Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный технический университет
имени Н. Э. Баумана» (национальный исследовательский университет)

Дисциплина: «Анализ алгоритмов»

Отчет по лабораторной работе №5

Тема работы:
«Конвейерная обработка данных»

Студент: Волков Е. А.
Группа: ИУ7-55Б
Преподаватели: Волкова Л. Л.,
Строганов Ю. В.

Москва, 2019 г.

Содержание

Введение

Имеется большое количество важных задач, решение которых требует использования огромных вычислительных мощностей, зачастую недоступных для современных вычислительных систем.

Параллелизм может значительно ускорить решение таких задач. Особое место среди систем подобного рода занимает конвейерный обработчик. Он прост для понимания, ведь принцип его работы основан на реальном механизме непрервного действия. [3]

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

Задачи работы:

1) изучить алгоритм конвейеризации и алгоритм шифрования RSA;
2) реализовать конвейерную систему шифрования/дешифрования алгоритмом RSA;
3) провести журналирование событий в конвейерной обработке;
4) описать и обосновать полученные результаты в отчете о лабораторной работе, выполненного как расчётно-пояснительная записка.

1 Аналитический раздел

В данном разделе будет рассмотрены алгоритм конвейеризации и алгоритм шифрования RSA.

1.1 Описание алгоритмов

1.1.1 Алгоритм конвейерной обработки данных

Конвейеризация — это техника, в результате которой задача разбивается на некоторое число подзадач, которые выполняются последовательно. Конвейерный подход позволяет создавать эффективные параллельные реализации обычных неспециализированных алгоритмов [4].

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 [5].

Каждое звено конвейера выполняет следующие действия:

- получить данные,
- обработать данные,
- послать данные следующим звеньям.

1.1.2 Алгоритм шифрования RSA

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел [8].

Алгоритм находит пары чисел (e, n) и (d, n), являющиеся открытым и закрытым ключом соответственно. Открытым ключом зашифровывают сообщение, а закрытым – расшифровывают. Пара чисел закрытого ключа держится в секрете [8].

Шифруемый символ представляют в виде числа – его номера в алфавите или кода в таблице символов (например ASCII или Unicode).

Шифрование производится по формуле:

$$C_i = M_i^e \, mod \, n,$$

где (e, n) - открытый ключ, M_i - шифруемый символ, C_i - зашифрованный символ.

Расшифровка сообщения производится по формуле:

$$M_i = C_i^d \, mod \, n,$$

где (d, n) - закрытый ключ, M_i - расшифрованный символ, C_i - зашифрованный символ. [9].

Выводы

Рассмотрены алгоритм конвейерной обработки данных и алгоритм шифрования RSA, выделены их ключевые моменты.

2 Конструкторский раздел

В разделе приводятся схемы алгоритма конвейерной обработки данных и алгоритма RSA.

2.1 Разработка алгоритмов

На рис. 1 приведена схема общая схема конвейерной обработки.

