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»

Лабораторная работа №5
по дисциплине «Анализ алгоритмов»

Тема Организация параллельных вычислений по конвейерному принципу

Студент Звягин Д.О.
Группа ИУ7-53Б
Преподаватель Волкова Л.Л., Строганов Д.В.

Москва, 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Параллелизм — это возможность выполнения нескольких процессов одновременно [1].

С точки зрения программирования, параллелизм — это возможность систем производить вычисления одновременно.

В тех ситуациях, где это возможно и обоснованно, использование параллельных вычислений может привести к улучшению временных характеристик программ.

Такими случаями могут быть, например, задачи в которых нужно произвести сложную обработку нескольких независимых сущностей, или задачи, где возможно производить вычисления в ожидании ввода/вывода для других элементов программы [2].

В современных системах параллельные вычисления разделяют на два типа [2]:

— потоковая параллельность;

— параллельность, основанная на процессах.

Целью данной работы является разработка программного обеспечения, обрабатывающего веб-страницы онлайн ресурса chiefs.kz.

Для достижения этой цели нужно выполнить следующие задачи:

— рассмотреть структуру страниц с рецептами ресурса chiefs.kz;

— разработать ПО, выполняющее обработку веб-страницы и запись данных в БД;

— разработать ПО, выполняющее эту задачу методом конвейерной обработки;

— реализовать разработанное ПО;

— проанализировать полученную реализацию по временным характеристикам.

1 Требования к разрабатываемому ПО

К разрабатываемому ПО предъявляется несколько требований:

— на вход подаются html-файлы веб-страниц с рецептами с ресурс http://chiefs.kz/;

— на выход подаётся файл базы данных с обработанными данными входных страниц.

На момент написания программы для лабораторной работы (7 декабря 2024г.) ((А теперь и на момент переписывания для правильной обработки Русской кодировки — 15 февраля 2025 г.)), веб-ресурс http://chiefs.kz/ выглядит, как сайт с содержимым-заглушкой. На нём доступно всего 4 рецепта, поэтому дополнительных программ для получения списка рецептов не требуется.

Для замеров временных характеристик, ссылки будут использованы повторно, а содержимое страниц будет загружаться заново.

2 Разработка ПО

Для реализации алгоритмов был выбран язык программирования c++ стандарта 14882 (c++20) [3]. Этот язык достаточен для выполнения работы, так как его стандартная библиотека предлагает средства для работы с нативными потоками.

Для работы с базами данных была использована библиотека SQLite3 [4]. Она предоставляет такие функции для работы с базой данных, как:

— создание БД;

— создание таблиц;

— поддержка SQL запросов;

Для извлечения элементов с html страниц была использована библиотека Gumbo [5]. Эта библиотека предоставляет возможность поиска определённых html-элементов с учётом различных параметров, а также даёт возможность извлечения данных с них.

Принцип конвейерной обработки был реализован с использованием стандартных функций для работы с потоками языка c++, однако для поддержания потоконезависимой очереди была использована библиотека OneTBB [6], а конкретно — её модуль «concurrent_queue»

Структуры данных, использующиеся для конвейерной обработки:

Структура задачи представлена в листинге 2.1.

Листинг 2.1 — Структура задачи

```
class Task {
    public:
      int id;
      int stage;
      std::chrono::time_point<std::chrono::high_resolution_clock>
         timestamp;
      std::vector<std::chrono::microseconds> times;

      std::string filename;
      std::string data;

      std::string title;
      std::string image;
      std::string ingredients;
      std::vector<std::string> steps;

    public:
      Task() {}
      Task(int id, const std::string &filename) : id(id), filename(
         filename) {
```

```
        timestamp = std::chrono::high_resolution_clock::now();
    }

    std::chrono::microseconds nextStage() {
        std::chrono::time_point<std::chrono::high_resolution_clock> t2 =
            std::chrono::high_resolution_clock::now();

        times.push_back(
            std::chrono::duration_cast<std::chrono::microseconds>(t2 -
                timestamp));

        stage++;

        timestamp = std::chrono::high_resolution_clock::now();

        return times.back();
    }
};
```

Класс, управляющий задачами представлен в листинге 2.2.

