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

Дисциплина: Анализ алгоритмов

Тема: Реализация конвейера с использованием параллельных вычислений

Студент: Платонова Ольга

Группа: ИУ7-55Б

Преподаватели: Волкова Л. Л.
Строганов Ю. В.

Москва, 2020 г.

Оглавление

Введение

В данной лабораторной работе требуется изучить и реализовать систему конвейерной обработки с использованием параллельных вычислений. Также необходимо выполнить тестирование и анализ разработанного алгоритма.

1. Аналитическая часть

В данном разделе будут рассмотрены понятия, связанные с конвейерной обработкой и параллельными вычислениями, а также сформулированы основные принципы алгоритмов, ее реализующих.

1.1. Конвейерная обработка

Основу конвейерной обработки составляет раздельное выполнение некоторой операции в несколько этапов (за несколько ступеней) с передачей данных одного этапа следующему. Для выполнения каждой ступени выделяется отдельный блок аппаратуры. Так, обработку любой машинной команды можно разделить на несколько этапов, организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняется несколько команд. [1]

Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

В результате организации многопоточности может возникнуть проблема защиты данных от повреждения в результате асинхронных изменений (состояние гонки). Одним из вариантов решения этой проблемы является использование мьютексов.

1.2. Организация параллельных вычислений

Мьютекс является аналогом одноместного семафора, в программировании необходим для сопоставления синхронно выполняющихся потоков. Мьютекс представляет собой концепцию программирования, которая используется для решения вопросов многопоточности. Мьютекс отличается от семафора тем, что допускает только один поток в контролируемом участке, заставляя другие потоки, которые пытаются получить доступ к этому разделу ждать, пока первый поток не вышел из этого раздела. [2]

Принимает два значения:

1. открыт - поток может войти в свою критическую секцию;
2. закрыт - поток не может войти в критическую секцию.

Задача мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом. В каждый конкретный момент только один поток может владеть объектом, защищённым мьютексом. Если другому потоку будет нужен доступ к переменной, защищённой мьютексом, то этот поток засыпает до тех пор, пока мьютекс не будет освобождён.

1.3. Параллельная реализация конвейера

На рисунке 1.1 изображен принцип работы алгоритма параллельной конвейерной обработки. В данной работе рассматривается конвейер с тремя лентами.

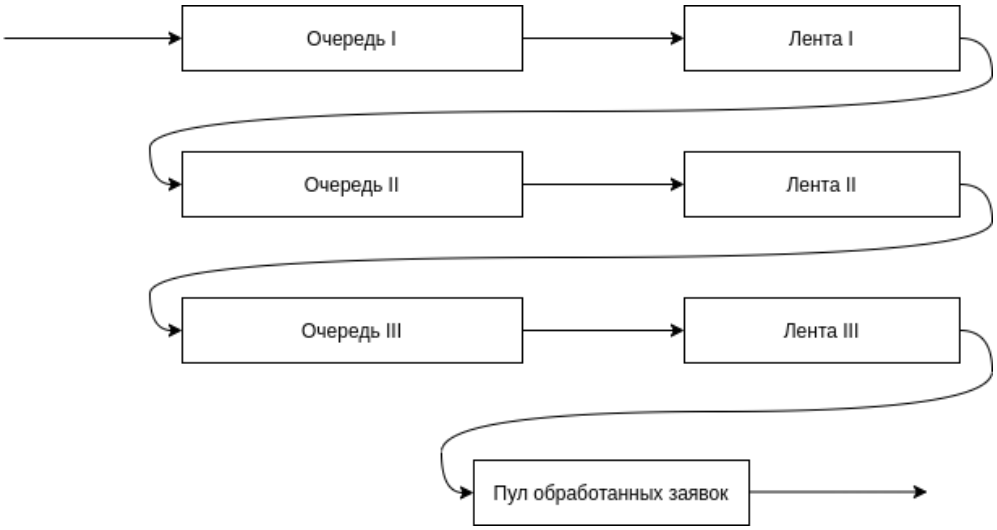

Рисунок 1.1. Алгоритм параллельной реализации конвейера.