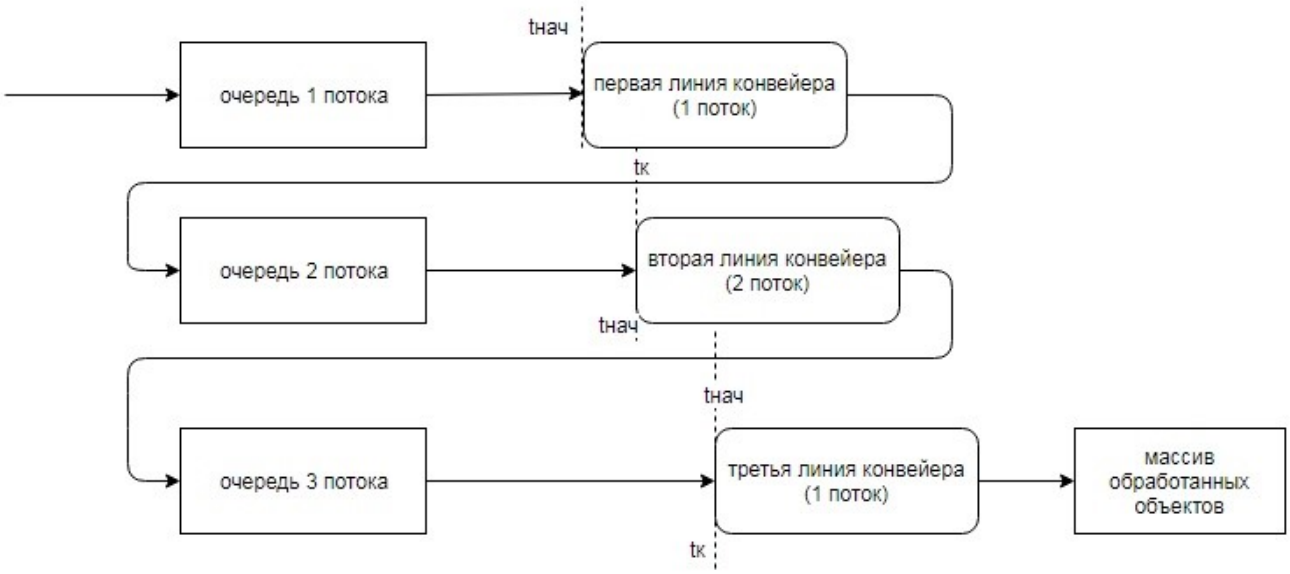

Рис. 1: Схема работы конвейера

Псевдокод главной функции:

```
1. Создать 3 потока, мьютекса и очереди
2. Заполнить массив объектов значениями от 1 до n
3. Добавить элементы массива в очередь 1
4. Применить join ко всем потокам
5. Записать информации из массива результатов в лог-файл
```

Псевдокод для ленты 1:

```
1. Бесконечный цикл
2.        Заблокировать мьютекс 1
3.        Если очередь 1 не пуста
4.              Извлечь элемент из очереди
5.              Записать информацию о начале работы
6.              Добавить элемент в результирующий массив
```

```
7.              Выполнить функции 1
8.              Добавить элемент в очередь 2
8.              Записать информацию об окончании работы
10.              Добавить элемент в результирующий массив
11.              Если элемент равен n
12.                    Завершить работу
13.        Разблокировать мьютекс 1
```

Псевдокод для ленты 2:

```
1. Бесконечный цикл
2.        Заблокировать мьютекс 2
3.        Если очередь 2 не пуста
4.              Извлечь элемент из очереди
5.              Записать информацию о начале работы
6.              Добавить элемент в результирующий массив
7.              Выполнить функции 2
8.              Добавить элемент в очередь 3
9.              Записать информацию об окончании работы
10.              Добавить элемент в результирующий массив
11.              Если элемент равен n
12.                    Завершить работу
13.        Разблокировать мьютекс 2
```

Псевдокод для ленты 3:

```
1. Бесконечный цикл
2.        Заблокировать мьютекс 3
3.        Если очередь 3 не пуста
4.              Извлечь элемент из очереди
5.              Записать информацию о начале работы
6.              Добавить элемент в результирующий массив
7.              Выполнить функции 3
8.              Записать информацию об окончании работы
9.              Добавить элемент в результирующий массив
10.              Если элемент равен n
11.                    Завершить работу
12.        Разблокировать мьютекс 3
```

На рисунках 2, 3 и 4 приведены схемы алгоритма RSA, разбитые на этапы для конвейера.

