*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана (национальный исследовательский институт)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА ИУ7

**Отчёт**

**по лабораторной работе №5**

**Дисциплина: Анализ алгоритмов**

**Тема лабораторной работы: Конвейерная обработка данных**

Студент гр. ИУ7-51Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Громова В.П.** (Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Л.Л.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

[**Введение**](#_v5e0rgwynkcz) **2**

[**Аналитическая часть**](#_dapi5c9z0m2q) **4**

[**Конструкторская часть**](#_jal821ed5dwt) **5**

[Описание архитектуры ПО](#_619z5bu7dq8d) 5

[Схемы алгоритмов](#_4m9hthguz677) 6

[**Технологическая часть**](#_o0ztcrggoesu) **9**

[Средства реализации](#_gc5ik9au4m58) 9

[Листинги кода](#_c7jou24c1u5) 9

[**Экспериментальная часть**](#_ok53mdg3ee3t) **13**

[Пример работы программы](#_wwk40k3lf8g2) 13

[Анализ лог-файла](#_6eqhhuopqzau) 14

[**Заключение**](#_84656lgl6hcz) **19**

### Введение

Выполнение каждой команды складывается из ряда последовательных этапов (шагов, стадий), суть которых не меняется от команды к команде. С целью увеличения быстродействия какого-либо процесса и максимального использования всех возможностей процессора в современных системах используется конвейерный принцип обработкиинформации. Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Цель данной лабораторной работы: получить навык организации асинхронной передачи данных между потоками на примере конвейерной обработки информации.

Задачи лабораторной работы следующие.

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. Провести тестирование системы.
5. Интерпретировать сформированный лог.

### Аналитическая часть

В данном разделе будет описан алгоритм работы конвейера.

Конвейер будет состоять из трех уровней. Генератор подает на вход конвейера (первый уровень) некоторые числа. Далее на каждом уровне осуществляется обработка данных, занимающая определенное время. Обработанные данные передаются последовательно с одного уровня (одной ленты) конвейера на следующий (следующую ленту). Для организации работы каждой ленты будет использована очередь задач, которые должны обработаться на этой ленте. Таким образом, лента будет работать, пока в её очереди есть задачи, но задачи попадают в очередь только в том случае, если данные уже были обработаны на предыдущем уровне или если их только сформировал генератор (для первой ленты). При этом время обработки на какой-либо ленте не должно сильно отличаться от времени обработки на остальных лентах, так как в противном случае возможны ситуации простоя одной или нескольких лент конвейера. На последнем уровне конвейера обработанные объекты попадают в пул обработанных задач. После завершения работы конвейера, то есть после завершения работы третьей ленты, проверяется равенство количества поданных на вход конвейера объектов количеству объектов в пуле обработанных задач. Если равенство верное, то конвейер отработал корректно, иначе где-то произошла ошибка и какие-то данные были обработаны неправильно, либо вообще были потеряны.

На каждом уровне конвейера алгоритм обработки информации заменен задержкой выполняемой программы по времени.

### Конструкторская часть

В данном разделе будет представлена схема работы конвейера и описана архитектура программы.

#### Описание архитектуры ПО

Каждой ленте конвейера выделен отдельный поток. В главном потоке main запускаются все три рабочих потока - ленты конвейера: line\_one, line\_two, line\_three. Для каждого потока есть своя очередь, в которой содержатся индекс выполняемой задачи и время в миллисекундах, необходимое на выполнение задачи (время задержки). Также в главном потоке генерируются входные данные, которые помещаются в очередь для первого потока. После завершения работы всех рабочих потоков проверяется массив результирующих элементов и выводится сообщение о результате работы конвейера. В рабочих потоках извлекается очередной элемент из соответствующей очереди, вносится запись в лог-файл о начале обработки очередного элемента, выполняется задержка по времени. После этого новое значение помещается в очередь для следующего потока (или в массив обработанных значений если обработка происходит уже в третьем потоке) и далее в лог-файл заносится запись о завершении обработки очередного элемента на определенной ленте (в определенном потоке). После окончания работы генератора в первую очередь записывается значение “-1”, что говорит о завершении работы конвейера. Это значение передается из очереди в очередь по лентам. На рисунке 2.1 представлена схема работы конвейера (“Г” - генератор числовых значений - времени задержки на первой ленте).

|  |
| --- |
|  |
| *Рис. 2.1 - схема работы конвейера* |

#### Схемы алгоритмов

В рабочих потоках line\_one, line\_two, line\_three выполняются функции firts\_line(), second\_line(), third\_line() соответственно. Функция main() выполняется в главном потоке. На рисунках 2.2 - 2.3 будет представлена схема вышеописанного алгоритма (см. пункт 2.1). Для функций second\_line() и third\_line() схемы будут выглядеть аналогично схеме first\_line().

|  |
| --- |
|  |
| *Рис. 2.2 - схема алгоритма работы главного потока* |

|  |
| --- |
|  |
| *Рис. 2.3 - схема алгоритма работы рабочего потока* |

### Технологическая часть

В этом разделе будут описаны средства реализации и представлены листинги кода.

