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: Левушкин И. К.
Группа: ИУ7-52Б
Преподаватели: Волкова Л. Л.,
Строганов Ю. В.

Москва, 2019 г.

Содержание

Введение **2**

1 Аналитический раздел **4**

1.1 Описание алгоритмов 4

1.1.1 Алгоритм конвейерной обработки данных 4

Выводы 4

2 Конструкторский раздел **5**

2.1 Разработка алгоритмов 5

Выводы 6

3 Технологический раздел **7**

3.1 Требования к программному обеспечению 7

3.2 Средства реализации 7

3.3 Листинг программы 8

Выводы 14

4 Исследовательский раздел **15**

4.1 Примеры работы и анализ файлов журналирования 15

Выводы 16

Заключение **16**

Список литературы **16**

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;
2) реализовать конвейерную систему;
4) описать и обосновать полученные результаты в отчете о лабораторной работе, выполненного как расчётно-пояснительная записка.

1 Аналитический раздел

В данном разделе будет рассмотрен алгоритм конвейеризации.

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.

Выводы

Рассмотрен алгоритм конвейерной обработки данных, выделены его ключевые моменты.

2 Конструкторский раздел

В разделе приводятся схемы выбранных алгоритмов сортировки, производится их теоретический сравнительный анализ.

2.1 Разработка алгоритмов

На рис. 1 приведена схема общая схема конвейерной обработки.

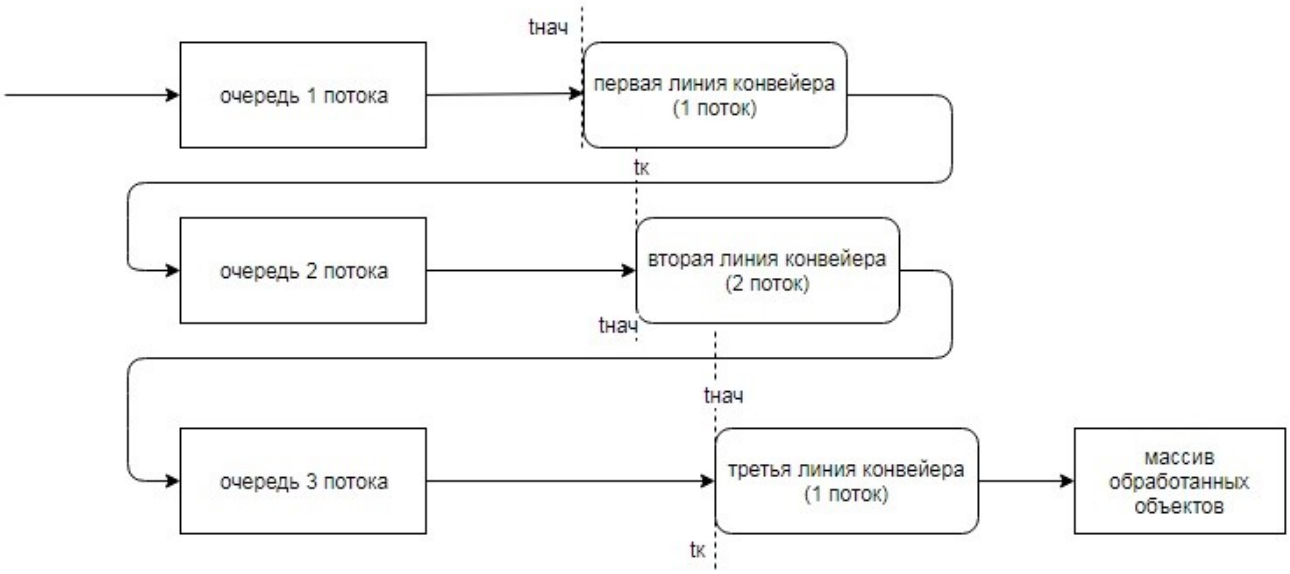

Рис. 1: Схема работы конвейера

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

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

Необходимо реализовать класс object, который будет в себе содержать данные, над которыми будут работать конвейеры и соответственно сами конвейеры (методы класса), которые будут применяться над объектом.

