Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Матвеев Д. Е.

Группа: М8О-207Б-21

Вариант: 6

Преподаватель: Черемисинов Максим

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Замеры времени
8. Выводы

**Репозиторий**

<https://github.com/MrDenli/OsLabs>

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

Управление потоками в ОС

Обеспечение синхронизации между потоками

**Задание**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При

обработки использовать стандартные средства создания потоков операционной системы

(Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей

программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входящих

данных и количества потоков. Получившиеся результаты необходимо объяснить.

6) Произвести перемножение 2-ух матриц, содержащих комплексные числа

**Общие сведения о программе**

Программа компилируется из файла main.c при помощи cmake. В программе реализована многопоточность. Функции для работы с потоками. которые я использовал:

* pthread\_create() — создание потока с передачей ему аргументов. В случае успеха возвращает 0.
* pthread\_join() — ожидает завершения потока обозначенного THREAD\_ID. Если этот поток к тому времени был уже завершен, то функция немедленно возвращает значение.
* pthread\_mutex\_init() — инициализация мьютекса
* pthread\_mutex\_lock() — блокировка мьютекса
* pthread\_mutex\_lock() — открытие доступа к мьютексу
* pthread\_mutex\_destroy() - удаление мьютекса

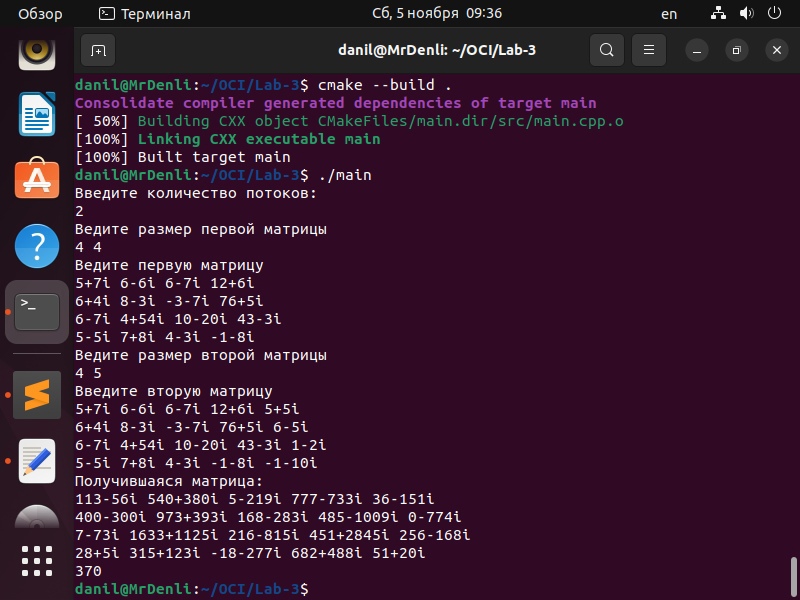
**Общий метод и алгоритм решения:**

Сначала нам нужно научиться работать с комплексными числами для этого пишем структуру complex\_number, функции умножения комплексных чисел, сложения комплексных числе, считывания и вывода комплексных чисел. Дальше задача становится достаточно тривиальной. Основная идея заключается в том, что мы делим количество строчек получившейся матрицы на количество потоков, столько строчек будет обрабатывать один поток. Запускаем эти потоки, каждый поток обрабатывает свое количество строчек матрицы.