#### Средства реализации

Для реализации конвейерной обработки информации с использованием потоков был выбран язык программирования C++, так как уже были практические навыки работы с ним. Также причиной выбора этого языка программирования стал тот факт, что в стандартной библиотеке реализован класс thread, который представляет нативный поток. Для того, чтобы потоки могли безопасно использовать разделяемую память, использовался класс mutex. Для определения текущего времени была использована функция стандартной библиотеки std::chrono::high\_resolution\_clock.

#### Листинги кода

Для реализации конвейера было создано 3 функции (не считая main()), функциональные свойства которых совпадают с функциями потоков, описанными в конструкторской части.

В листинге 1 представлена функция главного потока, в листингах 2 - 4 представлены функции рабочих потоков. При этом очереди, мьютексы и массив для лог-файла (items) описаны как глобальные переменные. Функция sort\_log сортирует записи лога по дате, write\_log - записывает данные лога в текстовый файл.

Листинг 1: main() - главный поток:

1. int main()
2. {
3. thread line\_one(&first\_line);
4. thread line\_two(&second\_line);
5. thread line\_three(&third\_line);
6. srand(time(NULL));
7. for(int i = 0; i < items\_number; ++i)
8. {
9. int tmp = (1 + rand() % 3) \* 1000 + 1;
10. cout << tmp << endl;
11. mtx1.lock();
12. que1.push({i, tmp});
13. mtx1.unlock();
14. Sleep(50);
15. }
16. mtx1.lock();
17. que1.push({-1, -1});
18. mtx1.unlock();
19. line\_one.join();
20. line\_two.join();
21. line\_three.join();
22. if (result.size() == items\_number)
23. {
24. cout << "all items passed the conveyor" << endl;
25. sort\_log();
26. write\_log("log.txt");
27. }
28. else
29. cout << "some problems happened" << endl;
30. return 0;
31. }

Листинг 2: first\_line() - рабочий поток - первая лента:

1. void first\_line()
2. {
3. bool finish = false;
4. while(!finish)
5. {
6. mtx1.lock();
7. if (!que1.empty())
8. {
9. pair<int, int> cur\_item = que1.front();
10. que1.pop();
11. int index = cur\_item.first;
12. int t0 = cur\_item.second;
13. if (t0 != -1)
14. {
15. mtx\_log.lock();
16. string tmp = "line 1: start " + to\_string(index) + " item " + to\_string(t0) + " milliseconds";
17. items.push\_back({currentTime(), tmp});
18. mtx\_log.unlock();
19. Sleep(t0);
20. mtx2.lock();
21. que2.push({index, t0 + 101});
22. mtx2.unlock();
23. mtx\_log.lock();
24. tmp = "line 1: end " + to\_string(index) + " item";
25. items.push\_back({currentTime(), tmp});
26. mtx\_log.unlock();
27. }
28. else
29. {
30. mtx2.lock();
31. que2.push({-1, -1});
32. mtx2.unlock();
33. finish = true;
34. }
35. }
36. mtx1.unlock();
37. }
38. }

Листинг 3: second\_line() - рабочий поток - вторая лента:

1. void second\_line()
2. {
3. bool finish = false;
4. while(!finish)
5. {
6. mtx2.lock();
7. if (!que2.empty())
8. {
9. pair<int, int> cur\_item = que2.front();
10. int index = cur\_item.first;
11. int t0 = cur\_item.second;
12. que2.pop();
13. if (t0 != -1)
14. {
15. mtx\_log.lock();
16. string tmp = "line 2: start " + to\_string(index) + " item " + to\_string(t0) + " milliseconds";
17. items.push\_back({currentTime(), tmp});
18. mtx\_log.unlock();
19. Sleep(t0);
20. mtx3.lock();
21. que3.push({index, t0 - 49});
22. mtx3.unlock();
23. mtx\_log.lock();
24. tmp = "line 2: end " + to\_string(index) + " item ";
25. items.push\_back({currentTime(), tmp});
26. mtx\_log.unlock();
27. }
28. else
29. {
30. mtx3.lock();
31. que3.push({-1, -1});
32. mtx3.unlock();
33. finish = true;
34. }
35. }
36. mtx2.unlock();
37. }
38. }

Листинг 4: third\_line() - рабочий поток - третья лента:

1. void third\_line()
2. {
3. bool finish = false;
4. while(!finish)
5. {
6. mtx3.lock();
7. if (!que3.empty())
8. {
9. pair<int, int> cur\_item = que3.front();
10. int index = cur\_item.first;
11. int t0 = cur\_item.second;
12. que3.pop();
13. if (t0 != -1)
14. {
15. mtx\_log.lock();
16. string tmp = "line 3: start " + to\_string(index) + " item " + to\_string(t0) + " milliseconds";
17. items.push\_back({currentTime(), tmp});
18. mtx\_log.unlock();
19. Sleep(t0);
20. mtx4.lock();
21. result.push({index, t0});
22. mtx4.unlock();
23. mtx\_log.lock();
24. tmp = "line 3: end " + to\_string(index) + " item ";
25. items.push\_back({currentTime(), tmp});
26. mtx\_log.unlock();
27. }
28. else
29. {
30. finish = true;
31. }
32. }
33. mtx3.unlock();
34. }
35. }

### Экспериментальная часть

В данном разделе будут приведены примеры работы программы, а также будет приведена интерпретация сформированного программой лог-файла.

#### Пример работы программы

На рисунках 4.1.1 - 4.1.3 будут приведены примеры работы программы. В консоль выводятся входные данные (данные для первой ленты конвейера) и результат прохождения этих данных через конвейер.

|  |
| --- |
|  |
| *Рис. 4.1.1 - один входной элемент* |

|  |
| --- |
|  |
| *Рис. 4.1.2 - три входных элемента* |

|  |
| --- |
|  |
| *Рис. 4.1.5 - пять входных элементов* |

Как видно из приведенных результатов: один входной элемент, который должен быть обработан на всех трех лентах, обрабатывается последовательно и его суммарное время обработки равно 3 секунды \* 3 ленты = 9 секунд. При этом уже для трех входных объектах суммарное время конвейерной обработки становится гораздо меньше суммарного времени последовательной обработки: 14 секунд < 24 секунд. Аналогичная ситуация и для входной очереди из пяти элементов.

#### Анализ лог-файла

Для того, чтобы отследить работу потоков, необходимо получить лог-файл, отражающий время начала работы ленты над очередным заданием и время окончания этой работы.

Например, на рисунке 4.2.1 представлен лог-файл для ситуации, когда генератор подал на вход конвейеру пять элементов:

|  |
| --- |
|  |
| *Рис 4.2.1 - лог для 5 различных по времени входных элементов* |

В первом столбце показано время записи, во втором лента конвейера, на которой производилась запись, далее указано начало или конец обработки и номер элемента, который обрабатывался. Если это начало обработки, то в конце указывается количество миллисекунд, которые должен обрабатываться этот элемент. Из представленного файла видно, что как только первый входной объект 0 item был обработан на первой ленте, его начинает обрабатывать вторая лента. В это время на первой ленте уже идет обработка следующего элемента. Как только на второй ленте закончилась обработка 0 item, начинает работать третья лента конвейера. В данном эксперименте время обработки текущего элемента на всех трех лентах конвейера практически одинаковое, однако на вход подаются числовые значения в пределах от 1000 до 3000, поэтому ленты большую часть времени не простаивают в течение работы программы, кроме ситуации в самом начале, когда первый элемент еще не поступил в очередь на обработку ко второй и третьей лентам. Также в конце работы программы возникла ситуация, когда предыдущая задача 3 item занимала ~1 секунду, а следующая задача 4 item занимала уже ~2 секунды, поэтому третья лента конвейера простаивала секунду перед тем, как приступить к обработке последней задачи.

Рассмотрим другой пример, когда на вход подаются задачи, требующие на обработку одинакового количества времени, однако на второй ленте конвейера обработка будет занимать в 3 раза больше времени, чем на первой ленте и в 2 раза, чем на третьей. На рисунке 4.2.2 представлен лог работы программы.

|  |
| --- |
|  |
| *Рис 4.2.2 - лог для 5 одинаковых по времени входных элементов, но различных по времени обработки на каждой ленте* |

Как видно из лог-файла, 3 лента конвейера простаивает 2 секунды на каждом элементе, кроме кроме первого.

##### Вывод:

Оптимальным использование конвейерной обработки будет тогда, когда все поступающие на конвейер задачи обрабатываются примерно одинаковое количество времени и разница времени обработки на каждой из лент конвейера по возможности сведена к минимуму. Такое использование конвейера даст наибольший выигрыш по времени и производительности. В подтверждение этому далее на рисунке 4.2.3 приведен лог-файл, демонстрирующий отсутствие простаивающих лент конвейера, сформированный программой, соответствующей описанию выше.

|  |
| --- |
|  |
| *Рис 4.2.3 - лог для 5 одинаковых по времени входных элементов и практически одинаковых по времени обработки на каждой ленте* |

### Заключение

В ходе лабораторной работы был изучен и реализован вычислительный конвейер с использованием методов распараллеливания процессов. Были выявлены оптимальные для конвейерной обработки параметры входных задач: сбалансированное время обработки данных на всех лентах (уровнях) конвейера.