Листинг 2.2 — Класс, управляющий задачами

```
class TaskManager {
private:
  tbb::concurrent_queue<Task> stage1Queue;
  tbb::concurrent_queue<Task> stage2Queue;
  tbb::concurrent_queue<Task> stage3Queue;
  tbb::concurrent_vector<Task> finished;
  std::atomic<int> taskCounter{0};

  std::atomic<bool> running = false;

  Logger logger;
  DatabaseManager db;

  void logTask(const Task &task, const std::string &message,
               const std::chrono::microseconds &time);

  void readFileStage();
  void processDataStage();
  void writeToDBStage();
```

```
public:
  TaskManager();
  void run();
  void stop();
  void push(Task t);
  const tbb::concurrent_vector<Task> &getFinished() { return finished; }
};
```

Задачи попадают в очередь первой стадии при использовании функции push у управляющего задачами.

Обработка начинается при использовании функции run и заканчивается при использовании функции stop (однако при использовании stop процесс будет заблокирован до завершения обработки всех процессов, находящихся в очереди).

Реализация функции run и функций каждого этапа обработки представлены в листинге 2.3.

Листинг 2.3 — Реализация функции run и функций каждого этапа обработки

```
void TaskManager::run() {
    running = true;
    std::thread reader(&TaskManager::readFileStage, this);
    std::thread processor(&TaskManager::processDataStage, this);
    std::thread writer(&TaskManager::writeToDBStage, this);
}

void TaskManager::readFileStage() {
  while (running || taskCounter) {
    Task task;
    if (stage1Queue.try_pop(task)) {
      auto duration = task.nextStage();
      logTask(task, "started step 1", duration);

      task.data = Reader::read(task.filename);

      duration = task.nextStage();
      stage2Queue.push(task);
      logTask(task, "finished step 1", duration);
    }
  }
}

void TaskManager::processDataStage() {
  while (running || taskCounter) {
```

```
    Task task;
    if (stage2Queue.try_pop(task)) {
      auto duration = task.nextStage();
      logTask(task, "started step 2", duration);

      Parser p(task.data);
      task.title = p.getTitle();
      task.image = p.getImage();
      task.ingredients = p.getIngredients();
      task.steps = p.getSteps();

      duration = task.nextStage();
      stage3Queue.push(task);
      logTask(task, "finished step 2", duration);
    }
  }
}

void TaskManager::writeToDBStage() {
  while (running || taskCounter) {
    Task task;
    if (stage3Queue.try_pop(task)) {
      auto duration = task.nextStage();
      logTask(task, "started step 3", duration);

      db.insertRecipe(task.id, task.title, task.ingredients, task.steps,
                      task.image);

      duration = task.nextStage();
      finished.push_back(task);
      taskCounter--;
      logTask(task, "finished step 3", duration);
    }
  }
}
```

Таким образом, создаётся 3 потока, выполняющих обработку каждого из трёх этапов соответственно.

Задачи перемещаются по очередям соответствующих этапов, пока их обработка не будет завершена.

3 Примеры работы ПО

В качестве примера будет рассмотрена обработка одной из веб-страниц сайта `chiefs.kz`. На рисунке 3.1 изображена веб-страница в браузере.

