Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по Лабораторной работе №4
по курсу «Анализы Алгоритмов»
на тему: «Параллельные вычисления на основе нативных потоков»

Студент ИУ7-56Б — Мансуров В. М.
(Группа) (Фамилия И. О.)
Преподаватели — Волкова Л. Л.
(Фамилия И. О.)

Москва — 2023 г.

Содержание

Введение **3**

1 Аналитическая часть **4**

1.1 Многопоточность 4

1.2 Термин частота 5

2 Конструкторская часть **7**

2.1 Разработка алгоритмов 7

3 Технологическая часть **12**

3.1 Требования к программному обеспечению 12

3.2 Средства реализации 12

3.3 Сведения о модулях программы 13

3.4 Реализация алгоритмов 13

4 Исследовательская часть **17**

4.1 Технические характеристики 17

4.2 Временные характеристики 17

4.3 Вывод 20

Заключение **21**

Список использованных источников **22**

Введение

По мере развития вычислительных систем программисты столкнулись с необходимостью производить параллельную обработку данных для улучшения отзывчивости системы, ускорения производимых вычислений и рационального использования вычислительных мощностей. Благодаря развитию процессоров стало возможным использовать один процессор для выполнения нескольких параллельных операций, что дало начало термину «многопоточность».

Целью данной лабораторной работы является изучение принципов и получение навыков организации параллельного выполнения операций.

Для поставленной цели необходимо выполнить следующие задачи.

1) Изучение основы распараллеливания вычислений.

2) Разработать программное обеспечение, которое реализует однопоточный алгоритм вычисления термина частоты для классификации.

3) Разработать и реализовать многопоточную версию данного алгоритма.

4) Определить средства программной реализации.

5) Выполнить замеры процессорного времени работы реализаций алгоритма.

6) Провести сравнительный анализ по времени работы реализаций алгоритмов.

1 Аналитическая часть

В данном разделе была представлена информация о многопоточности и исследуемом алгоритме выделения терминов для классификации.

1.1 Многопоточность

Многопоточность [1] (англ. *multithreading*)– это способность центрального процессора (ЦП) обеспечивать одновременное выполнение нескольких потоков в рамках использования ресурсов одного процессора. Поток – последовательность инструкций, которые могут исполняться параллельно с другими потоками одного и того же породившего их процесса.

Процессом [2] называют программу в ходе своего выполнения. Таким образом, когда запускается программа или приложение, создается процесс. Один процесс может состоять из одного или больше потоков. Таким образом, поток является сегментом процесса, сущностью, которая выполняет задачи, стоящие перед исполняемым приложением. Процесс завершается тогда, когда все его потоки заканчивают работу. Каждый поток в операционной системе является задачей, которую должен выполнить процессор. Сейчас большинство процессоров умеют выполнять несколько задач на одном ядре, создавая дополнительные, виртуальные ядра, или же имеют несколько физических ядер. Такие процессоры называются многоядерными.

При написании программы, использующей несколько потоков, следует учесть, что при последовательном запуске потоков и передаче управления исполняемому потоку не получится раскрыть весь потенциал многопоточности, т.к. большинство потоков будут существовать без дела. Необходимо создавать потоки для независимых, отнимающих много времени функций, и выполнять их параллельно, тем самым сокращая общее время выполнения процесса.

Одной из проблем, встающих при использовании потоков, является проблема совместного доступа к информации. Фундаментальным ограничением является запрет на запись из двух и более потоков в одну ячейку памяти одновременно.

Из того следует, что необходим примитив синхронизации обращения к данным – так называемый мьютекс (англ. *mutex – mutual exclusion*). Он может быть захвачен для работы в монопольном режиме или освобожден. Так, если 2 потока одновременно пытаются захватить мьютекс, успевает только один, а другой будет ждать освобождения.

Набор инструкций, выполняемых между захватом и освобождением мьютекса, называется *критической секцией*. Поскольку в то время, пока мьютекс захвачен, остальные потоки, требующие выполнения критической секции, ждут освобождения мьютекса, требуется разрабатывать программное обеспечение таким образом, чтобы критическая секция была минимальной.

1.2 Термин частота

Классификация текстов является одной из основных задач компьютерной лингвистики, поскольку к ней сводится ряд других задач: определение тематической принадлежности текстов, автора текста, эмоциональной окраски высказываний и др. Для обеспечения информационной и общественной безопасности большое значение имеет анализ в телекоммуникационных сетях контента, содержащего противоправную информацию. [3]

Под терминами документов будем понимать все одиночные слова, встреченные в тексте хотя бы одного документа коллекции, за исключением стоп-слов, то есть распространенных слов, не характеризующих документы по смыслу, например, предлогов, союзов и т. п. Вдобавок, каждой встреченной форме слова, например, в разных падежах и числах, будет соответствовать один и тот же термин, например, данное слово в начальной форме.