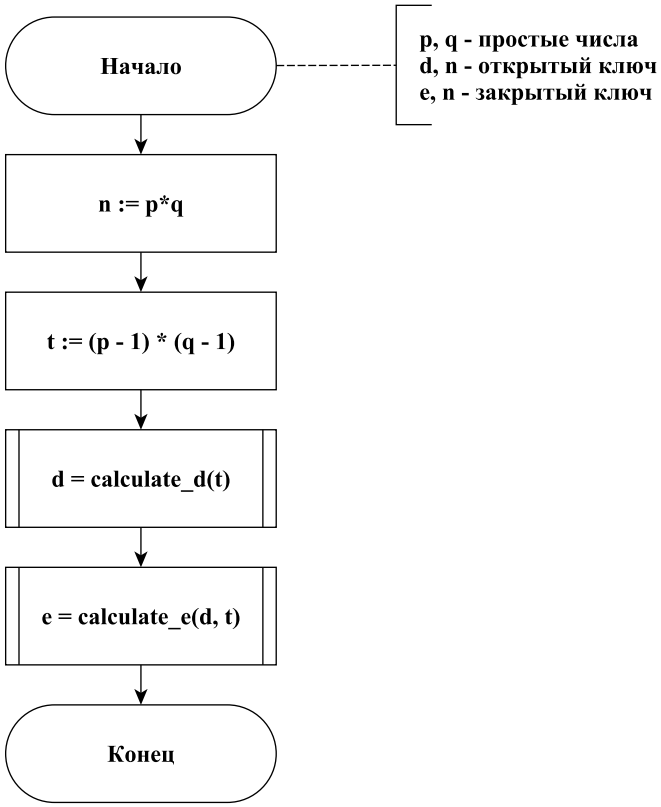

Рис. 2: Схема алгоритма генерации ключей

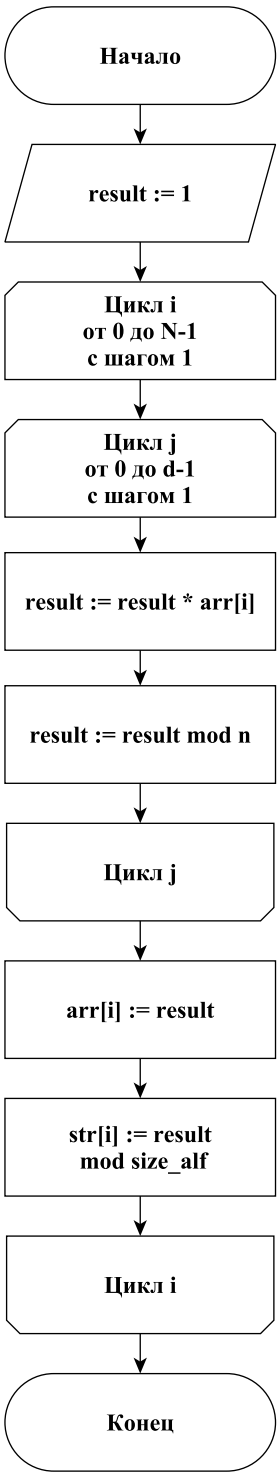

Рис. 3: Схема алгоритма шифрования строки

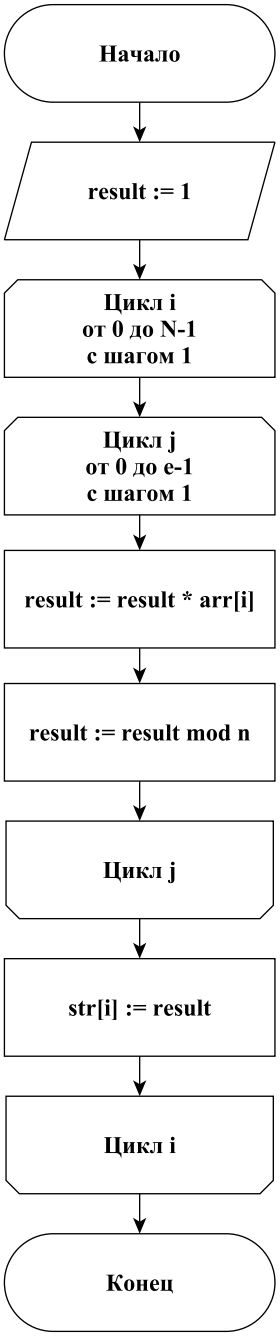

Рис. 4: Схема алгоритма дешифрования строки

Выводы

В разделе представлен псевдокод конвейерной обработки данных. Даны интерпретации главной функции и 3 функций, имитирующих работу лент.

3 Технологический раздел

Здесь описываются требования к программному обеспечению и средства реализации, приводятся листинги программы.

3.1 Требования к программному обеспечению

Входные данные:

1) целое положительное число - количество строк для шифрования;
2) целое положительное число - максимальная длина строки;
3) строка - имя файла логов.

Выходные данные: файл с логами.

На рис. 5 приведена функциональная схема конвейерной обработки данных.