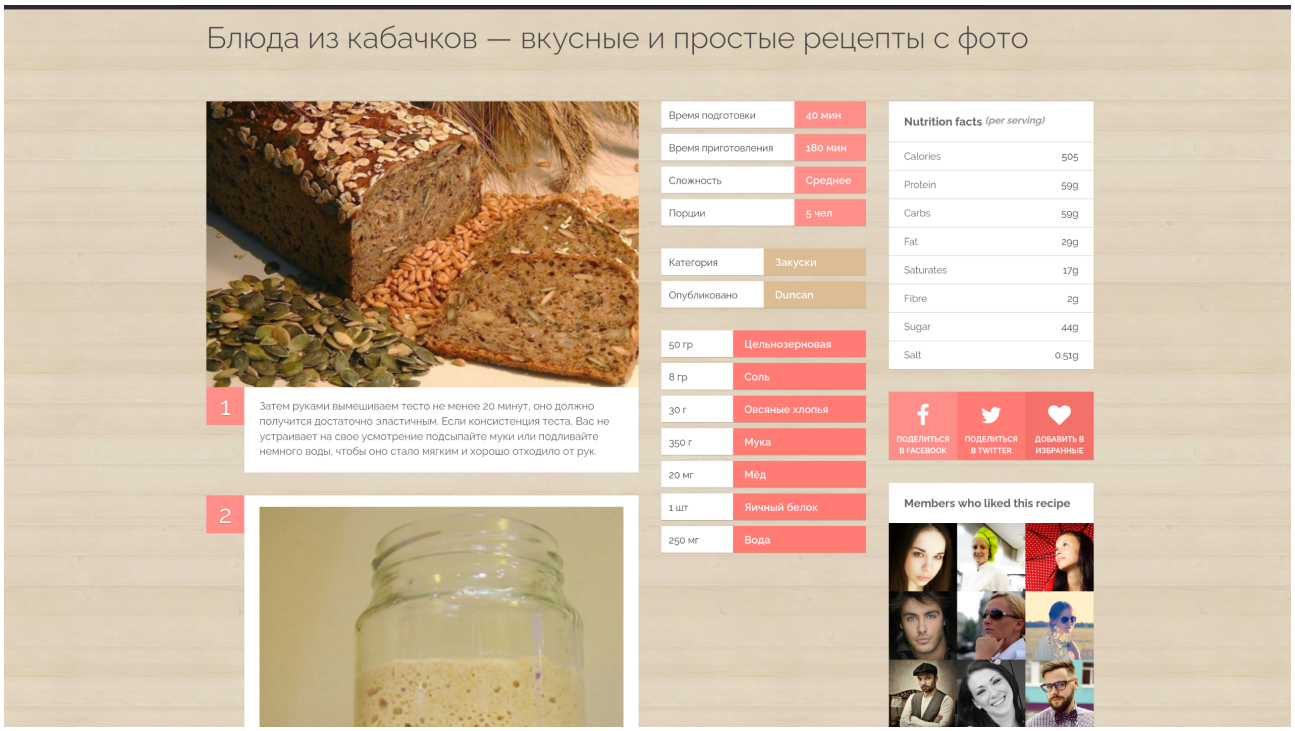

Рисунок 3.1 — Скриншот одной из веб-страниц сайта

json-файл, полученный после обработки алгоритмом представлен в листинге 3.1;

Листинг 3.1 — json-файл, полученный после обработки рецепта

```
{
    "id": 4,
    "image_url": "chiefs.kz/img/0/2061.jpg",
    "ingredients": "{[{\"name\":\"Цельнозерновая\",\"quantity\":\"50 гр
       \"},{\"name\":\"Соль\",\"quantity\":\"8 гр\"},{\"name\":\"Овсяные х
       лопья\",\"quantity\":\"30 г\"},{\"name\":\"Мука\",\"quantity
       \":\"350 г\"},{\"name\":\"Мёд\",\"quantity\":\"20 мг\"},{\"name
       \":\"Яичный белок\",\"quantity\":\"1 шт\"},{\"name\":\"Вода\",\"
       quantity\":\"250 мг\"},]}",
    "issue_id": 9185,
    "steps": "1: Затем руками вымешиваем тесто не менее 20\nминут, оно дол
       жно получится достаточно\nэластичным. Если консистенция теста, Вас
       не\n устраивает на свое усмотрение подсыпайте муки\nили подливайте
       немного воды, чтобы оно\nмягким и хорошо отходило от рук.\n2: В одн
       ой порции пшеничной закваски около 50\nграмм, это нужное количество
        для приготовления\nзакваски используемой цельнозерновую муку. Для\
       nприготовления такой опары, добавляем в готовую\nпшеничную закваску
```

```
        50 грамм теплой воды и такое\nже количество цельнозерновой муки и
        даем\nнастояться и подняться.\n 3: Из полученной порции опары и буд
        ем готовить наш\n хлеб.\n4: Для этого все ингредиенты смешиваем вое
        дино,\nисключая яичный белок и овсяные хлопья, они\nбудут участвова
        ть в завершительном процессе\nприготовления. Смесь пока особо не вы
        мешиваем,\nоставляем на полчаса.\n",
    "title": "Блюда из кабачков — вкусные и простые рецепты с фото"
},
```

4 Тестирование ПО

Тестирование программы проводилось посредством загрузки 1024 html-файлов. При этом отслеживалось отсутствие ошибок, а также вручную были проверены полученные в БД записи обо всех четырёх страницах, доступных на сайте.

