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 "Анализ алгоритмов"

Тема Параллельные вычисления на основе нативных потоков

Студент Шавиш Тарек

Группа ИУ7И-54Б

Оценка (баллы) _____

Преподаватели Волкова Л. Л.

Москва — 2024 г.

Содержание

Введение

Целью данной лабораторной работы является исследование возможностей использования нативных потоков для оптимизации процессов обаратоки веб-страниц. Нативные потоки, реализуемые предоставляют инструмент для параллельных вычислений, позволяющий значительно повысить эффективность выполнения многозадачных операций.

1 Аналитическая часть

1.1 Цель и задачи

Цель данной работы заключается в демонстрации преимуществ параллельной обработки данных при обработке веб-страниц с использованием нативных потоков.

Основные задачи:

1. Изучить принципы работы и особенности применения нативных потоков в операционных системах.
2. Разработать алгоритмы для последовательного и параллельного обработки данных.
3. Сравнить производительность последовательного и параллельного обработки веб-страниц на примере выбранных интернет-ресурсов.
4. Оценить возможные улучшения производительности и эффективности при использовании многопоточности.

1.2 Теоретическая основа

Обработка веб-страниц это процесс автоматического извлечения данных с них. Использование нативных потоков позволяет обрабатывать несколько страниц одновременно, что существенно повышает скорость работы.

Нативные потоки — это потоки, управляемые операционной системой, которые позволяют выполнять несколько задач одновременно. Это достигается за счет распределения задач по ядрам процессора.

1.3 Вывод

Были подставлены задачи и изучены теоретические основы для выполнения нужных алгоритмов для обработки веб-страниц.

2 Конструкторская часть

В данной главе представлены основные конструкторские решения, использованные в процессе разработки программного обеспечения. Раздел включает требования к программному обеспечению, определение типов данных, а также структуру проекта.

2.1 Требования к реализации программного обеспечения

Для успешной реализации и функционирования разработанного программного обеспечения были установлены следующие требования:

- Использование многопоточности для обеспечения параллельной обработки данных.
- Предоставление времени выполнения каждого режима обработки данных веб-страниц для оценки их производительности.
- Поддержка последовательного и параллельного режимов обработки данных веб-страниц.
- Возможность вывода результатов тестирования.

2.2 Описание типов данных и классов

Выделены следующие классы:

- `WebPageDownloader` - сущность, содержашая логику скачиваний исходный код веб-страницы для его дальнейшей обработки.
- `HTMLParser` - сущность, содержащая логику анализа и извлечения данных веб-страницы.
- `WebScraper` - сущность, содержащая логику обработки веб-страницы.

- `TimeMeasurer` - сущность, содержащая логику для проведения замеров времени обработки страниц.

Используются следующие типы данных:

- `TaskType` - для описания типа задачи при обработки ссылок веб-страниц.
- `Task` - для описания узла в очереди задач при многопоточном режиме обработки страниц.
- `WebScraperConfig` - для описания конфигурации при создании объектов класса WebScraper.

2.3 Структура проекта

Программа разделена на заголовочные файлы (.h), исходные файлы (.cpp) и файл сборки (Makefile). Ниже представлена структура каталогов и файлов проекта:

- **include** - каталог для заголовочных файлов
  - `webpagedownloader.h` - интерфейс класса *WebPageDownloader*.
  - `htmlparser.h` - интерфейс класса *HTMLParser*.
  - `webscraper.h` - интерфейс класса *WebScraper*.
  - `timemeasurer.h` - интерфейс класса *TimeMeasurer*.
- **source** - каталог с исходным кодом и реализацией методов классов.
  - `main.cpp` - главный файл программы, содержащий точку входа программы.
  - `webpagedownloader.cpp` - реализация методов класса *WebPageDownloa*
  - `htmlparser.cpp` - реализация методов класса *HTMLParser*.
  - `webscraper.cpp` - реализация методов класса *WebScraper*.
  - `timemeasurer.cpp` - реализация методов класса *TimeMeasurer*.
- **makefile** - файл для сборки проекта с использованием утилиты make.

- **plot.py** - логика получения аналитических графиков времени выполнения процессов обработки страниц.

Вывод

Были представлены конструкторские решения, использованные при создании программного обеспечения для обработки веб-страниц.

3 Технологическая часть

В данном разделе описывается выбор инструментов для реализации программы, включая язык программирования. Также представлены реализации ключевых алгоритмов исследования, описания методов тестирования программы и анализ полученных результатов.

3.1 Выбор языка программирования

Для реализации алгоритмов вычисления редакционного расстояния был выбран язык программирования C++. Выбор обусловлен следующими факторами:

- Производительность
- Объектно-ориентированный подход
- Стандартная библиотека

3.2 Реализация классов

Интерфейсы классов `WebPageDownloader`, `HTMLParser`, и `WebScraper` представлены в приложенных листингах.

