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»

Отчёт по лабораторной работе №4
по курсу «Анализ алгоритмов»

Тема Параллельные вычисления на основе нативных потоков

Студент Талышева О.Н.

Группа ИУ7-55Б

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных вычислительных системах параллельные вычисления играют важную роль в повышении производительности программного обеспечения. Одним из основных методов организации параллелизма является использование нативных потоков операционной системы (OS threads), которые позволяют распределять выполнение задач между несколькими ядрами процессора.

Цель работы: исследовать особенности параллельных вычислений на основе нативных потоков и время работы программы на их разном количестве.

Для достижения этой цели были поставлены следующие задачи:

1. анализ предметной области;
2. разработка алгоритма обработки данных;
3. создание ПО реализующего разработанный алгоритм;
4. исследование характеристик созданного ПО.

1 Входные и выходные данные

Входными данными для ПО является адрес главной страницы ресурса. Выходные данные — директория с файлами, которые содержат скачанные данные со страниц в формате, пригодном для дальнейшей обработки (html), а также время работы.

2 Преобразование входных данных в выходные

Программа считывает URL-адреса из файла, загружает содержимое страниц и сохраняет их локально как файлы. Для каждого URL создаётся поток, который обрабатывает загрузку и запись данных. Имена файлов формируются на основе URL с заменой недопустимых символов.

3 Примеры работы программы

На рисунке 1 представлен пример работы программы: запуск замеров времени на разном количестве потоков и отдельно на 5 потоках для 100 ссылок с сайта kedem.ru.

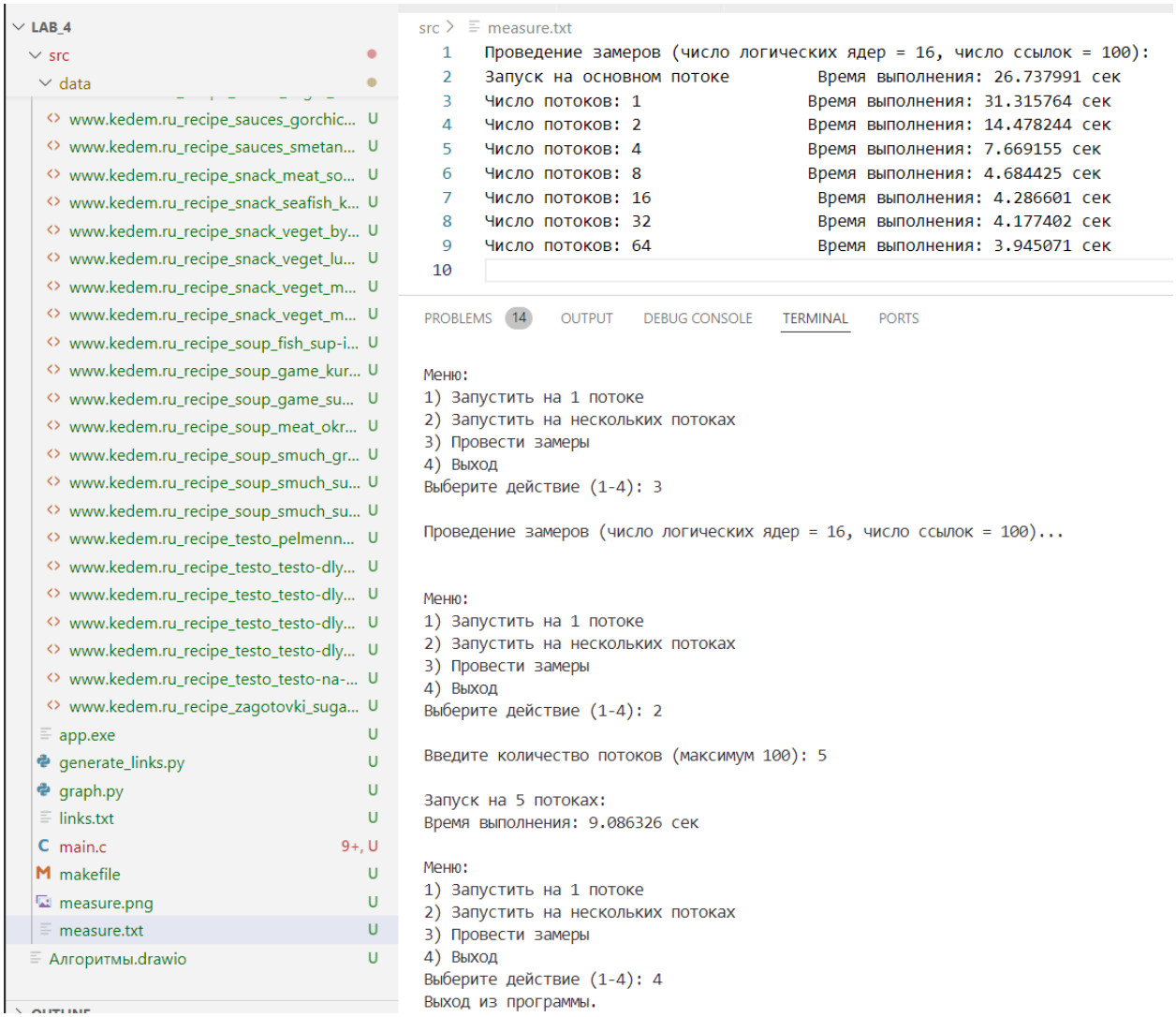

Рисунок 1 – Пример работы программы.