На листинге 4.1 представлен фрагмент лога, сформированного конвейером в процессе обработки задач.

Листинг 4.1 — фрагмент лога программы

```
Task 1023 finished step 1 in    [34      microseconds.]
Task 1024 started step 1 in     [38559   microseconds.]
Task 1024 finished step 1 in    [33      microseconds.]
Task 31 finished step 2 in      [1066    microseconds.]
Task 32 started step 2 in       [36726   microseconds.]
Task 25 finished step 3 in      [1227    microseconds.]
Task 26 started step 3 in       [6108    microseconds.]
Task 32 finished step 2 in      [1193    microseconds.]
Task 33 started step 2 in       [37940   microseconds.]
Task 26 finished step 3 in      [1258    microseconds.]
Task 27 started step 3 in       [6291    microseconds.]
```

Как видно из фрагмента лога, первый этап обработки для всех 1024 задач был завершён уже тогда, когда обработку вторым этапом проходила только 31-я задача. После окончания обработки 31 задачи, конвейер сразу приступил к обработке задачи с номером 32 — следующей в очереди. Аналогичное поведение наблюдается и с третьим этапом (задачами 25, 26).

4.1 Описание исследования

Было проведено исследование времени обработки и времени ожидания задач в очередях каждого из этапов обработки.

Результаты тестирования приведены в таблице 4.1.

Таблица 4.1 — Таблица времени обработки задач на разных этапах

Этап	Среднее время этапа (мкс)
Полная обработка	1308225
Ожидание в очереди первого этапа	91305
Обработка на первом этапе	107
Ожидание в очереди второго этапа	1212228
Обработка на втором этапе	2256
Ожидание в очереди третьего этапа	446
Обработка на третьем этапе	1881

4.2 Вывод

В результате измерений было обнаружено, что при обработке 1024 страниц с ресурса chiefs.kz, второй этап оказался более трудоёмким, чем первый примерно в 22 раза. Именно этим обусловлено сравнительно большое время простоя в очереди второго этапа. Третий же этап обработки занимал меньше времени, чем второй, поэтому блокировки в очереди на третий этап не происходило.

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута, а именно: было разработано ПО, осуществляющее обработку html-файлов — страниц ресурса `chiefs.kz` методом конвейерной обработки.

В результате анализа реализованной программы было обнаружено, что в среднем — этап обработки и получения данных с html-страницы является самым долгим этапом. Из-за этого происходило долгое ожидание задач в очереди, а также простой обработчика первого этапа (т. к. все 1024 задачи обрабатывались гораздо раньше окончания работы всего конвейера).

Для достижения цели были выполнены следующие задачи:

— была рассмотрена структура страниц с рецептами ресурса chiefs.kz;

— было разработано ПО, выполняющее обработку веб-страницы и запись данных в БД;

— было разработано ПО, выполняющее эту задачу методом конвейерной обработки;

— было реализовано разработанное ПО;

— полученная реализация была проанализирована по временным характеристикам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. David B. Skillicorn, Domencio Talia Models and languages for parallel computation // Журнал ACM Comput. Surv. // Нью Йорк: Association for Computing Machinery // 1998г. // страницы 123-169

2. Таненбаум Э., Бос Х. Современные операционные системы. // Таненбаум Э., Бос Х. 4-е издание // СПб.: Питер // 2015г. // 1120с.

3. ISO International Standard ISO/IEC 14882:2020(E) [Working Draft] – Programming Language C++ // Geneva, Switzerland: International Organization for Standardization (ISO)

4. Документация библиотеки SQLite3 / [Электронный ресурс] // [сайт] URL: `https://www.sqlite.org/docs.html` (дата обращения: 15 февраля 2025 г.)

5. Документация библиотеки Gumbo / [Электронный ресурс] // [сайт] URL: `https://github.com/google/gumbo-parser` (дата обращения: 15 февраля 2025 г.)

6. Документация библиотеки OneTBB / [Электронный ресурс] // [сайт] URL: `https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb-documentation.html` (дата обращения: 15 февраля 2025 г.)