Листинг 3.1 – Интерфейс класса WebPageDownloader

```
1 class WebPageDownloader
2 {
3     public:
4     WebPageDownloader() = default;
5
6     static size_t WriteCallback(void *contents, size_t size, size_t
        nmemb, std::string *userp);
7
8     std::string fetchContent(const std::string &url);
9
```

```
10      void fetchAndSaveContent(const std::string &url, const
            std::string &filename);
11
12      private:
13      std::string convertEncoding(const std::string &input, const
            std::string &from_enc, const std::string &to_enc);
14  };
```

Листинг 3.2 – Интерфейс класса HTMLParser

```
1   class HTMLParser
2   {
3       public:
4       HTMLParser()
5       {
6           myhtml = myhtml_create();
7           myhtml_init(myhtml, MyHTML_OPTIONS_DEFAULT, 1, 0);
8           tree = myhtml_tree_create();
9           myhtml_tree_init(tree, myhtml);
10      }
11
12      ~HTMLParser()
13      {
14          myhtml_tree_destroy(tree);
15          myhtml_destroy(myhtml);
16      }
17
18      void parseHTML(const std::string &html);
19
20      std::vector<std::string> extractCategoryURLs(void);
21
22      std::vector<std::string> extractRecipes(const std::string
            &className);
23
24      std::vector<std::string> extractRecipeIngredients(const
            std::string &className);
25
26      void replaceFileExtension(std::string &fileExtension);
27
28      std::vector<std::string> formatIngredients(const
            std::vector<std::string> &rawIngredients);
29
```

```
30      private:
31      myhtml_t* myhtml;
32      myhtml_tree_t* tree;
33
34      std::vector<std::string> findRecipesInNode(myhtml_tree_node_t*
            node);
35
36      std::string collectNodeText(myhtml_tree_node_t* node);
37
38  };
```

Листинг 3.3 – Интерфейс класса WebScraper

```
1  enum class TaskType
2  {
3      FetchRecipes,
4      SaveRecipeDetails
5  };
6
7  struct Task
8  {
9      std::string url;
10     TaskType type;
11     std::string directory;
12 };
13
14 struct WebScraperConfig
15 {
16     std::string url;
17     int maxSectionN = -1;
18     int maxLinksPerSectionN = -1;
19 };
20
21 class WebScraper
22 {
23     public:
24     WebScraper(const WebScraperConfig &config)
25     : homePageUrl(config.url),
26     maxSectionN(config.maxSectionN),
27     isSectionLimitSet(config.maxSectionN >= 0),
28     maxLinksPerSectionN(config.maxLinksPerSectionN),
29     isLinksLimitSet(config.maxLinksPerSectionN >= 0) {}
```

```
30
31      void addSectionUrl(const std::string &url);
32
33      void addSectionUrls(const std::vector<std::string>
          &sectionUrls);
34
35      std::string getHomeUrl(void) const;
36
37      std::vector<std::string> getBaseSectionUrls(void) const;
38
39      std::unordered_map<std::string, std::vector<std::string>>
          getSectionRecipes(void) const;
40
41      void mapSectionRecipes(void);
42
43      void saveRecipe(const std::string &url, const std::string
          &directory);
44
45      void seriesScraping(void);
46
47      void parallelScraping(int threadsN);
48
49      void filterRecipeLinks(std::vector<std::string> &recipes);
50
51      friend std::ostream &operator<<(std::ostream &os, const
          WebScraper &scraper);
52
53      private:
54      HTMLParser parser;
55      WebPageDownloader downloader;
56
57      std::string homePageUrl;
58      std::unordered_map<std::string, std::vector<std::string>>
          baseSections;
59
60      int maxSectionN;
61      bool isSectionLimitSet;
62
63      int maxLinksPerSectionN;
64      bool isLinksLimitSet;
65
```

```
66      std::vector<std::string> splitUrl(const std::string &url);
67  };
```

Вывод

Разработанное программное обеспечение успешно справляется с задачами обработки данных веб-страниц, обеспечивая высокую производительность за счет возвможности использования параллельной обработки данных.

4 Исследовательская часть

TODO

4.1 Время выполнения алгоритмов

Для получения наиболее точных результатов время выполнения обработки данных каждый алгоритм был запущен 15 раз, после чего полученные результаты были усреднены.

