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»

ДИСЦИПЛИНА «Анализ алгоритмов»

Лабораторная работа № 5

Тема «Конвейерная обработка данных»

Студент Чалый А.А.

Группа ИУ7 – 52Б

Преподаватели Волкова Л.Л.
Строганов Ю.В.

Москва.
2020 г.

Оглавление

Введение

Целью данной работы является получение навыка организации асинхронной передачи данных между потоками на примере конвейерной обработки информации.

Задачи работы:

1) выбрать и описать методы обработки данных, которые будут сопоставлены методам конвейера;
2) описать архитектуру программы, а именно какие функции имеет главный поток, принципы и алгоритмы обмена данными между потоками;
3) реализовать конвейерную систему, а также сформировать лог событий с указанием времени их происхождения, описать реализацию;
4) интерпретировать сформированный лог.

1. Аналитическая часть

Одной из важнейших идей при создании многопроцессорных систем и при эффективной реализации на этих системах является идея конвейерных вычислений.

1.1 Описание задачи

Конвейеризация — это техника, в результате которой задача или команда разбивается на некоторое число подзадач, которые выполняются последовательно. Каждая подкоманда выполняется на своем логическом устройстве. Все логические устройства (ступени) соединяются последовательно таким образом, что выход i-ой ступени связан с входом (i+1)-ой ступени, все ступени работают одновременно. Множество ступеней называется конвейером. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы ступеней, вовлекая на каждом такте новую задачу или команду [1].

В конвейере различают r последовательных этапов, так что когда i-я операция проходит s-й этап, то (i+k)-я операция проходит (s-k)-й этап. На рис. 1 изображена работа конвейера.

