МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

# Конвейер

Работу выполнил: Подвашецкий Дмитрий, ИУ7-54Б

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

*Москва, 2019*

# Оглавление

# Введение

При обработке данных могут возникать ситуации, когда необходимо обработать множество данных последовательно несколькими алгоритмами. В этом случае удобно использовать конвейерную обработку данных.

Для решения конвейреным методом была выбрана задача декодирования шифра Цезаря на основе статистического анализа.

Всего имеется 3 ленты:

1. читает сообщение;
2. кодирует сообщение;
3. угадывает сдвиг шифра;

**Задачами** данной лабораторной являются:

1. изучение метода конвеерной обработки данных;
2. реализация данного метода;
3. экспериментальное подтверждение работостопобности алгоритма;

# 1 Аналитическая часть

## 1.1 Описание алгоритма

Сначала запускается 3 параллельных процесса, каждый из которых отвечает за определенную ленту. Каждому процессу присвоена своя очеред заданий. Процесс проверяет есть ли в очереди задания, если есть, то забирает его из очереди, обрабатывает и отправляет в следующую очередь, иначе ждет поступления задания.

После того как 3 ленты запустились, запускается параллельный процесс, отвечающий за генерацию заданий. Этот процесс создает задания и помещает их в первую очередь, тем самым запуская обработку.

Завершение каждой из трех конвейерных лент происходит в том случае, если время ожидания задания, в некоторый момент времени, превосходит удвоенное среднее время ожидание на этой ленте. Среднее время ожидания высчитывается по мере работы алгоритма.

Первая лента реализует чтение сообщения. Необходимо, чтобы в задании было указано имя файла, в котором находится сообщение.

Вторая лента реализует шифрование шифром Цезаря.[2]

Третья лента подсчитывает частоту встречи каждой буквы в сообщении, и сверяет со статистическими данными. [1] На основе сравнение выдвигается предположение о изначальном сдвиге шифрованного сообщения.

## Вывод

В данном разделе была описана схема работы конвейера и реализуемая задача.

# 2 Конструкторская часть

## 2.1 Схема алгоритмов

### 2.1.1 Схема работы ленты

Так как все 3 ленты однотипны, далее будет описан общий принцип работы каждой ленты.