1.4. Вывод

В данном разделе были рассмотрены понятия, связанные с конвейерной обработкой и параллельными вычислениями, а также сформулированы основные принципы алгоритмов, ее реализующих.

2. Конструкторская часть

В данном разделе будет рассмотрена схема и описание алгоритма параллельной реализации конвейерной обработки.

2.1. Разработка алгоритмов

На рисунке 2.1 изображена схема *запуска конвейера*.

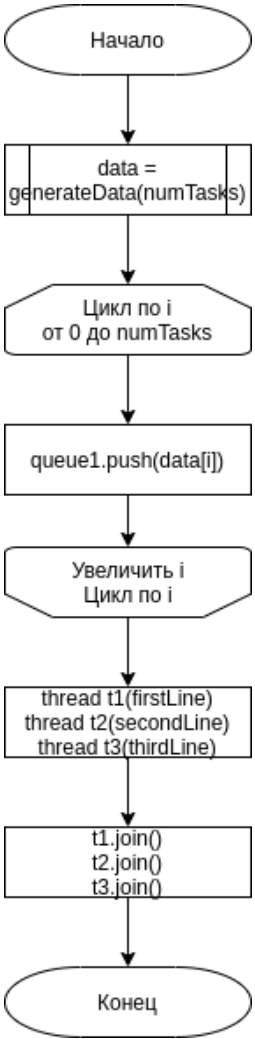

Рисунок 2.1. Схема запуска конвейерной обработки.

На рисунке 2.2 изображена схема работы *линии конвейера.*

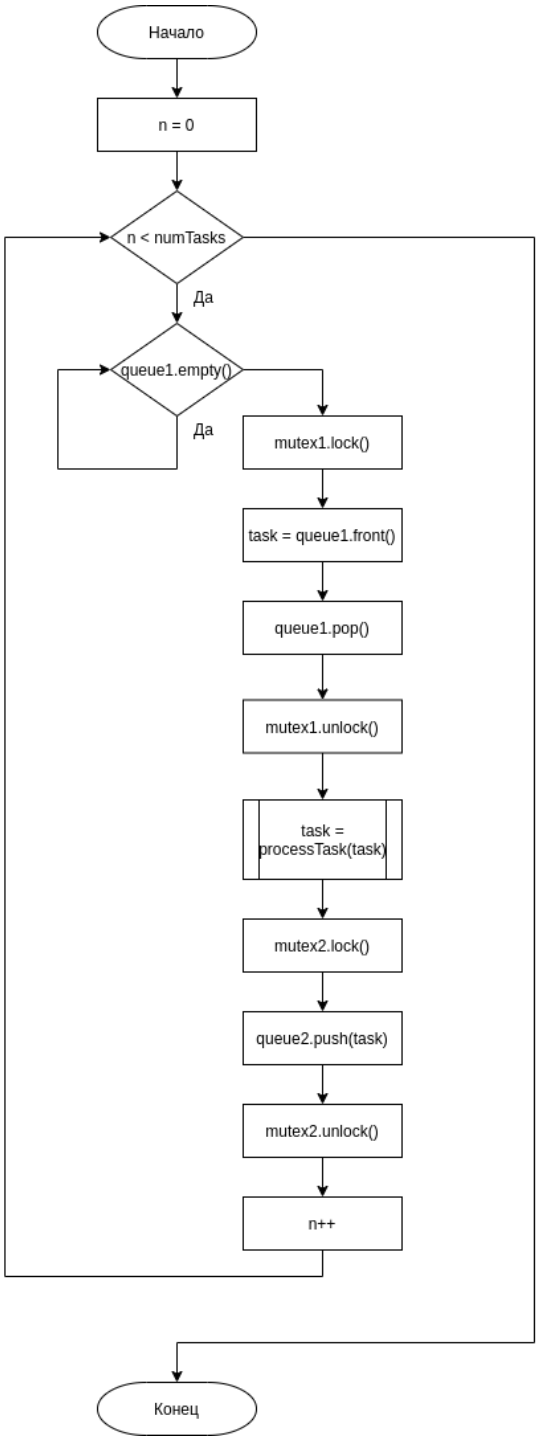

Рисунок 2.2. Схема работы конвейерной линии.