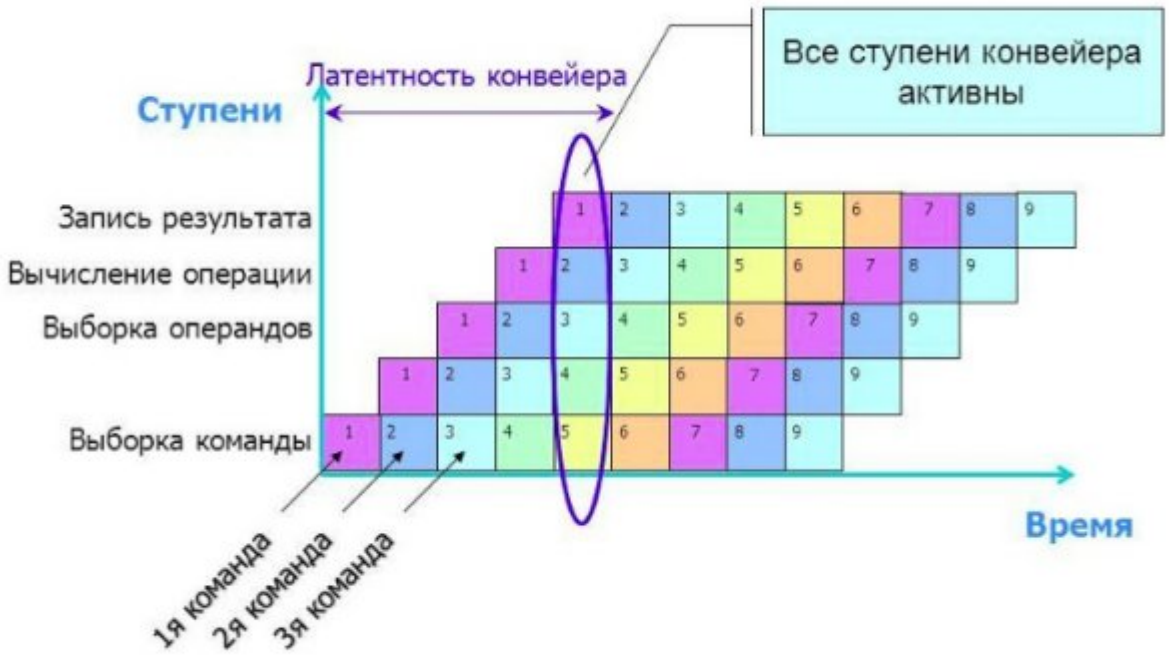

Рисунок 1. Работа конвейера

В данной лабораторной работе реализована некая функция кодирования строки, которая состоит из трех последовательных действий: применения первой функции цифра Цезаря, применения функции которая меняет элемент местами с элементом, стоящим на n / 2 от него, где n - размер строки. Если необходимо закодировать какой-то массив строк, то можно использовать конвейерную обработку данных. Таким образом задача будет решена эффективнее, чем при последовательном применении алгоритмов к массиву значений.

Конвейер будет состоять из четырех уровней. Обработанные данные передаются последовательно с одного уровня (одной ленты) конвейера на следующий (следующую ленту). Далее на каждом уровне осуществляется обработка данных, занимающая определенное время. Для каждой ленты создается своя очередь задач, в которой хранятся все необработанные строки. На последнем уровне конвейера обработанные объекты попадают в пул обработанных задач.

Уровни конвейера:

- 0 уровень — генерация входных данных в первую очередь;
- 1 уровень (лента) — применение шифра Цезаря к строкам из первой очереди, запись результата во 2 очередь;
- 2 уровень (лента) — применение функции, меняющей регистры символов к строкам 2 очереди, запись результата в 3 очередь;
- 3 уровень (лента) — применение функции, меняющей местами символы в строке по определенному выше закону к строкам 3 очереди, запись результата в пул обработанных задач.

Поскольку запись в очередь и извлечение из очереди это не атомарные операции, необходимо создать их путем использования

мьютексов (по одному на одну очередь) и критических секций, чтобы избежать ошибок в ситуации гонок.

2. Конструкторская часть

В данном разделе будут рассмотрена схема работы алгоритма конвейерной обработки, описание архитектуры ПО и схемы рабочего и главного процессов.

2.1 Описание архитектуры ПО

В конвейере 3 основные ленты, содержание их работы описано выше, в аналитической части отчета. Каждой ленте выделен свой поток, в котором она выполняется. В главном потоке (функция main) создаются три рабочих потока: по одному на каждую ленту.

Для каждой ленты есть своя очередь, однако с ней могут работать все потоки, поэтому при доступе к элементам очереди необходимо блокировать доступ для других потоков. Для реализации доступа из разных потоков используются мьютексы, по одному для каждой очереди и для результирующего массива.

Также в главном потоке генерируется массив входных данных и заполняется первая очередь (уровень 0). Для моделирования ситуации постепенного поступления данных, первая очередь заполняется с задержкой.

В рабочих процессах считывается по одному элементу из соответствующей очереди, выполняется вызов обрабатывающих функций, замеры времени и задержка по времени. После обработки текущей задачи, рабочий процесс записывает результат в следующую очередь или в результирующий массив. Также выполняется запись в лог—файл.

На рис. 2 изображена схема работы конвейерной обработки.