Сами данные объекта будут представлять собой динамический массив, над которым будут совершаться некоторые действия.

В качестве первого конвейера было решено взять сортировку Шелла. Во втором конвейере происходит добавление 10 * $len(mas)$ элементов к текущему массиву. И в третьем происходит вывод на экран 1% всех элементов массива.

Входные данные: нет

Выходные данные:

- номер объекта;
- номер конвейера;
- время начала выполнения на первом конвейере;
- время конца выполнения на первом конвейере;
- время начала выполнения на втором конвейере;
- время конца выполнения на втором конвейере;
- время начала выполнения на третьем конвейере;
- время конца выполнения на третьем конвейере;

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования C++ [10]. Проект был выполнен в среде QT Creator [8]. Для измерения процессроного времени была использована ассемблерная инструкция rdtsc [9].

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 *clock_ gettime* из библиотеки *time.h*.

3.3 Листинг программы

Реализованная программа представлена в листингах 1-5.

Листинг 1: Структура класса object

```
#include <vector>
#include <iostream>
#include <ctime>

#define DEFAULT_SIZE 10

using namespace std;

struct time_for_work
{
timespec from;
timespec to;
};

class object
{
public:
object();
object(size_t count);
object(vector<int> mas);

void add_elem_to_mas(const int value);

void delete_last_elem_from_mas();

void do_conveyer_1();

void do_conveyer_2();

void do_conveyer_3();

void set_time_1(time_for_work t1) {this->t1 = t1;}
void set_time_2(time_for_work t2) {this->t2 = t2;}
void set_time_3(time_for_work t3) {this->t3 = t3;}

time_for_work get_time_1();
time_for_work get_time_2();
time_for_work get_time_3();

vector<int> get_mas();


~object() = default;

private:
vector<int> mas;
time_for_work t1;
time_for_work t2;
time_for_work t3;
};
```

Листинг 2: Реализация класса object

```
#include "object.h"


object::object()
{
this->mas.resize(DEFAULT_SIZE);
for (size_t i = 0; i < this->mas.size(); i++)
{
this->mas[i] = rand();
}
}

object::object(size_t count)
{
this->mas.resize(count);
for (size_t i = 0; i < this->mas.size(); i++)
{
this->mas[i] = rand();
}
}

object::object(vector<int> mas)
{
this->mas = mas;
}

void object::add_elem_to_mas(const int value)
{
this->mas.push_back(value);
}

void object::delete_last_elem_from_mas()
{
this->mas.pop_back();
}

void ShellSort (vector<int>& array)
{
int tmp;
int size = array.size();
for (int step = size / 2; step > 0; step /= 2)
{
for (int i = step; i < size; i++)
{
for (int j = i - step; j >= 0 && array[size_t(j)] > array[size_t(j + step)];
    j -= step)
{
tmp = array[size_t(j)];
array[size_t(j)] = array[size_t(j + step)];
array[size_t(j + step)] = tmp;
}
}
}
}
```

```
54
55 void add_elements(vector<int> &mas)
56 {
57 int count_elements = DEFAULT_SIZE * 10;
58 for (int i = 0; i < count_elements; i++)
59 {
60 mas.push_back(rand());
61 }
62 }
63
64 void print_few_elements(const vector<int> &mas)
65 {
66 size_t count_print_elements = mas.size() / 100;
67 for (size_t i = 0; i < count_print_elements; i++)
68 {
69 cout << mas[i] << " ";
70 if (i % 10 == 0)
71 {
72 cout << endl;
73 }
74 }
75 cout << endl;
76 }
77
78 void object::do_conveyer_1()
79 {
80 ShellSort(mas);
81 }
82
83 void object::do_conveyer_2()
84 {
85 add_elements(mas);
86 }
87
88 void object::do_conveyer_3()
89 {
90 print_few_elements(mas);
91 }
92
93 time_for_work object::get_time_1()
94 {
95 return t1;
96 }
97
98 time_for_work object::get_time_2()
99 {
100 return t2;
101 }
102
103 time_for_work object::get_time_3()
104 {
105 return t3;
106 }
107
108 vector<int> object::get_mas()
```

```
109 {
110 return mas;
111 }
```