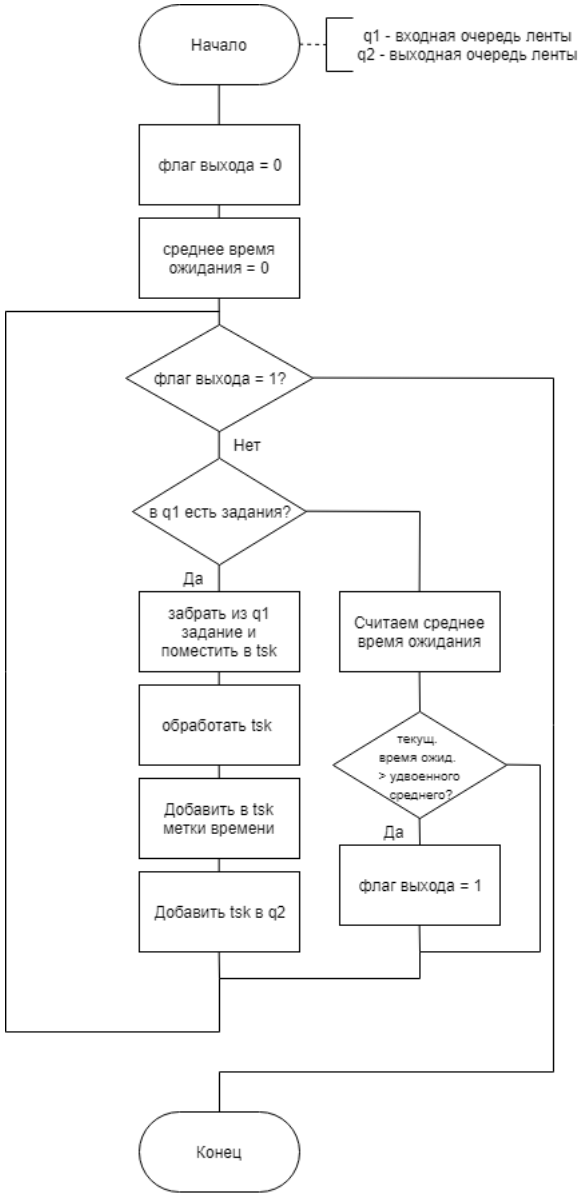

Рис.1. Схема работы ленты

## Вывод

В данном разделе была разработана схема работы ленты.

# 3 Технологическая часть

## 3.1 Выбор ЯП

В качестве языка программирования был выбрал C++, так как он позволяет реализовать задачу максимально эффективно.

## 3.2 Замеры времени

Замер времени работы алгоритмов производился при помощи функций из библиотеки <chorno>.[3] Для реализации потоков была выбрана библиотека <thread>.[4] Для реализации раздленного доступа к ресурсам использовалась библиотека <mutex>.[5]

## 3.3 Требования к ПО

**Требования к выводу:**

1. информация о временных метках каждого задания;

**Требования к программе:**

1. Корректный вывод, программа не должна аварийно завершаться

**Требования структуре входного файла:** В считываемом файле должны быть буквы Анлгийского алфавита. Так же допуситимы знаки пробела, запятой и точки.

## 3.4 Сведения о модулях программы

Программа состоит из:

- main.cpp - главный файл программы
- conveyor.cpp - файл с реализацией конвейера(Листинг 3.1.)