2.2. Описание работы конвейерной системы

Работа алгоритма начинается с запуска конвейера. Для этого необходима предварительная обработка данных, заключающаяся в заполнении очереди первой ленты сгенерированными данными. Конвейер состоит из трех лент. Для каждой ленты организована собственная очередь, состоящая из объектов, ожидающих поступление на ленту. Как только объект из очереди попадает на ленту, он обрабатывается некоторое время, а затем попадает в очередь следующей ленты. Объект, обработанный на третьей линии, попадает в пул обработанных объектов.

Также, для организации монопольного доступа к очередям, для каждой очереди определен собственный мьютекс. При работе с очередью линия блокирует ее соответствующим мьютексом и выполняет добавление/извлечение объекта. Ожидание объекта линией организовано бесконечным циклом, который выполняется до тех пор, пока очередь линии пуста.

2.3. Вывод

В данном разделе была рассмотрена схема и был описан алгоритм параллельной реализации конвейерной обработки.

3. Технологическая часть

В данном разделе будет рассмотрен язык программирования, среда разработки, требуемые инструменты для реализации. Также будет представлена реализация алгоритма.

3.1. Требования к программному обеспечению

I. Программа должна предусматривать генерацию данных.
II. Все сгенерированные данные должны быть обработаны конвейером.
III. В результате обработки порядок данных не должен быть нарушен.
IV. Объем данных должен быть задан заранее, число линий равно 3.

3.2. Средства реализации

В данной работе используется язык программирования C++ из-за опыта написания на нем. Среда разработки – Qt.

Для замеров процессорного времени использовалась функция clock(). [3]

3.3. Реализация алгоритмов

Реализация запуска конвейера представлена в листинге 3.1.

Листинг 1. Запуск конвейера.

```
int main()
{
   vector<int> objects = generateData(numTasks);

   clock_t begin = clock();

   //Заполнение первой очереди
   for (int i = 0; i < numTasks; i++) {
      mutex1.lock();
         queue1.push(objects[i]);
      mutex1.unlock();
   }

   thread t1(firstLine);
   thread t2(secondLine);
   thread t3(thirdLine);

   t1.join();
   t2.join();
   t3.join();
```

```
    clock_t end = clock();
    cout << "Time: " << end - begin << endl;

    return 0;
}
```

Реализация конвейерных линий представлена в листинге 3.2.

Листинг 2. Алгоритм обработки объекта на линиях.

```
void firstLine()
{
    int n = 0;
    while (n < numTasks) {
        if (queue1.empty()) {
            continue;
        }

        mutex1.lock();
            int task = queue1.front();
            queue1.pop();
        mutex1.unlock();

        time_t begin = clock();
            this_thread::sleep_for(chrono::seconds(3));
        time_t end = clock();
        conveyorOutput(1, task, begin, end);

        mutex2.lock();
            queue2.push(task);
        mutex2.unlock();

        n++;
    }
}

void secondLine()
{
    int n = 0;
    while (n < numTasks) {
        if (queue2.empty()) {
            continue;
        }

        mutex2.lock();
            int task = queue2.front();
            queue2.pop();
        mutex2.unlock();

        time_t begin = clock();
            this_thread::sleep_for(chrono::seconds(1));
        time_t end = clock();
        conveyorOutput(2, task, begin, end);

        mutex3.lock();
```

```
            queue3.push(task);
        mutex3.unlock();

        n++;
    }
}

void thirdLine()
{
    int n = 0;
    while (n < numTasks) {
        if (queue3.empty()) {
            continue;
        }

        mutex3.lock();
            int task = queue3.front();
            queue3.pop();
        mutex3.unlock();

        time_t begin = clock();
            this_thread::sleep_for(chrono::seconds(2));
        time_t end = clock();
        conveyorOutput(3, task, begin, end);

        objResPool.push_back(task);

        n++;
    }
}
```

3.4. Описание тестирования

Для проверки корректности программы необходимо выполнить контроль за порядком обработки аргументов каждой из линии и за сформированным набором обработанных аргументов.