Листинг 3: Лента 1

```
1  void conveyer_1()
2  {
3  timespec now, after;
4  while (1)
5  {
6  m1.lock();
7  if (!q1.empty())
8  {
9  auto object = q1.front();
10 q1.pop();
11
12 clock_gettime(CLOCK_REALTIME, &now);
13 object.do_conveyer_1();
14 clock_gettime(CLOCK_REALTIME, &after);
15
16 object.set_time_1({now, after});
17
18 m2.lock();
19 q2.push(object);
20 m2.unlock();
21 counter_1++;
22 if (counter_1 == count_objects)
23 {
24 return;
25 }
26 }
27 m1.unlock();
28 }
29 }
```

Листинг 4: Лента 2

```
1  void conveyer_2()
2  {
3  timespec now, after;
4  while (1)
5  {
6  m2.lock();
7  if (!q2.empty())
8  {
9  auto object = q2.front();
10 q2.pop();
11
12 clock_gettime(CLOCK_REALTIME, &now);
13 object.do_conveyer_2();
14 clock_gettime(CLOCK_REALTIME, &after);
15
16 object.set_time_2({now, after});
17
18 m3.lock();
```

```
q3.push(object);
m3.unlock();
counter_2++;
if (counter_2 == count_objects)
{
return;
}
}
m2.unlock();
}
}
```

Листинг 5: Лента 3

```
void conveyer_3(vector<object> &return_objects)
{
timespec now, after;
while (1)
{
m3.lock();
if (!q3.empty())
{
auto object = q3.front();
q3.pop();

clock_gettime(CLOCK_REALTIME, &now);
object.do_conveyer_3();
clock_gettime(CLOCK_REALTIME, &after);


object.set_time_3({now, after});

return_objects.push_back(object);
counter_3++;
if (counter_3 == count_objects)
{
return;
}
}
m3.unlock();
}
}
```

Листинг 6: main

```
mutex m1, m2, m3;
queue<object> q1, q2, q3;
size_t count_objects = 100;
size_t counter_1 = 0;
size_t counter_2 = 0;
size_t counter_3 = 0;

vector<object> objects_generator(size_t count_objects)
{
vector<object> mas_objects;
for (size_t i = 0; i < count_objects; i++)
```

```
12 {
13 object *obj = new object(i * 100);
14 mas_objects.push_back(*obj);
15 }
16 return mas_objects;
17 }
18
19 void print_time(time_for_work tim)
20 {
21 long ms_from, ms_to; // Milliseconds
22 long long ns_from, ns_to; // Nanoseconds
23 time_t s_from, s_to;  // Seconds
24 tm *timeinfo_from, *timeinfo_to;
25
26 s_from  = tim.from.tv_sec;
27 timeinfo_from = localtime (&s_from);
28 ms_from = floor(tim.from.tv_nsec / 1.0e6); // Convert nanoseconds to
    milliseconds
29
30 s_to  = tim.to.tv_sec;
31 timeinfo_to = localtime (&s_to);
32 ms_to = floor(tim.to.tv_nsec / 1.0e6); // Convert nanoseconds to
    milliseconds
33
34 ns_from = abs(tim.from.tv_nsec - ms_from * 1.0e6);
35
36 ns_to = abs(tim.to.tv_nsec - ms_to * 1.0e6);
37
38
39
40 cout << "Start: " << timeinfo_from->tm_hour <<
41 ":" << timeinfo_from->tm_min << ":" <<
42 timeinfo_from->tm_sec << ":" << ms_from << ":" << ns_from << endl;
43 cout << "Ending: " << timeinfo_to->tm_hour <<
44 ":" << timeinfo_to->tm_min << ":" <<
45 timeinfo_to->tm_sec << ":" << ms_to << ":" << ns_to << endl;
46 }
47
48 void print_results(vector<object> res)
49 {
50 time_for_work tim;
51
52 for (size_t i = 0; i < res.size(); i++)
53 {
54 cout << "Object " << i << ":" << endl;
55
56 cout << "Konveyer 1: " << endl;
57 tim = res[i].get_time_1();
58 print_time(tim);
59
60 cout << "Konveyer 2: " << endl;
61 tim = res[i].get_time_2();
62 print_time(tim);
63
64 cout << "Konveyer 3: " << endl;
```

```
65 tim = res[i].get_time_3();
66 print_time(tim);
67 }
68 }
69
70 int main()
71 {
72 vector<object> mas_objects = objects_generator(count_objects);
73
74 vector<object> return_objects;
75
76 vector<thread> thr(3);
77
78 thr[0] = thread(conveyer_1);
79 thr[1] = thread(conveyer_2);
80 thr[2] = thread(conveyer_3, ref(return_objects));
81
82 for (size_t i = 0; i < count_objects; i++)
83 {
84 q1.push(mas_objects[i]);
85 }
86
87
88 for (size_t i = 0; i < thr.size(); i++)
89 {
90 thr[i].join();
91 }
92
93 print_results(return_objects);
94
95 return 0;
96 }
```