## 3.5 Листинг кода

Листинг 3.1. Реализация конвейера.

```
#include "conveyor.h"

static std::chrono::time_point<std::chrono::steady_clock> start;

void first_tape(std::queue<task> &first_queue, std::mutex &first_mutex,
                                    std::queue<task> &second_queue, std::mutex &second_mutex)
{
        bool exit = false;
        bool waiting = false;
        std::chrono::time_point<std::chrono::steady_clock> start_awaiting_time;

        long long total_waiting_time = 0;
        int amount_of_waitings = 0;

        while(!exit)
        {
                first_mutex.lock();
                if (first_queue.size() > 0)
                {
                        auto cur_task = first_queue.front();
                        first_queue.pop();

                        auto startft = get_time(start);
                        first_task_part(cur_task);
                        auto endft = get_time(start);

                        cur_task.first_tape_start = startft;
                        cur_task.first_tape_end = endft;

                        first_mutex.unlock();

                        second_mutex.lock();
                        second_queue.push(cur_task);
                        second_mutex.unlock();

                        if (waiting)
                        {
```

```
38                                      waiting = false;
39                                      total_waiting_time +=
                                           get_time(
                                           start_awaiting_time);
40                                      amount_of_waitings += 1;
41                              }
42                      }
43                      else
44                      {
45                              first_mutex.unlock();
46
47                              if (!waiting)
48                              {
49                                      waiting = true;
50                                      start_awaiting_time = std
                                           ::chrono::steady_clock
                                           ::now();
51                              }
52                              else if (amount_of_waitings > 0 &&
                                    get_time(start_awaiting_time)
                                   > 2*(total_waiting_time/
                                   amount_of_waitings))
53                              {
54                                      exit = true;
55                              }
56                      }
57              }
58      }
59
60      void first_task_part(task &inTask)
61      {
62              std::ifstream file(inTask.fileName);
63
64              if (file.is_open())
65              {
66                      char c;
67                      while (file.get(c))
68                      {
69                              inTask.msg += c;
70                      }
71              }
72
73              file.close();
74      }
75
76      void second_tape(std::queue<task> &second_queue, std::
           mutex &second_mutex,
```

```
77                                      std::queue<task> &
                                           third_queue, std::
                                           mutex &third_mutex)
78 {
79         bool exit = false;
80         bool waiting = false;
81         std::chrono::time_point<std::chrono::steady_clock>
               start_awaiting_time;
82
83         long long total_waiting_time = 0;
84         int amount_of_waitings = 0;
85
86         while(!exit)
87         {
88                 second_mutex.lock();
89                 if (second_queue.size() > 0)
90                 {
91                         auto cur_task = second_queue.front
                            ();
92                         second_queue.pop();
93
94                         auto starts = get_time(start);
95                         second_task_part(cur_task);
96                         auto ends = get_time(start);
97
98                         cur_task.second_tape_start =
                            starts;
99                         cur_task.second_tape_end = ends;
100
101                        second_mutex.unlock();
102
103                        third_mutex.lock();
104                        third_queue.push(cur_task);
105                        third_mutex.unlock();
106
107                        if (waiting)
108                        {
109                                waiting = false;
110                                total_waiting_time +=
                                   get_time(
                                   start_awaiting_time);
111                                amount_of_waitings += 1;
112                        }
113                }
114                else
115                {
116                        second_mutex.unlock();
```

```
117
118                         if (!waiting)
119                         {
120                                 waiting = true;
121                                 start_awaiting_time = std
                                       ::chrono::steady_clock
                                       ::now();
122                         }
123                         else if (amount_of_waitings > 0 &&
                                     get_time(start_awaiting_time)
                                    > 2*(total_waiting_time/
                                    amount_of_waitings))
124                         {
125                                 exit = true;
126                         }
127                 }
128         }
129 }
130
