 29.11.2020, режим доступа: свободный).