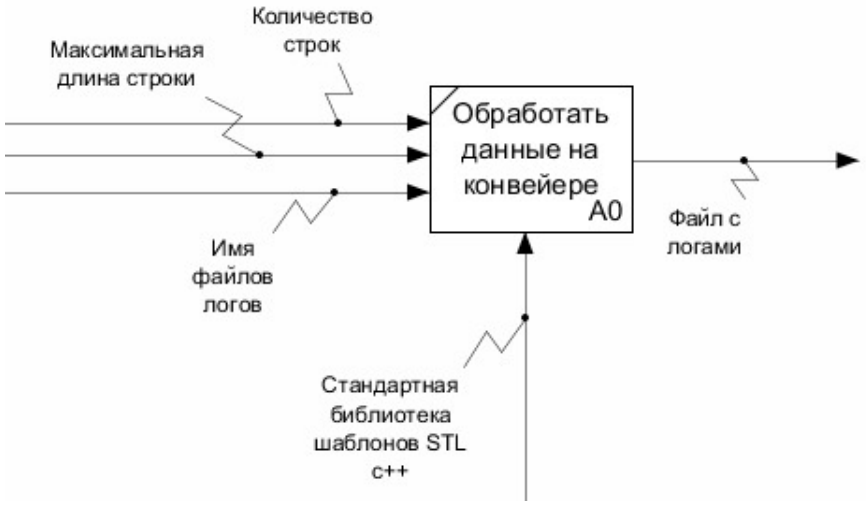

Рис. 5: Функциональная схема конвейерной обработки данных

3.2 Средства реализации

Программа написана на языке C++, т. к. этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов, обладает высокой эффективностью и значительным набором стандартных классов и процедур. В качестве среды разработки использовался фреймворк QT 5.13.1.

Для хранения массива, строк применяются контейнерные классы *std::vector*, *std::string* из стандартной библиотеки шаблонов STL. Кроме того, для реализации конвейера используются *std::thread* и *std::mutex*.

Замер времени производится с помощью функции *GetLocalTime* из библиотеки *windows.h*.

3.3 Листинг программы

Реализация конвейерной обработки данных представлена на листингах 1-6.

Листинг 1: Структура объекта конвейера

```
1  struct Conv_obj {
2      int index;
3      std::string str;
4      std::vector<long long int> vec;
5      Key public_key, private_key;
6  };
```

Листинг 2: Структура объекта для записи в файл логов

```
1  struct File_obj {
2      int index_str;
3      int num_belt;
4      SYSTEMTIME t;
5      char state;
6  };
```

Листинг 3: Главная функция конвейера

```
1  void conveyor() {
2      std::queue<Conv_obj> queue_1, queue_2, queue_3;
3      std::vector<std::string> vec_str;
4      std::vector<Conv_obj> vec_elem;
5      vec_init(vec_str, MAX_ELEM, len_str);
6      for (size_t i = 0; i < vec_str.size(); i++) {
7          Conv_obj tmp;
8          tmp.str = vec_str[i];
9          tmp.index = i + 1;
10         vec_elem.push_back(tmp);
11     }
12
13     for (size_t i = 0; i < vec_str.size(); i++) {
14         queue_1.push(vec_elem[i]);
15     }
16
17     std::thread thread_1(belt_1, std::ref(queue_1),
18                                          std::ref(queue_2));
```

```
19      std::thread thread_2(belt_2, std::ref(queue_2),
20                                              std::ref(queue_3));
21      std::thread thread_3(belt_3, std::ref(queue_3));
22      thread_1.join();
23      thread_2.join();
24      thread_3.join();
25
26      write_res();
27  }
```

Листинг 4: Лента 1

```
1   void belt_1(std::queue<Conv_obj> &queue_1,
2                   std::queue<Conv_obj> &queue_2) {
3       int count = 0;
4       while (count < MAX_ELEM) {
5           m1.lock();
6           if (!queue_1.empty()) {
7               Conv_obj cur_conv_obj = queue_1.front();
8               queue_1.pop();
9
10              File_obj cur_file_obj;
11              cur_file_obj.index_str = cur_conv_obj.index;
12              cur_file_obj.num_belt = 1;
13              GetLocalTime(&cur_file_obj.t);
14              cur_file_obj.state = 'S';
15
16              m.lock();
17              log_vec.push_back(cur_file_obj);
18              m.unlock();
19
20              handler_1(cur_conv_obj);
21
22              m2.lock();
23              queue_2.push(cur_conv_obj);
24              m2.unlock();
25
26              GetLocalTime(&cur_file_obj.t);
27              cur_file_obj.state = 'F';
28
29              m.lock();
30              log_vec.push_back(cur_file_obj);
31              m.unlock();
```

```
32
33                 count++;
34             }
35             m1.unlock();
36         }
37
38 }
```