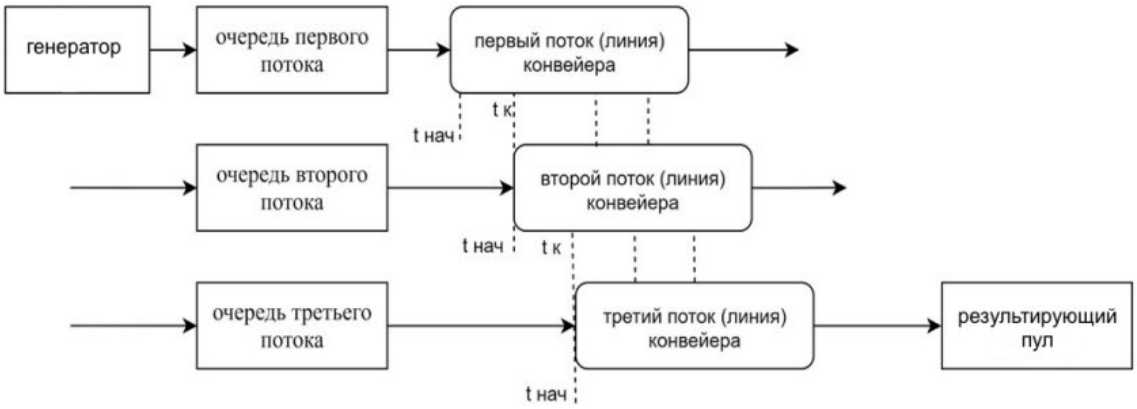

Рисунок 2. Схема работы конвейерной обработки.

2.2 Схема алгоритмов работы конвейерной обработки

На рис. 3 и 4 приведены схемы главного и рабочего процессов.