4 Тестирование

Выполнено тестирование реализованной основной части программы по методологии чёрного ящика. В таблице 1 представлено описание тестов. Все тесты пройдены успешно.

Таблица 1 – Функциональные тесты

№ теста	Входные данные	Ожидаемые выходные данные	Успешность
1	Файл ‘links.txt‘ не существует	Программа выводит ошибку: "Не удалось открыть файл"	Ошибка (1)
2	Файл ‘links.txt‘ пуст	Программа выводит: "Нет доступных URL для обработки."	Ошибка (2)
3	Файл ‘links.txt‘ содержит 1 URL	Программа успешно загружает страницу на 1 потоке и выводит время загрузки	Успех
4	Файл ‘links.txt‘ содержит 1000 URL	Программа корректно загружает все страницы в многопоточном режиме	Успех
5	Недопустимые символы в URL	Программа корректно заменяет недопустимые символы в именах файлов	Успех
6	Ввод числа потоков больше числа URL	Программа выводит ошибку и запрашивает корректное число потоков	Ошибка
7	Ввод несуществующего пункта меню	Программа выводит: "Неверный пункт меню. Попробуйте снова."	Ошибка
8	Файл уже обработан, повторный запуск загрузки	Программа корректно завершает работу без повторной загрузки	Успех
9	URL превышает максимальную длину	Программа корректно обрабатывает или игнорирует длинный URL	Успех
10	Папка ‘data‘ уже существует	Программа сохраняет страницы в существующую папку без ошибок	Успех

Файл ‘links.txt‘ со списком ссылок для обработки.

Данные выводятся в папку ‘data‘.

5 Описание исследования

Было выполнено исследование зависимости производительности разработанного ПО (в терминах количества обработанных страниц в единицу времени) от количества дополнительных потоков. Количество дополнительных потоков изменялось от 0 (вычисление в основном потоке), до $4 \cdot 16 = 64$, где 16 — количество логических ядер используемой ЭВМ (см листинг 1 и график 2).

```
Taking measurements (number of logical cores = 16, number of links = 100):
Running on the main thread Execution time: 06/290003 sec
Number of threads: 1 Execution time: 30.503609 sec
Number of threads: 2 Execution time: 13.407696 sec
Number of threads: 4 Execution time: 7.321394 sec
Number of threads: 8 Execution time: 4.626752 sec
Number of threads: 16 Execution time: 3.882306 sec
Number of threads: 32 Execution time: 4.112705 sec
Number of threads: 64 Execution time: 3.792753 sec
```