131 void second_task_part(task &inTask)
132 {
133         std::string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ
               ";
134
135         int encode_num = 3;
136         inTask.encoded_num = encode_num;
137
138         for (int i = 0; i < inTask.msg.size(); i++)
139         {
140                 if (inTask.msg[i] != '␣' || inTask.msg[i]
                       != '\n' || inTask.msg[i] != '.' ||
                       inTask.msg[i] != ',')
141                 {
142                         for (int j = 0; j < alphabet.size
                               (); j++)
143                         {
144                                 if (toupper(inTask.msg[i])
                                     == alphabet[j])
145                                 {
146                                         if (isupper(inTask
                                               .msg[i]))
147                                                 inTask.msg
                                                       [i] =
                                                       alphabet
                                                       [(j +
                                                       encode_num
                                                       ) %
```

```
                                                                alphabet
                                                                .size()
                                                                ];
148                                                     else
149                                                             inTask.msg
                                                                [i] =
                                                                tolower
                                                                (
                                                                alphabet
                                                                [(j +
                                                                encode_num
                                                                ) %
                                                                alphabet
                                                                .size()
                                                                ]);
150                                                     break;
151                                             }
152                                     }
153                             }
154             }
155 }
156
157 void third_tape(std::queue<task> &third_queue, std::mutex
       &third_mutex,
158                                     std::vector<task> &
                                           complited_tasks, std::
                                           mutex &com_t_mutex)
159 {
160             bool exit = false;
161             bool waiting = false;
162             std::chrono::time_point<std::chrono::steady_clock>
                    start_awaiting_time;
163
164             long long total_waiting_time = 0;
165             int amount_of_waitings = 0;
166
167             while(!exit)
168             {
169                     third_mutex.lock();
170                     if (third_queue.size() > 0)
171                     {
172                             auto cur_task = third_queue.front
                                   ();
173                             third_queue.pop();
174
175                             auto starttd = get_time(start);
176                             third_task_part(cur_task);
```

```
177                         auto endtd = get_time(start);
178
179                         cur_task.third_tape_start =
                               starttd;
180                         cur_task.third_tape_end = endtd;
181
182                         third_mutex.unlock();
183
184                         com_t_mutex.lock();
185                         complited_tasks.push_back(cur_task
                               );
186                         com_t_mutex.unlock();
187
188                         if (waiting)
189                         {
190                                 waiting = false;
191                                 total_waiting_time +=
                                       get_time(
                                       start_awaiting_time);
192                                 amount_of_waitings += 1;
193                         }
194                 }
195                 else
196                 {
197                         third_mutex.unlock();
198
199                         if (!waiting)
200                         {
201                                 waiting = true;
202                                 start_awaiting_time = std
                                       ::chrono::steady_clock
                                       ::now();
203                         }
204                         else if (amount_of_waitings > 0 &&
                                    get_time(start_awaiting_time)
                                   > 2*(total_waiting_time/
                                   amount_of_waitings))
205                         {
206                                 exit = true;
207                         }
208                 }
209         }
210 }
211
212 void third_task_part(task &inTask)
213 {
214         std::vector<std::pair<char, double>> analisys;
```

```
215         analisys.push_back(std::pair<char, double>('A',
                0.081));
216         analisys.push_back(std::pair<char, double>('B',
                0.014));
