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 № 5 по курсу "Анализ алгоритмов"

Тема Организация параллельных вычислений по конвейерному принципу

Студент Шавиш Тарек

Группа ИУ7И-54Б

Оценка (баллы)

Преподаватели Волкова Л. Л.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы является разработка и анализ работы конвейерной системы обработки данных. Конвейерная обработка — это метод, который позволяет распределить различные стадии обработки данных по отдельным потокам исполнения, что способствует параллелизации процессов и увеличению производительности системы.

1 Аналитическая часть

1.1 Цель и задачи

Цель данной работы заключается в разработке многопоточной конвейерной системы обработки данных.

Требуется решить следующие задачи.

1. Описать принципы работы и особенности применения системы обработки данных по конвейерному принципу.

2. Разработать алгоритмы для стадий обработки данных.

1.2 Теоретическая основа

Конвейерная обработка данных — это техника, при которой общий процесс обработки разделяется на несколько последовательных этапов, причем каждый этап способен выполняться параллельно в отдельном потоке. Это разделение позволяет каждому этапу начинать обработку своей части данных как только она становится доступной, не дожидаясь завершения предыдущих этапов. Таким образом, данные обрабатываются на каждой стадии.

1.3 Вывод

Конвейерная обработка данных представляет собой инструмент для оптимизации вычислительных процессов, особенно в задачах, связанных с большим объёмом данных. Использование многопоточности позволяет значительно ускорить обработку за счёт параллельного выполнения различных этапов.

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 на каждой стадии соответствующим обработчиком;

— вывод в реальном времени статуса задачи;

— поддержка параллельного режима обработки данных стадиями конвейера;

— вывод статистику каждой задачи после её обработки конвейерной системой.

2.2 Описание типов данных и классов

Выделены следующие классы:

— `Task` — сущность, описывающая обрабатываемую задачу.

— `StageBase` — абстрактный класс, описывающий стадию обработки данных.

— `TaskGeneratorStage` — сущность, содержащая логику создания задач и их добавление во входную очередь конвейерной системы.

— `ReaderStage` — сущность, содержащая логику стадии чтения данных.

— `ParserStage` — сущность, содержащая логику стадии обработки полученных данных.

— `WriterStage` — сущность, содержащая логику стадии записи данных в базу данных.

— `AccumulatorStage` — сущность, содержащая логику анализа каждой задачи после её обработки на всех стадиях.

2.3 Структура проекта

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

Каталог **include** — каталог для заголовочных файлов:

— `stagebase.h` — интерфейс абстрактного класса *StageBase*;

— `taskgeneratorstage.h` — интерфейс класса *TaskGeneratorStage*;

— `readerstage.h` — интерфейс класса *ReaderStage*;

— `parserstage.h` — интерфейс класса *ParserStage*;

— `writerstage.h` — интерфейс класса *WriterStage*;

— `accumulatorstage.h` — интерфейс класса *AccumulatorStage*.

Каталог **source** — каталог с исходным кодом и реализацией методов классов:

— `main.cpp` — главный файл программы, содержащий точку входа программы;

— `taskgeneratorstage.cpp` — реализация методов класса *TaskGeneratorStage*;

— `readerstage.cpp` — реализация методов класса *ReaderStage*;

— `parserstage.cpp` — реализация методов класса *ParserStage*;

— `writerstage.cpp` — реализация методов класса *WriterStage*;

— `accumulatorstage.cpp` — реализация методов класса *AccumulatorStage*.

Файл **makefile** - файл для сборки проекта с использованием утилиты make. Файл **plot.py** - логика получения аналитических графиков времени выполнения процессов обработки страниц.

2.4 Вывод

Были представлены конструкторские решения, использованные при создании программного обеспечения для создания конвейерной системы.

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;

— наличие стандартной библиотеки шаблонов.

3.2 Реализация классов

Интерфейсы классов `StageBase`, `Task`, `TaskGeneratorStage`, `ReaderStage`, `WriterStage`, и `TaskAccumulatorStage` представлены в приложенных листингах `3.1`, `3.2`, `3.3`, `3.4`, `3.5`, `3.6`, `3.7`.