3.5. Вывод

В данном разделе были рассмотрены инструменты, необходимые для реализации алгоритма конвейерной обработки с использованием параллельных вычислений, а также были представлены непосредственно реализации.

4. Экспериментальная часть

В данном разделе будут рассмотрены примеры работы программы, произведено тестирование и контроль выполнения, выполнены замеры времени, а также выполнен анализ полученных данных.

4.1. Примеры работы программы

Результат работы программы представлен на рисунке 4.1.

```
Line: 1
Task: 0
        Time begin: 10599
        Time end: 5996942
Line: 2
Task: 0
        Time begin: 5997260
        Time end: 6997243
Line: 1
Task: 1
        Time begin: 5997055
        Time end: 8997246
Line: 3
Task: 0
        Time begin: 6997653
        Time end: 8997674
Line: 2
Task: 1
        Time begin: 8997662
        Time end: 9997247
Line: 1
Task: 2
        Time begin: 8997277
        Time end: 11996707
Line: 3
Task: 1
        Time begin: 9997843
        Time end: 11997223
Line: 2
Task: 2
        Time begin: 11997211
        Time end: 12996472
Line: 1
Task: 3
        Time begin: 11996734
        Time end: 14992113
Line: 3
Task: 2
        Time begin: 12997080
        Time end: 14992835
Line: 2
Task: 3
        Time begin: 14992697
        Time end: 15991845
Line: 3
Task: 3
        Time begin: 15992526
        Time end: 15992623

Resulted pool: 0 1 2 3
Time: 15990810
```

Рисунок 4.1. Результат работы программы.

4.2. Технические характеристики устройства.

Операционная система – LinuxMint 64-bit;

Память – 8 ГБ;

Процессор – Intel™ Core™ i3-7100U CPU @ 2.40 ГГц;

Логических процессов – 4.

4.3. Анализ полученных данных

Рассмотрим работу программы для случая 4 объектов.

Изначально все четыре объекта расположены в очереди первой линии. Затем первый объект попадает на линию, обрабатывается. и после обработки попадает в очередь второй линии. Далее первый объект обрабатывается на второй линии и попадает в очередь третьей линии. Однако затем управление передается снова на первую линию, где обрабатывается второй объект. И только после этого первый объект завершает обработку, попадая на третью линию, откуда затем размещается в пуле обработанных объектов.

Стоит отметить, что порядок обработанных данных не был изменен, и порядок обработки на линиях не был нарушен.

4.4. Вывод

В данном разделе были рассмотрены примеры работы программы, произведено тестирование и контроль выполнения, выполнены замеры времени, а также выполнен анализ полученных данных.

Заключение

Цель работы достигнута, все поставленные задачи выполнены: был разработан и реализован алгоритм конвейерной обработки с использованием параллельных вычислений, а также проведено тестирование и анализ его работы.

Была продемонстрирована распараллеленная обработка задач конвейера: сначала первый объект обрабатывается на первой, второй линиях, затем второй объект обрабатывается на первой, и только после этого первый объект попадает на третью линию.

Несомненно, такая реализация задач дает выигрыш по времени, параллельная конвейерная система работает быстрее примерно в 1.5 раза линейной реализации.

Список литературы

1. Конвейерная обработка данных. – URL: https://studref.com/636041/ekonomika/konveyernaya_obrabotka_dannyh (дата обращения: 01.12.2020). Текст: электронный.
2. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. — СПб: БХВ-Петербург, 2002. — 608 с. — ISBN 5-94157-160-7.
3. Техническая документация.-URL: https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=vs-2019 (дата обращения: 20.09.2020). Текст: электронный.