Термин частота (англ. TF) [4] — это отношение числа раз вхождений термина документа к общему количеству терминов в документе:

$$TF_k = \frac{t_k}{n}, \tag{1.1}$$

где t_k — количество вхождений k-го термина и n — общее количество терминов документа.

В данной лабораторной работе проводится распараллеливания алгоритма выделения терминов из выборки текстов на основе TF. Для этого все документы поровну распределяются между всеми потоками.

В качестве одного из аргументов каждый поток получает выделенный для него строку массива счетчиков TF длины N, где N — это количество терминов документа. Так как каждая строка массива передается в монопольное использование каждому потоку, не возникает конфликтов доступа к разделяемым ячейки памяти,следовательно, в использовании средства синхронизации в виде мьютекса нет нужды.

Вывод

В данном разделе была представлена информация о многопоточности и исследуемом алгоритме.

2 Конструкторская часть

В данном разделе разработаны схемы реализаций алгоритма выделения наиболее информативных терминов из выборки документов.

2.1 Разработка алгоритмов

На рисунках 2.1 – 2.3 приведены схемы однопоточной и многопоточной реализаций рассматриваемого алгоритма.

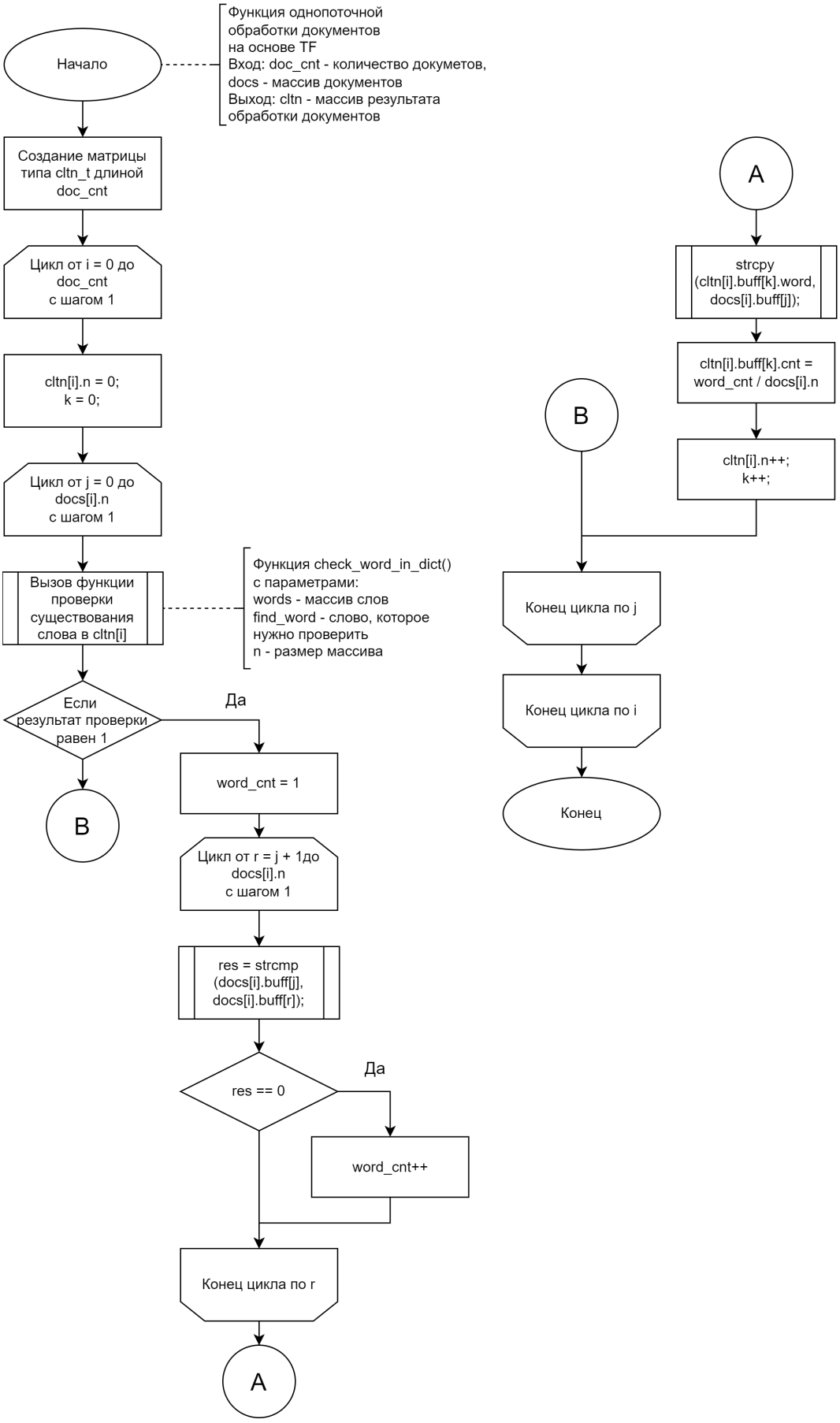

Рисунок 2.1 – Схема однопоточного алгоритма выделения терминов из выборки документов на основе частоты термина