Листинг 3.1 – Интерфейс класса StageBase

```
1 class StageBase
2 {
3     public:
4     virtual ~StageBase() {}
5
6     virtual void process(TaskQueue& inputQueue, TaskQueue&
        outputQueue) = 0;
7
8     virtual void shutdown() = 0;
9 };
```

Листинг 3.2 – Интерфейс класса Task

```
1  class Task
2  {
3      public:
4
5      static int globalId;
6
7      Task(const std::string &filePath_) : filePath(filePath_),
           creationTime(std::chrono::system_clock::now())
8      {
9          id = ++globalId;
10     }
11
12     int getId() const;
13     const std::string &getFilePath() const;
14     const std::string &getContent() const;
15     const std::string &getUrl() const;
16     const std::string &getTitle() const;
17     const std::vector<Ingredient> &getIngredients() const;
18     const std::vector<std::string> &getInstructions() const;
19     const std::string &getImageUrl() const;
20
21     void setId(int id_);
22     void setFilePath(const std::string &path_);
23     void setContent(const std::string &content_);
24     void setUrl(const std::string &url_);
25     void setTitle(const std::string &title_);
26     void setIngredients(const std::vector<Ingredient>
           &ingredients_);
27     void setInstructions(const std::vector<std::string>
           &instructions_);
28     void setImageUrl(const std::string &imageUrl_);
29     void addIngredient(const Ingredient &ingredient_);
30
31
32     std::chrono::system_clock::time_point getCreationTime() const;
33     std::chrono::system_clock::time_point
           getQueueEntryTime(TaskQueueID queueID) const;
34     std::chrono::system_clock::time_point
           getQueueExitTime(TaskQueueID queueID) const;
35     std::chrono::system_clock::time_point
```

```
        getStageEntryTime(PipelineStage stageID) const;
36      std::chrono::system_clock::time_point
        getStageExitTime(PipelineStage stageID) const;
37      std::chrono::system_clock::time_point getDestructionTime()
        const;
38
39      void markQueueEntry(TaskQueueID queueID);
40      void markQueueExit(TaskQueueID queueID);
41      void markStageEntry(PipelineStage stageID);
42      void markStageExit(PipelineStage stageID);
43      void markDestruction();
44
45      long long getQueueDuration(TaskQueueID queueID) const;
46      long long getStageDuration(PipelineStage stageID) const;
47      long long getTaskLifeTime() const;
48
49      friend std::ostream &operator<<(std::ostream &os, const Task
        &task);
50
51      private:
52      int id;
53      std::string filePath;
54      std::string content;
55      std::string url;
56      std::string title;
57      std::vector<Ingredient> ingredients;
58      std::vector<std::string> instructions;
59      std::string imageUrl;
60
61      struct Timestamp
62      {
63          std::chrono::system_clock::time_point entry;
64          std::chrono::system_clock::time_point exit;
65      };
66
67      std::chrono::system_clock::time_point creationTime;
68      std::chrono::system_clock::time_point destructionTime;
69
70      std::map<TaskQueueID, Timestamp> queueTimes;
71      std::map<PipelineStage, Timestamp> stageTimes;
72  };
```

Листинг 3.3 – Интерфейс класса TaskGeneratorStage

```
1  class TaskGeneratorStage : public StageBase
2  {
3      public:
4
5      TaskGeneratorStage(const std::string& directoryPath_, int
           numThreads_, Logger &logger_)
6      : directoryPath(directoryPath_), numThreads(numThreads_),
           threadsCompleted(0), logger(logger_){}
7
8      void process(TaskQueue& inputQueue, TaskQueue& outputQueue)
           override;
9
10     void shutdown() override;
11
12     private:
13
14     std::string directoryPath;
15     int numThreads;
16     std::atomic<int> threadsCompleted;
17     std::mutex mtx;
18     std::condition_variable cv;
19     Logger &logger;
20
21     void generateTasks(TaskQueue &outputQueue, const
           std::vector<std::string> &paths, int start, int end);
22
23     std::vector<std::string> collectAllFilePaths(const std::string
           &directoryPath);
24 };
```

Листинг 3.4 – Интерфейс класса ReaderStage

```
1  class ReaderStage : public StageBase
2  {
3      public:
4
5      explicit ReaderStage(int numThreads_, Logger &logger_)
6      : numThreads(numThreads_), logger(logger_) {}
7
8      void process(TaskQueue& inputQueue, TaskQueue& outputQueue)
           override;
```

```
9
10      void shutdown() override;
11
12      private:
13
14      int numThreads;
15      Logger &logger;
16
17      void readTasks(TaskQueue &inputQueue, TaskQueue &outputQueue);
18  };
```