Рисунок 3. Схема работы главного процесса

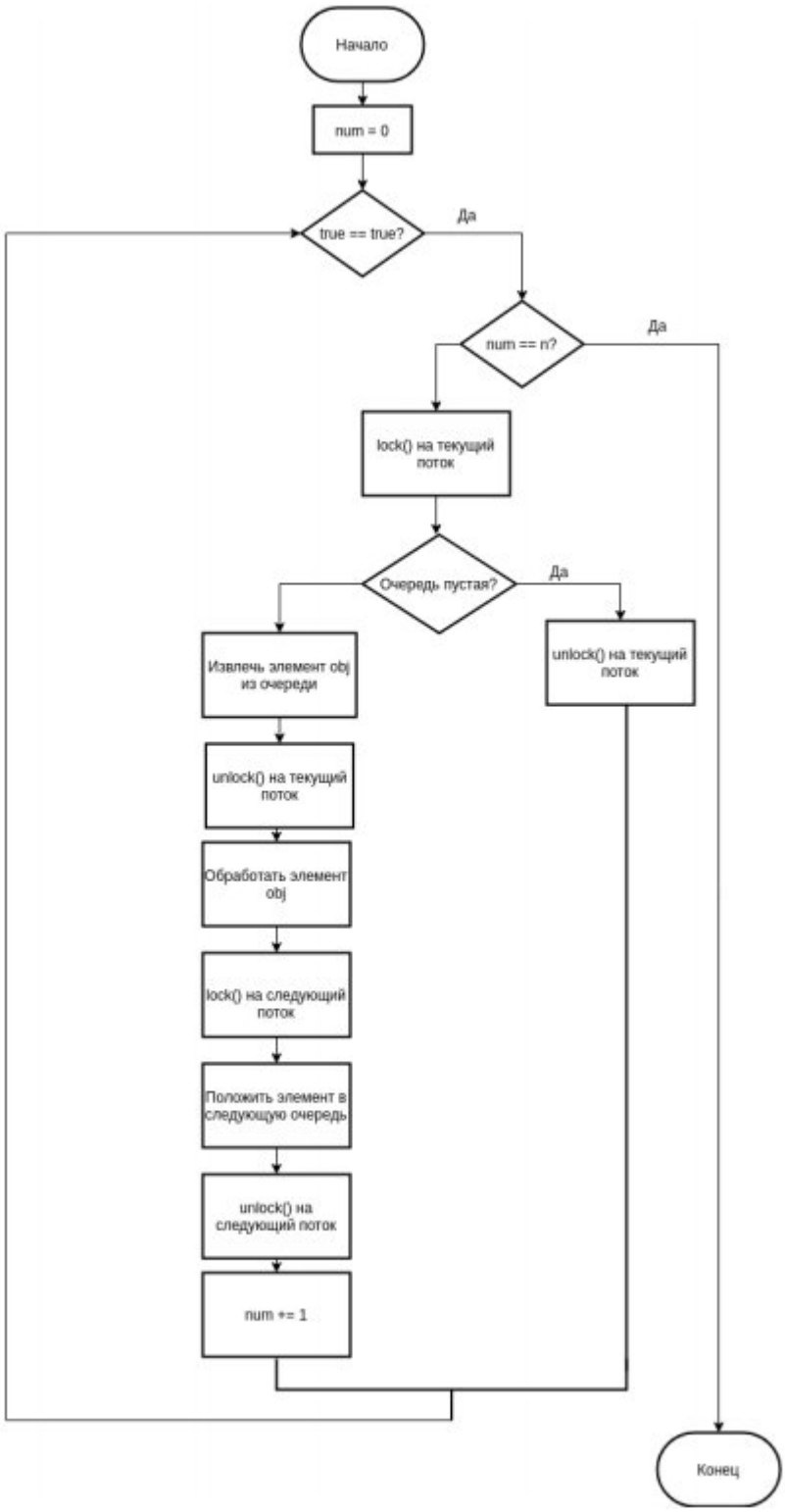

Рисунок 4. Схема работы рабочего процесса

3.Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

Входные данные: n — количество строк.

Выходные данные: зашифрованные строки.

Функциональная схема процесса шифрования строк в нотации IDEF0 представлена на рис. 2.

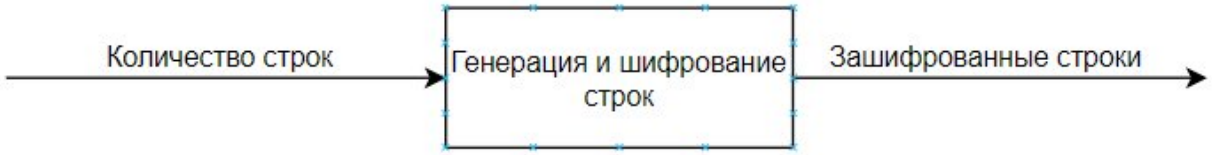

Рисунок 5. Функциональная схема процесса шифрования строк

3.2 Средства реализации

В качестве языка программирования был выбран C++ т.к. данный язык часто использовался мною. Данный язык позволяет писать высокоуровневый абстрактный код, который при этом работает со скоростью близкой к машинному коду.

Среда разработки – QtCreator, которая предоставляет умную проверку кода, быстрое выявление ошибок и оперативное исправление, вкупе с автоматическим рефакторингом кода и богатыми возможностями в навигации.

Время работы было замерено с помощью функции clock() из библиотеки ctime. [2]

Для тестирования использовался компьютер на базе процессора Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz, 3401 МГц, ядер: 4, логических процессоров: 4.

Многопоточное программирование было реализовано с помощью библиотеки thread. [3]

3.3 Листинг кода

В данном пункте представлен листинг кода функций. В листинге 1, 2, 3 представлен код рабочих потоков. В листинге 4 представлен код главного потока.

Листинг 1. Реализация первого уровня конвейера

```
void conveyor1(){
    int num = 0;
    while (1)
    {
        if (num == n)
            break;
        m1.lock();
        if (queue1.empty())
        {
            m1.unlock();
            continue;
        }
        input_t myObj = queue1.front();
        queue1.pop();
        m1.unlock();
        input_t newObj = caesar(myObj);
        m2.lock();
        queue2.push(newObj);
        sleep(1);
        clock_t time = clock();
        log.print(1, newObj, num, time);
        m2.unlock();
        num++;
    }
```

```
}
```

Листинг 2. Реализация второго уровня конвейера

```
void conveyor2(){
    int num = 0;
    while (1)
    {
        if (num == n)
            break;
        m2.lock();
        if (queue2.empty())
        {
            m2.unlock();
            continue;
        }
        input_t myObj = queue2.front();
        queue2.pop();
        m2.unlock();
        input_t newObj = upper_lower(myObj);
        m3.lock();
        queue3.push(newObj);
        sleep(3);
        clock_t time = clock();
        log.print(2, newObj, num, time);
        m3.unlock();
        num++;
    }
}
```