Листинг 5: Лента 2

```
1  void belt_2(std::queue<Conv_obj> &queue_2,
2                  std::queue<Conv_obj> &queue_3) {
3      int count = 0;
4      while (count < MAX_ELEM) {
5          m2.lock();
6          if (!queue_2.empty()) {
7              Conv_obj cur_conv_obj = queue_2.front();
8              queue_2.pop();
9
10             File_obj cur_file_obj;
11             cur_file_obj.index_str = cur_conv_obj.index;
12             cur_file_obj.num_belt = 2;
13             GetLocalTime(&cur_file_obj.t);
14             cur_file_obj.state = 'S';
15
16             m.lock();
17             log_vec.push_back(cur_file_obj);
18             m.unlock();
19
20             handler_2(cur_conv_obj);
21
22             m3.lock();
23             queue_3.push(cur_conv_obj);
24             m3.unlock();
25
26             GetLocalTime(&cur_file_obj.t);
27             cur_file_obj.state = 'F';
28
29             m.lock();
30             log_vec.push_back(cur_file_obj);
31             m.unlock();
32
33             count++;
```

```
34            }
35            m2.unlock();
36        }
37
38    }
```

Листинг 6: Лента 3

```
1  void belt_3(std::queue<Conv_obj> &queue_3) {
2      int count = 0;
3      while (count < MAX_ELEM) {
4          m3.lock();
5          if (!queue_3.empty()) {
6              Conv_obj cur_conv_obj = queue_3.front();
7              queue_3.pop();
8
9              File_obj cur_file_obj;
10             cur_file_obj.index_str = cur_conv_obj.index;
11             cur_file_obj.num_belt = 3;
12             GetLocalTime(&cur_file_obj.t);
13             cur_file_obj.state = 'S';
14
15             m.lock();
16             log_vec.push_back(cur_file_obj);
17             m.unlock();
18
19             handler_3(cur_conv_obj);
20
21             GetLocalTime(&cur_file_obj.t);
22             cur_file_obj.state = 'F';
23
24             m.lock();
25             log_vec.push_back(cur_file_obj);
26             m.unlock();
27
28             count++;
29         }
30         m3.unlock();
31     }
32 }
```

Реализация обработчиков лент конвейера, реализующих шифрование строк алгоритмом SPA представлена на листингах 7-9..

Листинг 7: Обработчик 1 ленты. Вычисление открытого и закрытого ключей

```
1
2  void handler_1(Conv_obj &obj) {
3
4      long long int p, q, n, t, e, d;
5      p = obj.str.size();
6      while (!is_prime(p)) {
7          p++;
8
9      }
10     q = p + 1;
11     while (!is_prime(q)) {
12         q++;
13     }
14     std::cout << q << " " << p << "\n";
15     n = p * q;
16     t = (p - 1) * (q - 1);
17     e = calculate_e(t);
18     d = calculate_d(e, t);
19
20     obj.public_key.e = e;
21     obj.public_key.n = n;
22     obj.private_key.e = d;
23     obj.private_key.n = n;
24 }
25
26
27 long int greatest_common_divisor(long int e, long int t)
28 {
29     long int tmp;
30     while (e > 0)
31     {
32         long int tmp = e;
33         e = t % e;
34         t = tmp;
35     }
36
37     return t;
38 }
39
40 long int calculate_e(long int t)
41 {
```

```
42      long int res = -1;
43      long int e;
44
45      for ( e = 2; e < t; e++ )
46      {
47          if (greatest_common_divisor(e, t) == 1)
48          {
49              res = e;
50              break;
51          }
52      }
53
54      return res;
55  }
56
57
58  long int calculate_d(long int e, long int t)
59  {
60      long int d;
61      long int k = 1;
62
63      while (true)
64      {
65          k += t;
66
67          if (!(k % e))
68          {
69              d = k / e;
70              break;
71          }
72      }
73      std::cout << "d" << d;
74      return d;
75
76  }
77
78
79  bool is_prime(long int prime)
80  {
81      bool res = true;
82      long int j = (long int)sqrt((long double)prime);
83
```

```
84      for (long int i = 2; i <= j; i++)
85      {
86          if (!(prime % i))
87          {
88              res = false;
89              break;
90          }
91      }
92
93      return res;
94  }
```