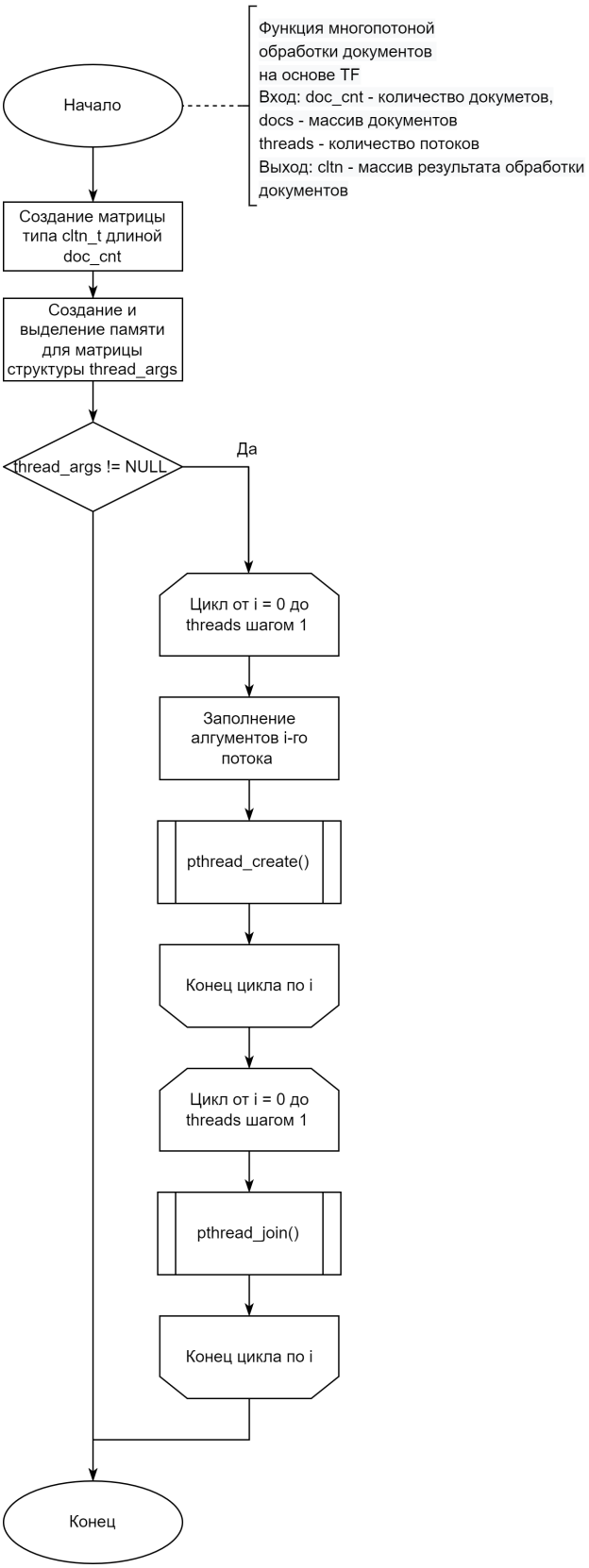

Рисунок 2.2 – Схема алгоритма работы основного потока, запускающего вспомогательные потоки