217         analisys.push_back(std::pair<char, double>('C',
                0.027));
218         analisys.push_back(std::pair<char, double>('D',
                0.039));
219         analisys.push_back(std::pair<char, double>('E',
                0.13));
220         analisys.push_back(std::pair<char, double>('F',
                0.029));
221         analisys.push_back(std::pair<char, double>('G',
                0.02));
222         analisys.push_back(std::pair<char, double>('H',
                0.052));
223         analisys.push_back(std::pair<char, double>('I',
                0.065));
224         analisys.push_back(std::pair<char, double>('J',
                0.002));
225         analisys.push_back(std::pair<char, double>('K',
                0.004));
226         analisys.push_back(std::pair<char, double>('L',
                0.034));
227         analisys.push_back(std::pair<char, double>('M',
                0.025));
228         analisys.push_back(std::pair<char, double>('N',
                0.072));
229         analisys.push_back(std::pair<char, double>('O',
                0.079));
230         analisys.push_back(std::pair<char, double>('P',
                0.020));
231         analisys.push_back(std::pair<char, double>('R',
                0.069));
232         analisys.push_back(std::pair<char, double>('S',
                0.061));
233         analisys.push_back(std::pair<char, double>('T',
                0.105));
234         analisys.push_back(std::pair<char, double>('U',
                0.024));
235         analisys.push_back(std::pair<char, double>('V',
                0.009));
236         analisys.push_back(std::pair<char, double>('W',
                0.015));
237         analisys.push_back(std::pair<char, double>('X',
                0.002));
238         analisys.push_back(std::pair<char, double>('Y',
```

```
            0.019));
239         analisys.push_back(std::pair<char, double>('Z',
            0.001));
240
241         std::string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ
            ";
242
243         std::vector<std::pair<char, double>> taskAnalisys;
244         int taskMsgLen = 0;
245
246         for (int i = 0; i < alphabet.size(); i++)
247                 taskAnalisys.push_back(std::pair<char,
                    double>(alphabet[i], 0));
248
249         for (int i = 0; i < inTask.msg.size(); i++)
250         {
251                 auto c = inTask.msg[i];
252                 if (c != '␣' || c != '\n' || c != '.' || c
                        != ',')
253                 {
254                         for (int j = 0; j < taskAnalisys.
                            size(); j++)
255                         {
256                                 if (toupper(c) ==
                                    taskAnalisys[j].first)
257                                 {
258                                         taskAnalisys[j].
                                            second += 1;
259                                 }
260                         }
261
262                         taskMsgLen += 1;
263                 }
264         }
265
266         for (int i = 0; i < taskAnalisys.size(); i++)
267         {
268                 taskAnalisys[i].second /= taskMsgLen;
269         }
270
271         double closest = 10.0;
272         int decode = -1;
273
274         for (auto i : taskAnalisys)
275         {
276                 for (auto j : analisys)
277                 {
```

```
278                         if (fabs(i.second - j.second) <
                               closest)
279                         {
280                                 closest = fabs(i.second -
                                       j.second);
281                                 decode = abs(i.first - j.
                                       first);
282                         }
283                 }
284         }
285
286         inTask.decoded_num = decode;
287 }
288
289 void task_pusher(std::queue<task> &first_queue, std::mutex
        &first_mutex)
290 {
291         for (int i = 1; i < 16; i++)
292         {
293                 task newTask;
294                 newTask.id = i;
295                 newTask.fileName = "msg";
296                 newTask.fileName += std::to_string(1 + i %
                        5);
297                 newTask.fileName += ".txt";
298
299                 first_mutex.lock();
300                 first_queue.push(newTask);
301                 first_mutex.unlock();
302         }
303 }
304
305 long long get_time(std::chrono::time_point<std::chrono::
       steady_clock> &start_time)
306 {
307         auto end = std::chrono::steady_clock::now();
308         auto result = std::chrono::duration_cast<std::
               chrono::microseconds>(end-start_time).count();
309
310         return (result % 1000000000);
311 }
312
313 void printTaskInfo(task &inTask)
314 {
315         std::cout << "Task␣id:␣" << inTask.id << std::endl
               ;
316         std::cout << "␣␣1st␣tape␣st:␣" << inTask.
```

```
            first_tape_start << "; 1st tape end: " <<
            inTask.first_tape_end << std::endl;
317         std::cout << "  2nd tape st: " << inTask.
            second_tape_start << "; 2nd tape end: " <<
            inTask.second_tape_end << std::endl;
318         std::cout << "  3d tape st: " << inTask.
            third_tape_start << "; 3d tape end: " << inTask
            .third_tape_end << std::endl;
319         std::cout << "  Encoded num: " << inTask.
            encoded_num << "; Decodec num: " << inTask.
            decoded_num << std::endl << std::endl;
320 }
```