Таблица 4.1 – Результаты замеров времени

Количество потоков	Время (в миллисекундах)
1	748670
2	349475
4	172167
8	104511
16	74627
32	61236
64	55549

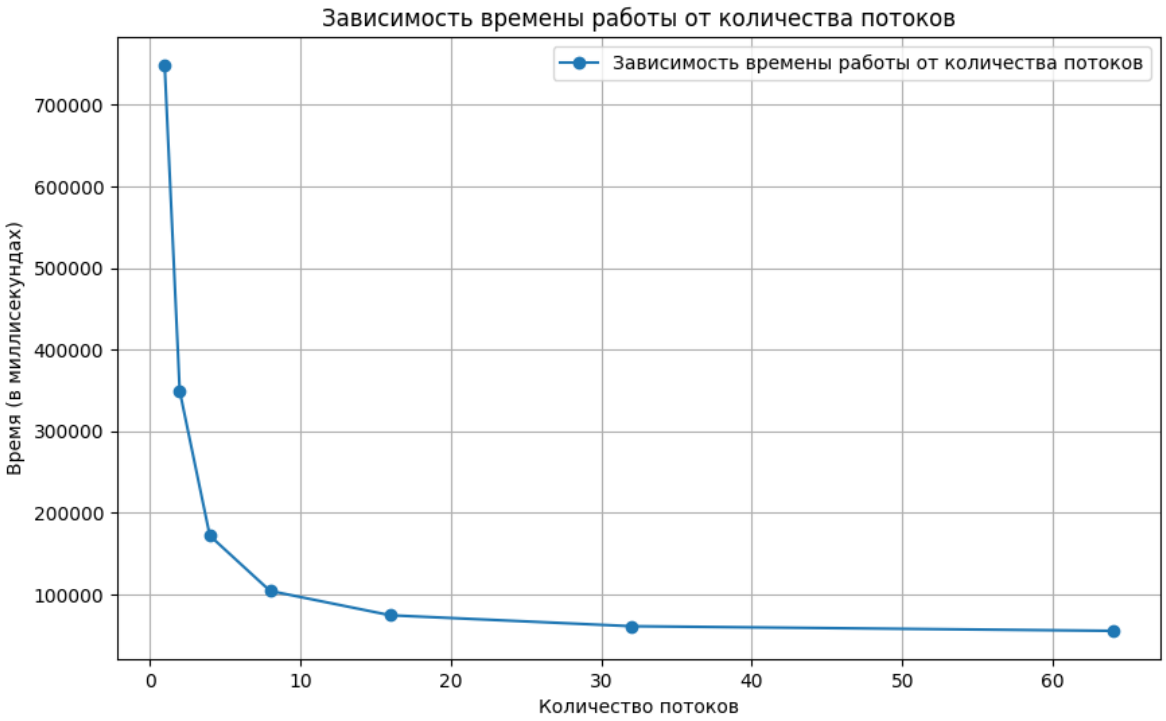

Рисунок 4.1 – Зависимость времени обработки данных веб-страниц от количества потоков

4.2 Технические характеристики устройства

Настройки устройства, на котором проводились замеры времени и разработка программы, приведены ниже:

- Операционная система: Ubuntu 22.04 LTS
- Процессор: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz
- Оперативная память: 16 Гб.

4.3 Вывод

Исходя из результатов исследования, можно наблюдать значительное уменьшение времени обработки данных с увеличением количества используемых потоков. Это подтверждает эффективность применения параллельных вычислений.

Заключение

В рамках данной работы была разработана и реализована система обработки данных веб-страниц, использующая параллельные вычисления на основе нативных потоков. Разработанное программное обеспечение демонстрирует значительные преимущества в скорости обработки данных по сравнению с последовательным подходом.

Основные результаты работы:

- Были исследованы и описаны основные принципы работы с нативными потоками в контексте операционных систем и их применение в параллельных вычислениях.
- Разработаны и реализованы компоненты системы, включая модули для загрузки веб-страниц, анализа HTML-кода и многопоточной обработки данных.
- Проведено сравнение производительности последовательной и параллельной обработки данных, результаты которого подтвердили высокую эффективность многопоточной обработки.
- Система была протестирована на реальных данных.

Список использованной литературы

[1] UNIX Профессиональное программирование / У. Ричард Стивенс, Стивен А. Раго - ИМВО 2007

[2] Официальная документация дистрибутива Linux - Ubuntu [Электронный ресурс]. - URL: `https://assets.ubuntu.com/v1/544d9904-ubuntu-server-guide-2024-01-22.pdf?` (дата обращения: 2.11.2024)

[3] Официальная документация библиотеки Matplotlib [Электронный ресурс]. - URL: `https://matplotlib.org/stable/index.html` (дата обращения: 2.11.2024)

[4] Справочник ЯП C++ [Электронный ресурс]. - URL: `https://learn.microsoft.com/en-us/cpp/?view=msvc-170` (дата обращения: 2.11.2024)

[5] Процессор Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz [Электронный ресурс]. - URL: `https://www.intel.com/content/www/us/en/products/sku/196603/intel-core-i51035g1-processor-6m-cache-up-to-3-60-ghz/specifications.html` (дата обращения: 2.11.2024)