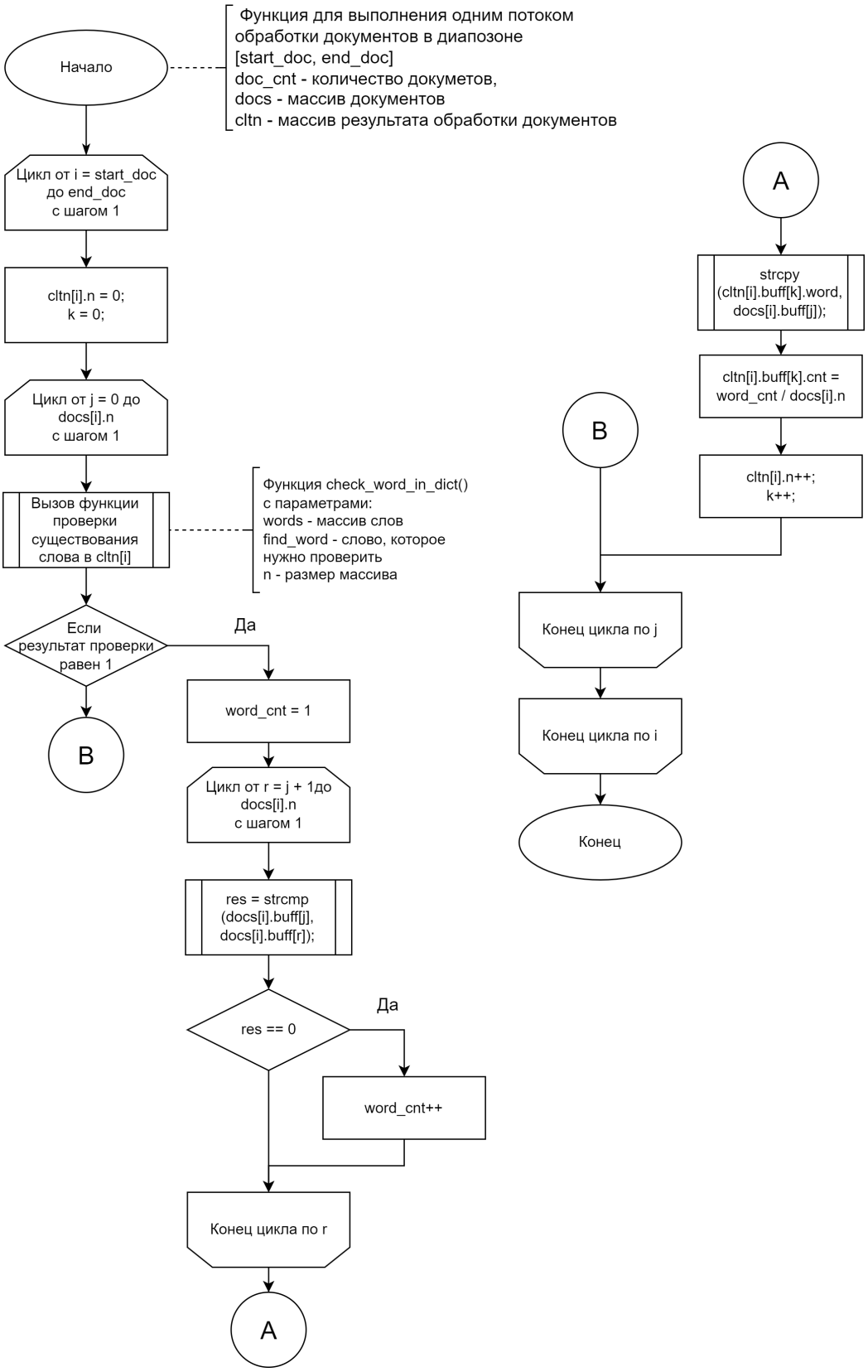

Рисунок 2.3 – Схема однопоточного алгоритма выделения терминов из выборки документов на основе частоты термина

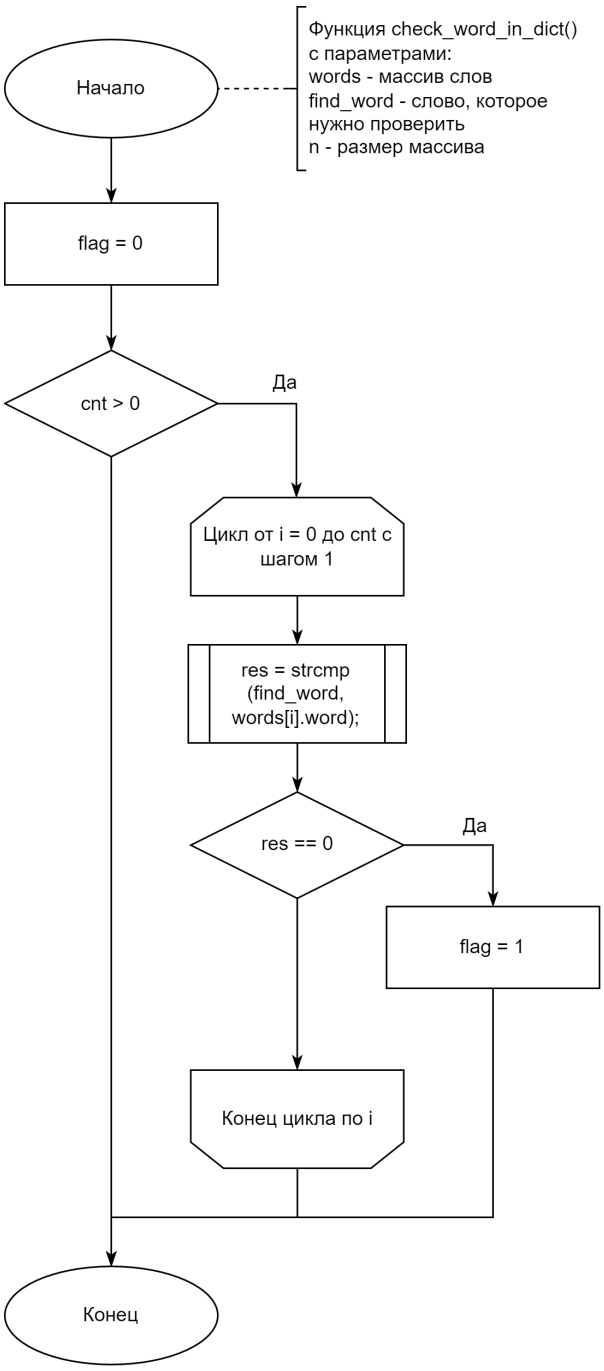

Рисунок 2.4 – Схема алгоритма проверки существования слова в массиве

Вывод

В данном разделе разработаны схемы реализаций рассматриваемого алгоритма

3 Технологическая часть

В данном разделе рассмотрены средства реализации, а также представлены листинги реализации алгоритма выделения наиболее информативных терминов, по метрике TF, из выборки документов.