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.

4.1 Примеры работы и анализ файлов журналирования

На рис. 2 приведен пример работы программы.

Обозначения:
Конвейер № – номер события,
Начало – время начала события в формате ч:м:с.мс.нс.
Конец – время окончания события в формате ч:м:с.мс.нс.

```
Объект 0:
Конвейер 1:
Начало: 10:50:41:143:548000
Конец: 10:50:41:143:549000
Конвейер 2:
Начало: 10:50:41:143:583000
Конец: 10:50:41:143:648000
Конвейер 3:
Начало: 10:50:41:143:701000
Конец: 10:50:41:143:845000
Объект 1:
Конвейер 1:
Начало: 10:50:41:143:589000
Конец: 10:50:41:143:607000
Конвейер 2:
Начало: 10:50:41:143:693000
Конец: 10:50:41:143:703000
Конвейер 3:
Начало: 10:50:41:143:879000
Конец: 10:50:41:143:892000
Объект 2:
Конвейер 1:
Начало: 10:50:41:143:689000
Конец: 10:50:41:143:729000
Конвейер 2:
Начало: 10:50:41:143:944000
Конец: 10:50:41:143:973000
Конвейер 3:
Начало: 10:50:41:144:0
Конец: 10:50:41:144:33000
```

Рис. 2: Пример работы программы

На рисунке можно наблюдать поэтапную обработку для 3 элементов. Первый объект обслуживается последовательно, то есть вторая и третья ленты

ждут пока первая лента обработает первый объект. Затем первая лента начинает обслуживать второй объект в то время как вторая лента начинает обслуживать первый объект и так далее. Время простоя для каждой ленты соответствует ожидаемой величине.

Выводы

Программа успешно демонстрирует конвейерную обработку. Из рисунка можно наблюдать поэтапную обработку для 3 элементов. Время простоя для каждой ленты соответствует ожидаемой величине. Все три ленты работают параллельно, следовательно последовательная обработка будет менее эффективна нежели конвейерная обработка.

Заключение

В ходе работы выполнено следующее:

1) изучен алгоритм конвейеризации;
2) реализована конвейерная система;
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] QT Creator Manual [Электронный ресурс]. – Режим доступа: https://doc.qt.io/qtcreator/index.html, свободный. (Дата обращения: 29.09.2019 г.)

[9] Microsoft «rdtsc» [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/intrinsics/rdtsc?view=vs-2019, свободный. (Дата обращения: 29.09.2019 г.)

[10] ISO/IEC JTC1 SC22 WG21 N 3690 «Programming Languages — C++» [Электронный ресурс]. – Режим доступа: https://devdocs.io/cpp/, свободный. (Дата обращения: 29.09.2019 г.)