**Исходный код**

1. #include<pthread.h>
2. #include<iostream>
3. #include<ctime>
4. #include<vector>
5. #include<fstream>
6. #include<chrono>
8. using namespace std;
10. pthread\_mutex\_t mutex;
11. int flag = 0;
12. int cell = 0;
14. struct complex\_number {
15. int a;
16. int b;
17. };
19. vector<vector<complex\_number>> answer;
21. complex\_number complex\_multiplication(complex\_number X, complex\_number Y) {
22. complex\_number ans;
23. ans.a = X.a \* Y.a + (-1) \* X.b \* Y.b;
24. ans.b = X.a \* Y.b + X.b \* Y.a;
25. return ans;
26. }
28. complex\_number complex\_adition(complex\_number X, complex\_number Y) {
29. complex\_number ans;
30. ans.a = X.a + Y.a;
31. ans.b = X.b + Y.b;
32. return ans;
33. }
35. void write\_complex\_number(complex\_number X) {
36. cout << X.a;
37. if (X.b == 0) {
38. return;
39. }
40. if (X.b > 0) {
41. cout << "+";
42. }
43. cout << X.b << "i";
44. return;
45. }
47. complex\_number read\_complex\_number() {
48. complex\_number ans;
49. cin >> ans.a;
50. cin >> ans.b;
51. char temp;
52. cin >> temp;
53. return ans;
54. }
56. struct arg\_to\_thread {
57. vector<vector<complex\_number>> A;
58. vector<vector<complex\_number>> B;
59. int partition;
60. int num\_of\_thread;
61. int count\_threads;
62. int n1;
63. int m1;
64. int n2;
65. int m2;
66. int n\_ans;
67. int m\_ans;
68. };
70. void\* thread\_func(void \*args)
71. {
72. arg\_to\_thread\* arguments = (arg\_to\_thread\*) args;
73. int num\_of\_thread = arguments->num\_of\_thread;
74. int partition = arguments->partition;
75. flag = 1;
76. int count\_threads = arguments->count\_threads;
77. int n1 = arguments->n1;
78. int m1 = arguments->m1;
79. int n2 = arguments->n2;
80. int m2 = arguments->m2;
81. int n\_ans = arguments->n\_ans;
82. int m\_ans = arguments->m\_ans;
83. for (int i = 0; i < partition; i++) {
84. int I = num\_of\_thread \* partition + i;
85. for (int J = 0; J < m\_ans; J++) {
86. for (int k = 0; k < m1; k++) {
87. complex\_number temp = complex\_multiplication((arguments->A)[I][k], (arguments->B)[k][J]);
88. answer[I][J] = complex\_adition(answer[I][J], temp);
89. }
90. cell++;
91. }
92. }
93. return 0;
94. }
96. int main(int argc, char const \*argv[])
97. {
98. string command;
99. int n1, n2, m1, m2;
100. vector<vector<complex\_number>> A;
101. vector<vector<complex\_number>> B;
102. cout << "1) Напишите work, если хотите переумножить матрицы" << endl << "2) Введите test, если хотите провести тест скорости" << endl;
103. cin >> command;
104. if (command == "work") {
105. cout << "Ведите размер первой матрицы" << endl;
106. cin >> n1 >> m1;
107. A.resize(n1, vector<complex\_number>(m1));
108. cout << "Ведите первую матрицу" << endl;
109. for (int i = 0; i < n1; i++) {
110. for (int j = 0; j < m1; j++) {
111. A[i][j] = read\_complex\_number();
112. }
113. }
114. cout << "Ведите размер второй матрицы" << endl;
115. cin >> n2 >> m2;
116. B.resize(n2, vector<complex\_number>(m2,complex\_number{}));
117. cout << "Введите вторую матрицу" << endl;
118. for (int i = 0; i < n2; i++) {
119. for (int j = 0; j < m2; j++) {
120. B[i][j] = read\_complex\_number();
121. }
122. }
123. }
124. else if (command == "test"){
125. cout << "Ведите размер первой матрицы" << endl;
126. cin >> n1 >> m1;
127. cout << "Ведите размер второй матрицы" << endl;
128. cin >> n2 >> m2;