Листинг 8: Обработчик 2 ленты. Шифрование строки.

```
1   void handler_2(Conv_obj &obj) {
2       std::string &str = obj.str;
3       for (size_t i = 0; i < str.size(); i++) {
4           obj.vec.push_back(encrypt(int(str[i]),
5                   obj.public_key.e, obj.public_key.n));
6           str[i] = obj.vec[i]  % alf.size() + 97;
7       }
8   }
9
10  long int encrypt( long int i, long int e, long int n)
11  {
12      long long int result = 1;
13
14      for ( long int j = 0; j < e; j++ )
15      {
16          result *= i;
17          result %= n;
18      }
19
20      return result;
21  }
```

Листинг 9: Обработчик 3 ленты. Расшифровка строки.

```
1   void handler_3(Conv_obj &obj) {
2       std::string &str = obj.str;
3       for (size_t i = 0; i < str.size(); i++) {
4           str[i] = encrypt(obj.vec[i], obj.private_key.e,
5                            obj.private_key.n);
6       }
```

```
7 }
```

Выводы

В данном разделе были рассмотрены требования к программному обеспечению, обоснован выбор средств реализации, приведены листинги программы.

4 Исследовательский раздел

В разделе представлены примеры выполнения программы, а также пояснения к некоторым файлам журналирования.

4.1 Примеры работы и анализ файлов журналирования

На рис. 6-7 приведены примеры работы программы.

Обозначения:
№ operation – номер события,
Belt – номер ленты,
Number str – номер обрабатываемой строки,
State – состояние обработки (S - начало, F - конец),
Time – время события в формате ч:м:с.мс.

№ operation	Belt	Number str	State	Time
1	1	1	S	16:29:37.55
2	1	1	F	16:29:37.56
3	2	1	S	16:29:37.56
4	2	1	F	16:29:37.66
5	3	1	S	16:29:37.67
6	3	1	F	16:29:41.115

Рис. 6: Обработка 1 элемента

На данном рисунке можно наблюдать поэтапную обработку для 1 элемента. Как видно, быстрее всего отрабатывает 1-я лента, а дольше всего - 3-я лента.

№ operation	Belt	Number str	State	Time
1	1	1	S	16:35:1.563
2	1	1	F	16:35:1.577
3	1	2	S	16:35:1.577
4	2	1	S	16:35:1.577
5	2	1	F	16:35:1.587
6	1	2	F	16:35:1.587
7	1	3	S	16:35:1.587
8	2	2	S	16:35:1.589
9	3	1	S	16:35:1.589
10	3	1	F	16:35:5.637
11	2	2	F	16:35:5.637
12	1	3	F	16:35:5.637
13	2	3	S	16:35:5.637
14	3	2	S	16:35:5.638
15	3	2	F	16:35:9.681
16	2	3	F	16:35:9.681
17	3	3	S	16:35:9.681
18	3	3	F	16:35:13.731

Рис. 7: Обработка 3 элементов

Такую же картинку наблюдаем при обработке 3 элементов. Записи лог-файлов подтверждают, что конвейер отрабатывает корректно.

Выводы

Программа успешно демонстрирует конвейерную обработку. Журналирование отражает информацию об этапах обработки.

Заключение

В ходе работы выполнено следующее:

1) изучены алгоритм конвейеризации и алгоритм шифрования RSA;
2) реализована конвейерная система;
3) проведено журналирование событий в конвейерной обработке;
4) описаны и обоснованы полученные результаты в отчете о лабораторной работе, выполненного как расчётно-пояснительная записка.

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.- М.:Техносфера, 2009.

[2] Д. Кнут. Искусство программирования, М., Мир, 1978

[3] Воеводин В.В. Математические модели и методы в параллельных процессах. М., 1986

[4] Погорелов Д.А. Применение конвейерной обработки данных на примере сортировки простыми вставками, М., Образование и наука в России и за рубежом 2019 .- Т. 49 , № 1

[5] Корнеев В.В. Параллельные вычислительные системы. М., 1999.

[6] ISO/IEC 14882:2017 [Электронный ресурс]. – Режим доступа: https://www.iso.org/standard/68564.html, свободный – (27.11.2019)

[7] <chrono> [Электронный ресурс]. – Режим доступа: http://www.cplusplus.com/reference/chrono/, свободный – (20.11.2019)

[8] RSA - шифрование на пальцах [Электронный ресурс]. – Режим доступа: http://www.michurin.net/computer-science/rsa.html, свободный – (17.12.2019)

[9] Алгоритм RSA [Электронный ресурс]. – Режим доступа: https://vscode.ru/prog-lessons/algoritm-rsa.html, свободный – (17.12.2019)