3.1 Требования к программному обеспечению

К программе предъявлены ряд требований:

— иметь интерфейс для выбора действий;

— динамически выделять память под массив данных;

— работа с массивами и «нативными» потоками;

— замерять процессорное время алгоритмов сортировки.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык *C* [5]. Данный выбор обусловлен наличием у языка встроенной модулем измерения процессорного времени и соответствием с выдвинутыми техническими требованиям.

Для работы с потоками использовались функции модуля <pthread.h> [6]. Для работы с сущностью вспомогательного потока необходимо воспользоваться функцией *pthread_create*() для создания потока и указания функции, которую начнет выполнять созданный поток. Далее при помощи вызова *pthread_join*() необходимо (в рамках данной лабораторной работы) дождаться завершения всех вспомогательных потоков, чтобы в главном потоке обработать результаты их работы.

3.3 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` – файл, содержащий точку входа в программу. В нем происходит общение с пользователем и вызов алгоритмов;

— `tf_alg.cpp` — файл содержит функции динамического выделения, очищения памяти и однопоточную реализацию алгоритма;

— `pthread_tf.cpp` – файл содержит функции многопоточной реализации алгоритма;

— `time_meassure.cpp` — файл содержит функции, замеряющее процессорное время методов однопоточной и многопоточной реализаций алгоритма.

3.4 Реализация алгоритмов

В листингах 3.1 – 3.3 приведены реализации алгоритма выделения наиболее информативных терминов для каждого документа.

В качестве терминов в данной реализации рассматриваются слова, состоящие из латинских букв. В качестве документов рассматриваются строки, состоящие из таких слов.

Листинг 3.1 – Функция работы одного вспомогательного потока

```
1  int check_word_in_dict(const map *words, const char *find, size_t
       cnt) {
2      if (cnt <= 0)
3          return 0;
4
5      for (int i = 0; i < cnt; i++) {
6          if (strcmp(find, words[i].word) == 0)
7              return 1;
8      }
9      return 0;
10 }
11
12 double tf(int find_word_cnt, int words_cnt) {
13     return (double) find_word_cnt / words_cnt;
14 }
15
16 void couting_words(cltn_t *cltn, const mtr_t *doc) {
17     cltn->n = 0;
18     int word_cnt;
19     for (int i = 0, k = 0; i < doc->n; i++) {
20         if (check_word_in_dict(cltn->buff, doc->buff[i], cltn->n))
21             continue;
22         word_cnt = 1;
23         for (int j = i + 1; j < doc->n; j++)
24             if (strcmp(doc->buff[i], doc->buff[j]) == 0)
25                 word_cnt++;
26
27         strcpy(cltn->buff[k].word, doc->buff[i]);
28         cltn->buff[k].cnt = tf(word_cnt, doc->n);
29         cltn->n++;
30         k++;
31     }
32 }
33
34 void find_cltn_tf(mtr_t *docs, cltn_t *cltn, size_t docs_cnt) {
35     for (int i = 0; i < docs_cnt; i++)
36         couting_words(&cltn[i], &docs[i]);
37 }
```

Листинг 3.2 – Функция работы одного вспомогательного потока

```
1  void *thread_work(void *args) {
2      thread_args_t *targs = args;
3      int start = targs->start_doc;
4      int end = targs->end_doc;
5      mtr_t *docs = targs->docs;
6      cltn_t *cltn = targs->cltn;
7
8      for (int thr = start; thr < end; thr++) {
9          int word_cnt;
10         cltn[thr].n = 0;
11         for (int i = 0, k = 0; i < docs[thr].n; i++) {
12             if (check_word_in_dict(cltn[thr].buff,
                   docs[thr].buff[i], cltn[thr].n))
13                 continue;
14             word_cnt = 1;
15             for (int j = i + 1; j < docs[thr].n; j++) {
16                 if (strcmp(docs[thr].buff[i], docs[thr].buff[j])
                       == 0)
17                     word_cnt++;
18             }
19             //mutex_lock
20             strcpy(cltn[thr].buff[k].word, docs[thr].buff[i]);
21             cltn[thr].buff[k].cnt = tf(word_cnt, docs[thr].n);
22             cltn[thr].n++;
23             //mutex_unlock
24             k++;
25         }
26     }
27
28     return NULL;
29 }
```