129. A.resize(n1, vector<complex\_number>(m1,complex\_number{}));
130. B.resize(n2, vector<complex\_number>(m2,complex\_number{}));
131. }
132. if (m1 != n2) {
133. cout << "Матрицы таких размеров нельзя перемножить" << endl;
134. return 0;
135. }
137. int n\_ans = n1;
138. int m\_ans = m2;
139. answer.resize(n1);
140. for (int i = 0; i < n1; i++) {
141. answer[i].resize(m2);
142. }
144. for (int i = 0; i < n1; i++) {
145. for (int j = 0; j < m2; j++) {
146. answer[i][j].a = 0;
147. answer[i][j].b = 0;
148. }
149. }
151. cout << "введите колличество потоков" << endl;
152. int count\_threads;
153. cin >> count\_threads;
155. pthread\_t threads[count\_threads];
156. pthread\_mutex\_init(&mutex, NULL);
157. int partition = n\_ans / count\_threads;
159. if (count\_threads > n\_ans\*m\_ans) {
160. cout << "Введено слишком много потоков" << endl;
161. count\_threads = n\_ans;
162. }
164. struct arg\_to\_thread arg;
165. arg.partition = partition;
166. arg.count\_threads = count\_threads;
167. arg.n1 = n1;
168. arg.m1 = m1;
169. arg.n2 = n2;
170. arg.m2 = m2;
171. arg.n\_ans = n\_ans;
172. arg.m\_ans = m\_ans;
174. arg.A.resize(n1, vector<complex\_number>(m1));
175. arg.B.resize(n2, vector<complex\_number>(m2));
177. for (int i = 0; i < n1; i++) {
178. for (int j = 0; j < m1; j++) {
179. arg.A[i][j].a = A[i][j].a;
180. arg.A[i][j].b = A[i][j].b;
181. }
182. }
184. for (int i = 0; i < n2; i++) {
185. for (int j = 0; j < m2; j++) {
186. arg.B[i][j].a = B[i][j].a;
187. arg.B[i][j].b = B[i][j].b;
188. }
189. }
191. chrono::high\_resolution\_clock::time\_point begin = chrono::high\_resolution\_clock::now();
192. for (int i = 0; i < count\_threads; i++) {
193. arg.num\_of\_thread = i;
194. if (i == count\_threads - 1) {
195. partition += (n1 \* m2 % count\_threads);
196. arg.partition = partition;
197. }
198. int status = pthread\_create(&threads[i], NULL, thread\_func, (void\*)&arg);
199. flag = 0;
200. if (status != 0) {
201. cout<<"Create thread error"<<endl;
202. }
203. }
205. for (int i = 0; i < count\_threads; ++i) {
206. pthread\_join(threads[i], NULL);
207. }
208. chrono::high\_resolution\_clock::time\_point end = chrono::high\_resolution\_clock::now();
210. pthread\_mutex\_destroy(&mutex);
211. if (command == "work"){
212. cout << "Получившаяся матрица:" << endl;
213. for (int i = 0; i < n\_ans; i++) {
214. for (int j = 0; j < m\_ans; j++) {
215. write\_complex\_number(answer[i][j]);
216. cout << " ";
217. }
218. cout << endl;
219. }
220. }
221. cout<< "программа выполнена за :" << chrono::duration\_cast<chrono::milliseconds>(end-begin).count() << " миллисекунды" << endl;
222. return 0;
224. }

**Демонстрация работы программы**



**Замеры времени**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 поток | 2 потока | 4 потока | 16 потоков |
| Матрицы 100х200 и 200х100 | 47 мс | 29мс | 19мс | 15мс |
| Матрицы 1000х1000 и 1000х1000 | 25971мс | 17144мс | 5809мс | 2906мс |

**Выводы**

Мне понравилась данная лабораторная работа, я научился работать с потоками на языке с++. Я считаю эта лабораторная работа очень полезна для меня, потому что полученные навыки с большой вероятностью помогут мне в будущем. Работа с потоками незаменимый навык для работы программистом, а поняв, как работают потоки на языке с++, можно легко научиться работать с потоками на других языках.