## Вывод

В данном разделе были реализованных необходимые алгоритмы.

# 4 Экспериментальная часть

## 4.1 Постановка эксперимента

Создадим 5 заданий и посмотрим временные метки от каждой ленты.

## 4.2 Пример работы

На рисунке ниже представлены временные метки каждого обработанного задания.

1. Task Id: N - номер задания
2. N tape st/end - время начала/конца обработки на N ленте
3. encoden num - сдвиг при кодировке
4. deconded num - предпологаемый сдвиг

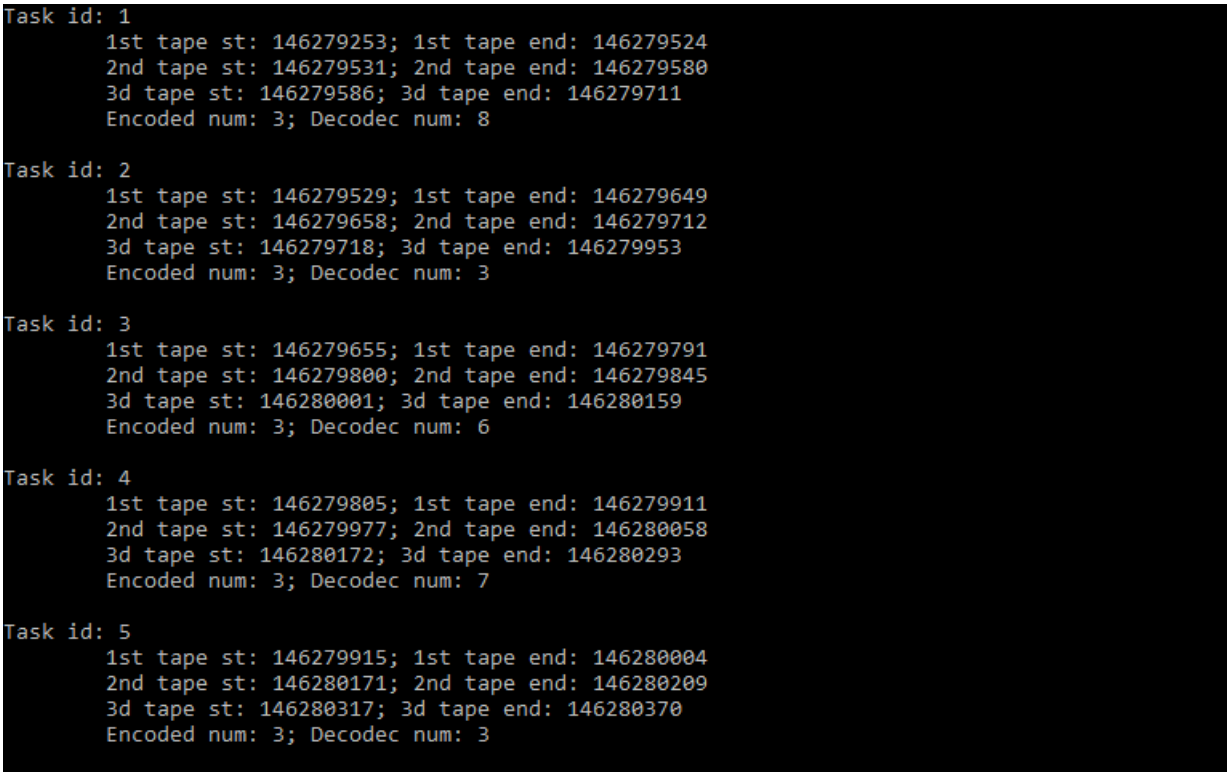


Рис. 2. Временные метки обработанных заданий.

# Вывод

На основе приведенных выше (Рис. 2.) временных меток обрабоки каждого задания, можно сделать вывод что конвейер работает правильно, т.е. время начала обработки всегда меньше времени конца, и так же видно как после выхода задания с ленты, на эту ленту поступает другое задание.

Задание 1 поступило на первую ленту в 146279253 мкс, вышло в 146279524 мкс. Задание 1 поступило на вторую ленту в 146279531 мкс, в тоже время Задание 2 поступило на первую ленту в 146279529.

Так же можно заметить, что данный алгоритм угадывает зашифрованное сообщение в 2 из 5 случаев, что не является хорошим результатом. Для улучшения этого результата можно увеличить объем изходного текста, так как алгоритм завязан на статичтическом анализе.

# Заключение

В результате данной лабораторной работы было:

1. изучен метод конвейерной обработки данных;
2. реализован данный метод;
3. эксперементально подтверждено корректная работа алгоритма (Рис. 2.);

На Рис. 2. можно увидеть, что Задание 1 поступило на первую ленту в 146279253 мкс, вышло в 146279524 мкс. Задание 1 поступило на вторую ленту в 146279531 мкс, в тоже время Задание 2 поступило на первую ленту в 146279529.

# Список литературы


1. Основы криптоанализа. [Электронный ресурс] Режим доступа: http://www.univer.omsk.su/o
   old/Edu/infpro/1/kodir/kripan.html Последнняя дата обращения: 17.12.2019

2. Шифр Цезаря. [Электронный ресурс] Режим доступа: http://kriptografea.narod.ru/chezar.htn
   Последнняя дата обращения: 17.12.2019

3. <chrono>. [Электронный ресурс] Режим доступа: http://www.cplusplus.com/reference/chrono
   Последнняя дата обращения: 17.12.2019

4. std::thread. [Электронный ресурс] Режим доступа: https://ru.cppreference.com/w/cpp/thread/
   Последнняя дата обращения: 17.12.2019

5. std::mutex. [Электронный ресурс] Режим доступа: https://ru.cppreference.com/w/cpp/thread/
   Последнняя дата обращения: 17.12.2019