Листинг 3. Реализация третьего уровня конвейера

```
void conveyor3(){
    int num = 0;
    while (1)
    {
        if (num == n)
            break;
        m3.lock();
        if (queue3.empty())
        {
            m3.unlock();
```

```
            continue;
        }
        input_t myObj = queue3.front();
        queue3.pop();
        m3.unlock();
        input_t newObj = reverse(myObj);
        resm.lock();
        res.push_back(newObj);
        sleep(1.5);
        clock_t time = clock();
        log.print(3, newObj, num, time);
        resm.unlock();
        num++;
    }
}
```

Листинг 4. Основная функция программы

```
int main(){
    f = fopen("log.txt", "w");
    n = 5;
    objvec.resize(n);

    thread t1(conveyor1);
    thread t2(conveyor2);
    thread t3(conveyor3);
    main_time = clock();

    for (int i = 0; i < n; i++)
    {
        string s = generate();
        objvec[i] = (s);
    }

    for (int i = 0; i < n; i++)
    {
        clock_t tm = clock();
        log.print(0, objvec[i], i, tm);
        m1.lock();

        queue1.push(objvec[i]);
```

```
            m1.unlock();
            sleep(2);
        }

        t1.join();
        t2.join();
        t3.join();
        fclose(f);
        printf("That's all folks!");
        return 0;
}
```

Вывод

В данном разделе была представлена реализация трех уровней конвейера , а также основной функции программы.

4. Экспериментальная часть

В данном разделе будет проведен анализ полученных данных.

4.1 Анализ данных

В результате работы программы получаем файл данных, содержимое которого приведено на рис. 6. Время замерено в тактах процессора.

```
step: 0 | item: 0 | time: 1 (0) | value: HgUeyNFXICSxgWfQUWNOVrKPEGxPRYtwyycpvkfMNmksWrnKcF
step: 1 | item: 0 | time: 1003 (1002) | value: IhVfzOGYJDTyhXgRVXOPWsLQFHyQSZuxzzdqwlgNOnltXsoLdG
step: 0 | item: 1 | time: 2003 (1000) | value: TSySAPofZCeUPAOgOyFBPJRIYmeWnrpxjLoAwxhsCXosZvtDnY
step: 1 | item: 1 | time: 3005 (1002) | value: UTzTBQpgADfVQBPhPzGCQKSJZnfXosqykMpBxyitDYptAwuEoZ
step: 0 | item: 2 | time: 4004 (999) | value: YFxSnkfUXRBngoktYNouzrdItijayPSmSjFGbAoLYNRdpsTsWD
step: 2 | item: 0 | time: 4005 (1000) | value: iHvFZogyjdtYHxGrvxopwSlqfhYqszUXZZDQWLGnoNLTxSOlDg
step: 1 | item: 2 | time: 5006 (1001) | value: ZGyTolgVYSCohpluZOpvaseJujkbzQTnTkGHcBpMZOSeqtUtXE
step: 3 | item: 0 | time: 5007 (1002) | value: hYqszUXZZDQWLGnoNLTxSOlDgiHvFZogyjdtYHxGrvxopwSlqf
step: 0 | item: 3 | time: 6006 (999) | value: zoZnWQjVWXRbDGlgUbbrTHspOGgKuDOObxEaFtueZhlUgejQJn
step: 1 | item: 3 | time: 7008 (1002) | value: apAoXRkWXYScEHmhVccsUItqPHhLvEPPcyFbGuvfAimVhfkRKo
step: 2 | item: 1 | time: 8008 (1000) | value: utZtbqPGadFvqbpHpZgcqksjzNFxOSQYKmPbXYITdyPTaWUeOz
step: 0 | item: 4 | time: 8008 (1000) | value: EQxeJrZAULCHFmiwOIyXUnSsSTkqCzEuMrczjpIneDSRAYmABZ
step: 3 | item: 1 | time: 9009 (1001) | value: NFxOSQYKmPbXYITdyPTaWUeOzutZtbqPGadFvqbpHpZgcqksjz
step: 1 | item: 4 | time: 9010 (1002) | value: FRyfKsABVMDIGnjxPJzYVoTtTUlrDaFvNsdakqJofETSBZnBCA
step: 2 | item: 2 | time: 11009 (2000) | value: zgYtOLGvyscOHPLUzoPVASEjUJKBZqtNtKghCbPmzosEQTuTxe
step: 3 | item: 2 | time: 12010 (1001) | value: JKBZqtNtKghCbPmzosEQTuTxezgYtOLGvyscOHPLUzoPVASEjU
step: 2 | item: 3 | time: 14010 (2000) | value: APaOxrKwxysCehMHvCCSuiTQphHlVeppCYfBgUVFaIMvHFKrkO
step: 3 | item: 3 | time: 15012 (1002) | value: hHlVeppCYfBgUVFaIMvHFKrkOAPaOxrKwxysCehMHvCCSuiTQp
step: 2 | item: 4 | time: 17012 (2000) | value: frYFkSabvmdigNJXpjZyvOtTtuLRdAfVnSDAKQjOFetsbzNbca
step: 3 | item: 4 | time: 18014 (1002) | value: uLRdAfVnSDAKQjOFetsbzNbcafrYFkSabvmdigNJXpjZyvOtTt
```