Листинг 3.5 – Интерфейс класса ParserStage

```
1  class ParserStage : public StageBase
2  {
3      public:
4
5      explicit ParserStage(int numThreads_, Logger &logger_)
6      : numThreads(numThreads_), logger(logger_){}
7
8      void process(TaskQueue& inputQueue, TaskQueue& outputQueue)
           override;
9
10     void shutdown() override;
11
12     private:
13
14     int numThreads;
15     Logger &logger;
16
17     void parseTasks(TaskQueue& inputQueue, TaskQueue& outputQueue);
18
19     void parseTask(Task &task);
20
21     Ingredient parseIngredient(const std::string &line);
22 };
```

Листинг 3.6 – Интерфейс класса WriterStage

```
1
2  class WriterStage : public StageBase
```

```
3 {
4     public:
5
6     WriterStage(const std::string &dbConnStr_, int numThreads_,
          Logger &logger_)
7     : dbConnStr(dbConnStr_), numThreads(numThreads_),
          logger(logger_) {}
8
9     void process(TaskQueue& inputQueue, TaskQueue& outputQueue)
          override;
10
11    void shutdown() override;
12
13    private:
14    std::string dbConnStr;
15    int numThreads;
16    Logger &logger;
17
18    void writeTasks(TaskQueue& inputQueue, TaskQueue& outputQueue);
19 };
```

Листинг 3.7 – Интерфейс класса TaskAccumulatorStage

```
1
2 class AccumulatorStage : public StageBase
3 {
4     public:
5
6     AccumulatorStage(int numThreads_, Logger &logger_)
7     : numThreads(numThreads_), logger(logger_) {}
8
9     void process(TaskQueue& inputQueue, TaskQueue& outputQueue)
          override;
10
11    void shutdown() override;
12
13    private:
14
15    int numThreads;
16    Logger &logger;
17
```

```
18      void accumulateTasks(TaskQueue& inputQueue, TaskQueue&
            outputQueue);
19  };
```

3.3 Тестирование

Программное обеспечение, реализующее конвейерной системы обработки данных было запущено с реальными данными. Логирование текущего статуса каждой задачи позволило проверять правильность работы системы.

3.4 Вывод

Разработанное программное обеспечение успешно справляется с обработкой данных по конвейерному принципу.

4 Исследовательская часть

В данном разделе будут анализированы данные, полученные при тестировании системы обработки данных по конвейерному принципу.

4.1 Анализ времени обработки задач конвейерной системы

Данные получены при обработке 150 задач.

— Среднее время существования задачи: 63002 микросекунды.

— Среднее время ожидания задачи в очереди для чтения данных: 3143 микросекунды.

— Среднее время ожидания задачи в очереди для обработки данных: 41 микросекунды.

— Среднее время ожидания задачи в очереди для записи данных: 55437 микросекунды.

— Среднее время обработки задачи на стадии чтения данных: 226 микросекунды.

— Среднее время обработки задачи на стадии обработки данных: 79 микросекунды.

— Среднее время обработки задачи на стадии записи данных: 3711 микросекунды.

4.2 Технические характеристики устройства

Настройки устройства, на котором проводилась разработка программы приведены ниже.

— Операционная система: Ubuntu 22.04 LTS.

— Процессор: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00 ГГц.

— Оперативная память: 16 Гб.

4.3 Вывод

Анализ времени обработки задач в конвейерной системе показывает значительные различия во времени ожидания задач на разных стадиях. Заметно, что среднее время обработки задачи в стадии записи больше всех, так как происходит обращение к базе данных, именно поэтому следует, что время ожидания задач для их записи также больше чем в остальных очередях.

ЗАКЛЮЧЕНИЕ

Цель достигнута. Все задачи были решены:

— были описаны принципы конвейерной обработки данных и многопоточности;

— разработаны алгоритмы для каждой стадии обработки данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. UNIX Профессиональное программирование / У. Ричард Стивенс, Стивен А. Раго. - СПб - Москва: ИМВО, 2007. — 1035 с.

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 языка программирования 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 @ 1.00 ГГц [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)