Листинг 1 – Результаты замеров времени на разном количестве потоков

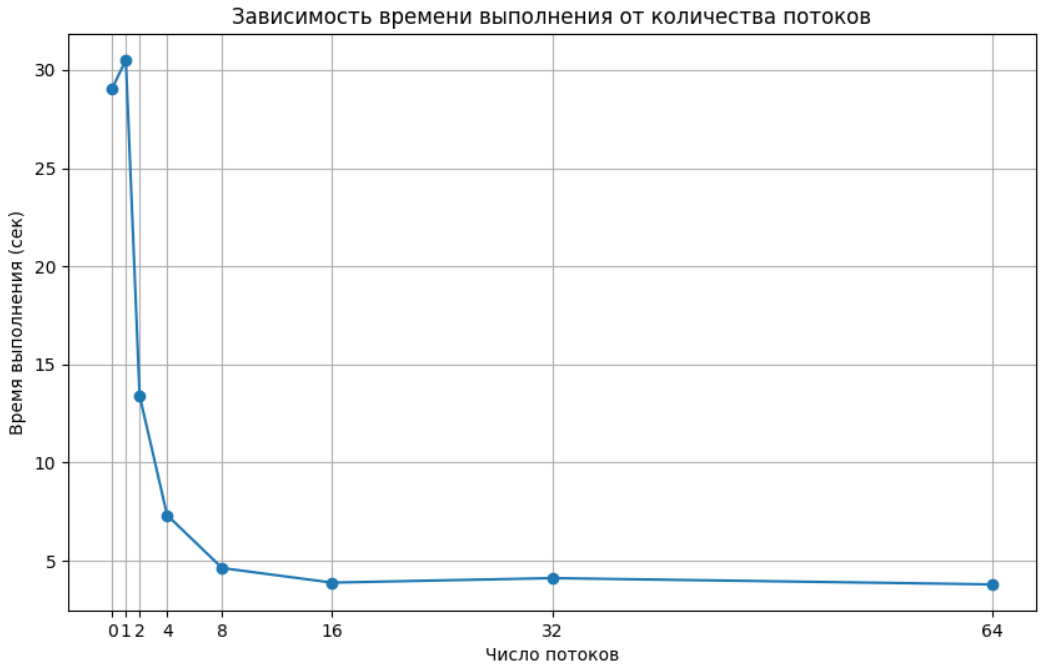

Рисунок 2 – График замеров времени на разном количестве потоков

Анализ производительности

На основе представленных данных о времени выполнения загрузки 100 страниц при разном количестве потоков сформулируем несколько выводов:

Последовательная загрузка

1. Запуск на основном потоке занял 29.06 секунд.
2. При использовании 1 потока время увеличилось до 30.50 секунд. Это может указывать на дополнительные накладные расходы на создание потоков и их управление, которые перекрывают выгоду от работы при одном потоке.

Увеличение числа потоков

Время выполнения значительно сокращается при увеличении числа потоков до 16. Например:

1. 2 потока – 13.41 секунд (почти вдвое быстрее, чем один поток).
2. 4 потока – 7.32 секунд.
3. 8 потоков – 4.63 секунд.
4. 16 потоков – 3.88 секунд.

Это демонстрирует эффективное масштабирование производительности, особенно при увеличении числа потоков до числа логических ядер процессора (16).

Сверх числа логических ядер

При увеличении числа потоков до 32 и 64 наблюдается незначительное ухудшение производительности:

1. 32 потока – 4.11 секунд.
2. 64 потока – 3.79 секунд.

Это явление можно объяснить накладными расходами на планирование потоков и синхронизацию между ними. Увеличение числа потоков сверх числа логических ядер (16) не даёт значительного прироста и даже начинает снижать эффективность [1].

Выводы

1. Оптимальное число потоков для данного теста – около 16, что соответствует числу логических ядер процессора. Увеличение числа потоков свыше этого значения не приводит к улучшению производительности.
2. В случае с малым числом потоков (1 и 2), многопоточность даёт значительный прирост по сравнению с последовательным запуском, однако использование одного потока может даже замедлять выполнение из-за накладных расходов на многопоточность.

3. Важно учитывать баланс между количеством потоков и ресурсами системы, чтобы избежать чрезмерной нагрузки на планировщик и неэффективного использования ядер.

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы были исследованы особенности параллельных вычислений на основе нативных потоков и время работы программы на их разном количестве.

В частности:

1. был проведён анализ предметной области;
2. разработан алгоритм обработки данных;
3. создано ПО реализующее разработанный алгоритм;
4. исследованы характеристики созданного ПО.

В ходе лабораторной работы был рассмотрен, спроектирован и запрограммирован алгоритм загрузки страниц с помощью нативных потоков.

Проведённые замеры времени выполнения программного обеспечения при использовании различного числа потоков показали значительное ускорение обработки данных. Эти результаты подтверждают эффективность подхода к параллельной обработке данных и показывают, что увеличение числа потоков приводит к улучшению производительности, особенно при наличии достаточного количества логических ядер. Наблюдаемое время выполнения на уровне 3.79 секунд при использовании 64 потоков также указывает на наличие эффекта насыщения, что подчёркивает необходимость оптимального выбора количества потоков для достижения максимальной производительности.

Таким образом, лабораторная работа не только обеспечила практические навыки в области работы с нативными потоками, но и подтвердила теоретические знания о преимуществах и ограничениях параллельных вычислений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Silberschatz, A., Galvin, P. B., Gagne, G. Operating System Concepts. 10-е изд. Wiley, 2018.

[2] Методы парсинга данных на C++ и Python. Tetraquark. [Электронный ресурс]. URL: `https://tetraquark.ru/archives/47` (дата обращения: 22.10.2024).

[3] Парсинг HTML и XML в C#. Metanit. [Электронный ресурс]. URL: `https://metanit.com/c/tutorial/11.1.php` (дата обращения: 22.10.2024).

[4] Парсинг на Python с библиотекой Beautiful Soup. PythonRu. [Электронный ресурс]. URL: `https://pythonru.com/biblioteki/parsing-na-python-s-beautiful-soup` (дата обращения: 22.10.2024).

[5] Парсинг с помощью Python. The Code. [Электронный ресурс]. URL: `https://thecode.media/parsing-2/` (дата обращения: 23.10.2024).