Рисунок 6. Содержимое файла с данными

Полученный файл состоит из записей, отсортированных по времени по возрастанию с начала запуска приложения. В первом столбце указан номер конвейера, во втором столбце — номер текущего элемента, затем время с начала работы программы и время выполнения данного этапа в миллисекундах, а также значение хеша на текущем этапе.

По листингу можно проследить выполнение каждого этапа и изменение значений хеша при обработке на каждой из лент. Также можно проследить, что обработка выполняется параллельно. Например, 2 линия не ждет полного завершения работы первой прежде чем начать работу, а начинает выполнение как только в очередь поступает первый элемент.

Так как обработка на каждой линии занимает разное кол-во времени, в листинге можно увидеть, насколько она эффективнее. Если бы мы

обрабатывали каждый элемент последовательно то потребовалось бы (1000 + 3000 + 1500) * 5 = 27500 миллисекунд. Однако алгоритм выполнился за 18014 миллисекунд, что дает выигрыш в 9.5 секунд.

Вывод

Конвейерная обработка данных — полезный инструмент, который уменьшает время выполнения программы за счет параллельной обработки данных. Самым эффективным временем считается время, когда все линии конвейера работают параллельно, обрабатывая свои задачи. Этот метод дает выигрыш по времени в том случае, когда выполняемые задачи намного больше по времени, чем время, затрачиваемое на реализацию конвейера (работу с потоками, перекладывание из очереди в очередь и тд).

Заключение

В ходе работы был получен навык организации асинхронной передачи данных между потоками на примере конвейерной обработки информации. Был выбран и описан метод обработки данных, которые будут сопоставлены методам конвейера. Описана архитектура программы, реализована конвейерная система, а также сформирован лог событий с указанием времени их происхождения, описана реализация, сформированный лог был интерпретирован. Все поставленные задачи были выполнены. Целью лабораторной работы являлось организация асинхронной передачи данных между потоками, что также было достигнуто.

Список использованных источников

1. Конвейерные вычисления [электронный ресурс]. Режим доступа: http://www.myshared.ru/slide/674082, свободный (Дата обращения: 01.12.2020)
2. Официальный сайт Microsoft, документация [электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=vs-2017, свободный (Дата обращения: 10.11.20)
3. Официальный сайт Microsoft, документация [электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/standard-library/thread-class?view=vs-2019, свободный (Дата обращения: 10.11.20)