Листинг 3.3 – Функция работы основного потока запускающего вспомогательные потоки

```
1  void parallel_find_cltn_tf(int threads, mtr_t *docs, cltn_t
       *cltn, size_t docs_cnt) {
2      if (threads <= 0)
3          return;
4
5      pthread_t *tid = malloc(threads * sizeof(pthread_t));
6      thread_args_t *args = malloc(threads * sizeof(thread_args_t));
7      if (tid && args) {
8          int delta_doc = docs_cnt / threads;
9          int last = docs_cnt % threads;
10         int start = 0;
11         int end = delta_doc + last;
12         for (int thr = 0; thr < threads; thr++) {
13             (args + thr)->start_doc = start;
14             (args + thr)->end_doc = end;
15             (args + thr)->docs = docs;
16             (args + thr)->cltn = cltn;
17             pthread_create(tid + thr, NULL, &thread_work, args +
                   thr);
18             start = end;
19             end += delta_doc;
20         }
21         for (int thread = 0; thread < threads; ++thread)
22             pthread_join(tid[thread], NULL);
23     }
24
25     free(tid);
26     free(args);
27 }
```

Вывод

В данном разделе были рассмотрены средства реализации, а также представлен листинг реализации алгоритма выделения наиболее информативных терминов для каждого документов.

4 Исследовательская часть

В данном разделе приведены технические характеристики устройства, на котором проводилось измерение времени работы программного обеспечения, а также результаты замеров времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени представлены далее.

— Процессор: Intel(R) Core(TM) i5-10300H CPU 2.50 ГГц [7].

— Количество ядре: 4 физических и 8 логических ядер.

— Оперативная память: 16 ГБайт.

— Операционная система: Windows 11 Pro 64-разрядная система [8].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Временные характеристики

Для замеров времени использовалась функция получения значения системных часов *clock_gettime*() [9]. Функция применялась два раза — в начале и в конце измерения времени, значения полученных временных меток вычитались друг из друга для получения времени выполнения программы. Документы заполнялись случайными буквами латинского алфавита. Замеры проводились по 1000 раз для набора значений количества потоков 0, 1, 2, 4, 6, 8, 16, 32, 64, где значение количества потоков 0 соответствует однопоточной программе, а значение 1 — программе, создающей один дополнительный поток, выполняющий все вычисления.

В таблице 4.1 представлены замеры времени выполнения программы в зависимости от количества потоков. Замеры проводились на документах,

представленными строками длиной в 64 символа и диапазон количеств документов начиная с 512 до 6656 с шагом 512.

Таблица 4.1 – Результаты нагрузочного тестирования (в мкс)

Ко-во документов	**Время, мкс**								
	0	**1**	**2**	**4**	**6**	**8**	**16**	**32**	**64**
512	1008	1204	1070	912	933	822	1167	1263	2981
1024	1421	1793	1673	1540	1428	1230	1736	2360	3134
1536	1829	2212	1878	1668	1590	1475	1974	2877	3479
2048	2344	2826	2039	1913	1899	1789	2536	3216	3500
2560	2529	3168	2459	2382	2268	2076	3171	3379	3681
3072	3192	3332	3018	3001	3099	2712	3816	4128	4307
3584	3639	3679	3491	3477	3288	3091	4188	4489	4486
4096	4135	4328	3630	3595	3323	3230	4335	4623	4628
4608	4679	4781	3786	3584	3485	3310	4436	4718	4815
5120	5201	5228	3837	3753	3536	3491	4525	4929	5278
5632	5580	5683	3900	3891	3785	3607	4890	5098	6205
6144	6218	6213	4090	3921	3811	4022	5108	5237	6924
6656	6423	6557	4154	4020	3927	4746	5348	5350	7645

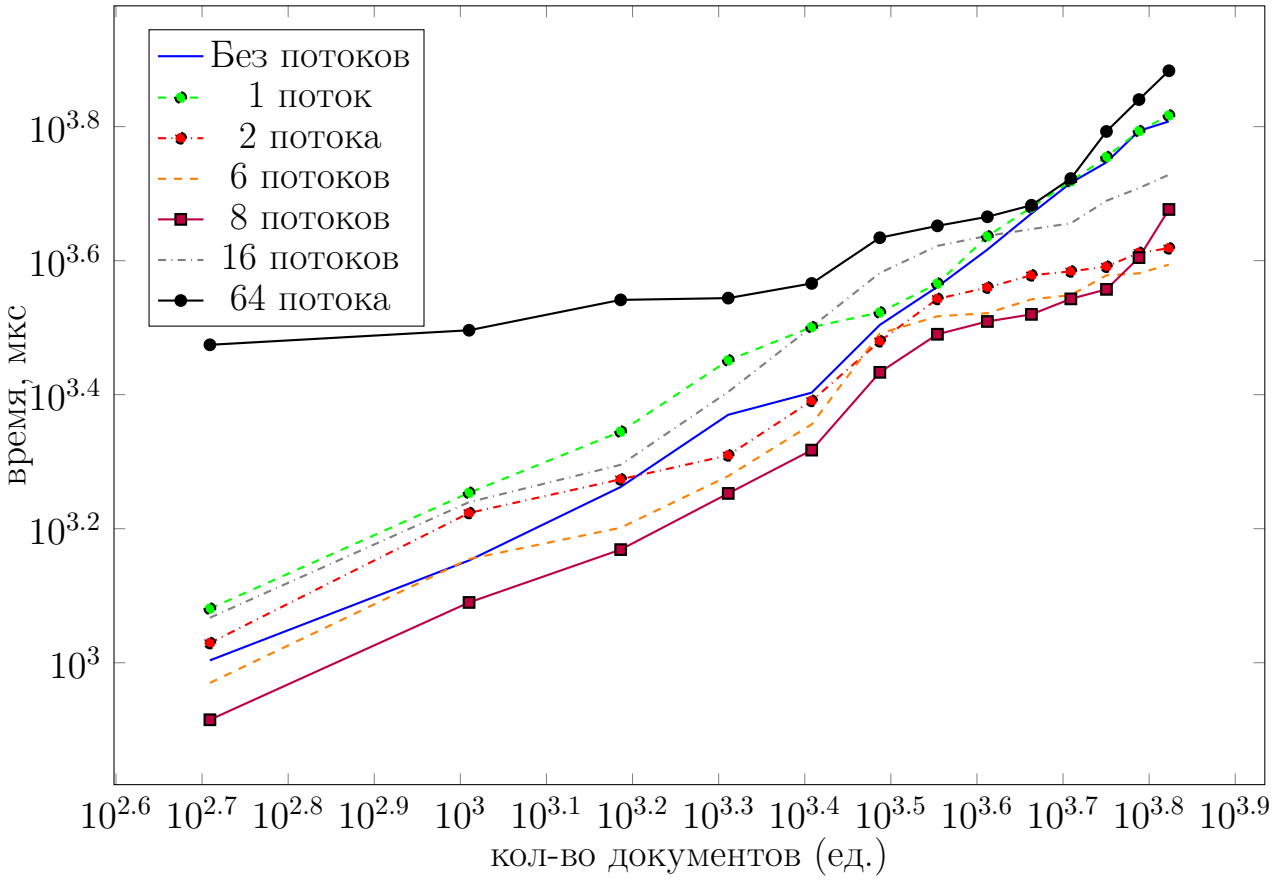

Рисунок 4.1 – Результаты замеров времени работы реализаций алгоритма с разным количеством потоков в зависимости от количества документов

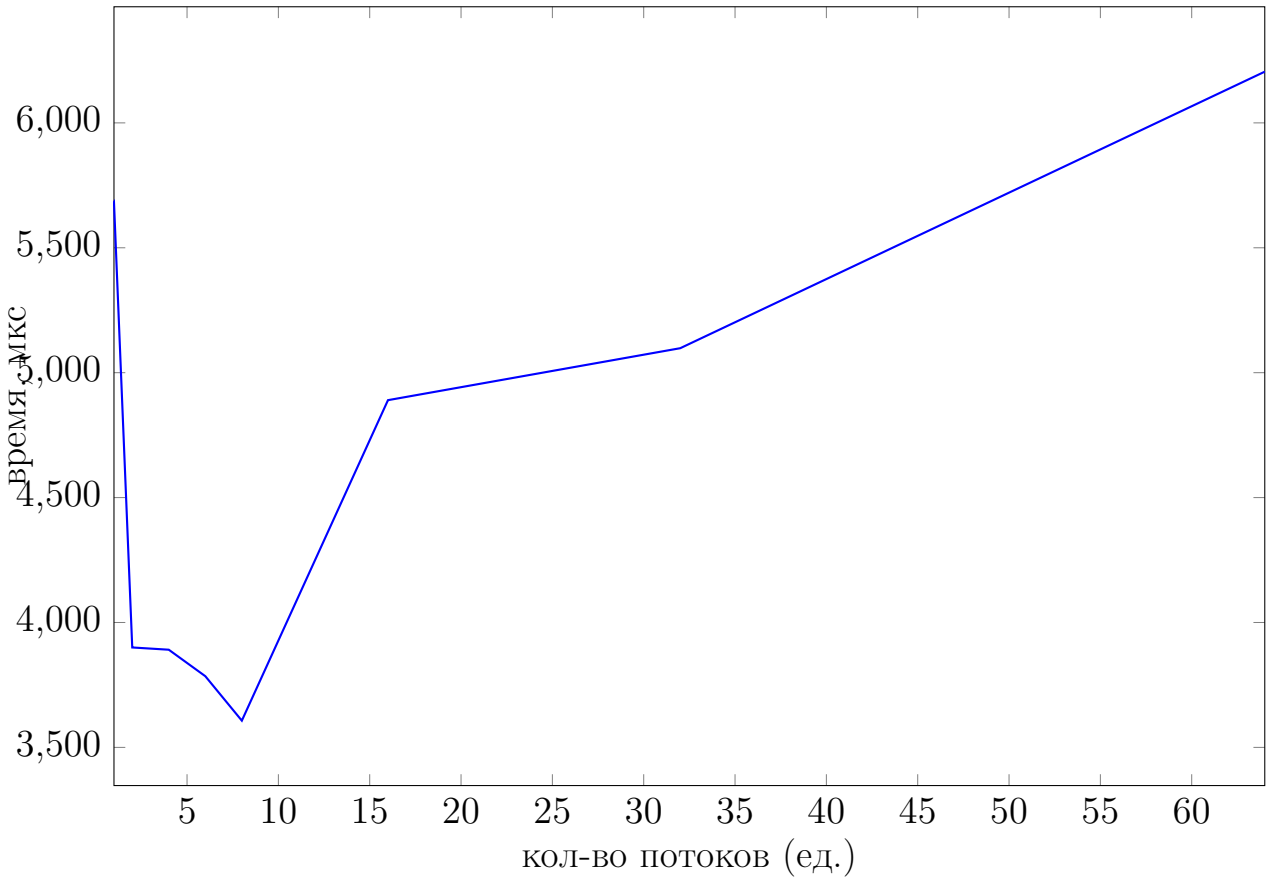

Рисунок 4.2 – Результаты замеров времени работы реализации алгоритма для 5632 документов в зависимости от количества потоков

Из полученных результатов можно сделать вывод, что однопоточный процесс работает быстрее процесса, создающего один отдельный поток для обработки всех документов. Это связано с дополнительными временными затратами на создание потока и передачи ему необходимых аргументов. Наилучший результат по времени для всех значений количества документов показал процесс с 8 дополнительными потоками, выполняющими вычисления. Рекомендуется использовать на данной архитектуре ЭВМ число дополнительных потоков равное числу логических ядер устройства. Для числа потоков, большего 8, затраты на содержание потоков превышают преимущество от использования многопоточности, и функция времени от количества потоков начинает расти

4.3 Вывод

В результате экспериментов было выявлено, что использование многопоточности может уменьшить время выполнения реализации алгоритма по сравнению с однопоточной программой при условии использования подходящего количества потоков.

Выборка из результатов замеров времени (для 5632 документов):

— однопоточный процесс — 5580 мкс;

— один дополнительный поток, выполняющий все вычисления — 5683 мкс;

— 8 потоков (лучший результат) — 3607 мкс, что в 1,57 раз быстрее выполнения однопоточного процесса;

— 64 потока (худший результат) — 6205 мкс, что в 0,89 раз медленнее выполнения однопоточного процесс.

Таким образом, использование дополнительных потоков может как ускорить выполнение процесса по сравнению с однопоточным процессом (в 1,57 раз для 8 потоков), так и увеличить время выполнения (в 0,89 раз для 64 потоков). Рекомендуется использовать на данной архитектуре ЭВМ число дополнительных потоков равное числу логических ядер устройства.

Заключение

В ходе выполнения лабораторной работы было выявлено, что в результате использования многопоточной реализации время выполнения процессам может как улучшиться, так и ухудшиться в зависимости от количества используемых потоков.

При слишком большом значении потоков (более 8 для устройства, на котором проводилось тестирование) затраты на содержание потоков превышают преимущество от использования многопоточности и время выполнения по сравнению с лучшим результатом (для 8 потоков) растут.

Цель, поставленная в начале работы, была достигнута. Кроме того были достигнуты все поставленные задачи.

1) Изучены основы распараллеливания вычислений.

2) Разработано программное обеспечение, которое реализует однопоточный алгоритм вычисления термина частоты для классификации.

3) Разработана и реализована многопоточная версия данного алгоритма.

4) Определены средства программной реализации.

5) Выполнены замеры процессорного времени работы реализаций алгоритма.

6) Проведен сравнительный анализ по времени работы реализаций алгоритмов.

Список использованных источников

1 Stoltzfus J. Multithreading. – Techopedia - Janalta Interactive Inc. [Электронный ресурс], 2022. URL: Режим доступа: https://www.techopedia.com/definition/24297/multithreading-computer-architecture (дата обращения: 28.01.2023).

2 У. Ричард Стивенс Стивен А. Раго. UNIX. Профессиональное программирование. 3-е издание. – СПб.: Питер, 2018. с. 994.

3 Большакова Е.И. Клышинский Э.С. Ландэ Д.В. Носков А.А. Пескова О.В. Ягунова Е.В. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика : учеб. пособие . – М.: МИЭМ, 2013. с. 272.

4 Боярский К. К. Введение в компьютерную лингвистику. Учебное пособие. – СПб.: НИУ ИТМО, 2013. с. 72.

5 C language [Электронный ресурс]. Режим доступа:https://en.cppreference.com/w/c/language (дата обращения: 28.01.2023).

6 pthread.h - threads [Электронный ресурс]. Режим доступа: https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread.h.html (дата обращения: 28.01.2023).

7 Intel [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/201839/intel-core-i510300h-processor-8m-cache-up-to-4-50-ghz.html (дата обращения: 25.09.2022).

8 Windows 10 Pro 2h21 64-bit [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 25.09.2022).

9 Определение текущего времени с высокой точностью [Электронный ресурс]. Режим доступа: http://all-ht.ru/inf/prog/c/func/clock_gettime.html (дата обращения: 28.01.2023).