**Федеральное государственное бюджетное образовательное учреждение   
высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Анализ алгоритмов**

**Лабораторная работа №6**

**Отчёт на тему:**

# «Конвейеры»

Выполнила:

Янова Даниэлла

ИУ7-53

**Москва, 2018**

**Введение**

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

**Описание**

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.

Идея заключается в параллельном выполнении нескольких инструкций процессора. Сложные инструкции процессора представляются в виде последовательности более простых стадий. Вместо выполнения инструкций последовательно (ожидания завершения конца одной инструкции и перехода к следующей), следующая инструкция может выполняться через несколько стадий выполнения первой инструкции. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца [1].

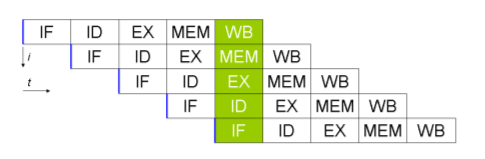


Рис. 1: Простой пятиуровневый конвейер в RISC-процессорах

На иллюстрации 1 показан простой пятиуровневый конвейер в RISCпроцессорах.

Здесь:

1. IF (англ. Instruction Fetch) — получение инструкции;
2. ID (англ. Instruction Decode) — раскодирование инструкции;
3. EX (англ. Execute) — выполнение;
4. MEM (англ. Memory access) — доступ к памяти;
5. WB (англ. Register write back) — запись в регистр.

Вертикальная ось — последовательные независимые инструкции, горизонтальная — время. Зелёная колонка описывает состояние процессора в один момент времени, в ней самая ранняя, верхняя инструкция уже находится в состоянии записи в регистр, а самая последняя, нижняя инструкция — только в процессе чтения.

**Применение**

Выполнение каждой команды складывается из ряда последовательных этапов (шагов стадий), суть которых не меняется от команды к команде. С целью увеличения быстродействия процессора и максимального использования всех его возможностей в современных микропроцессорах используется конвейерный принцип обработкии нформации. 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. По очередному тактовому импульсу каждая команда в конвейере продвигается на следующую стадию обработки, выполненная команда покидает конвейер, а новая поступает в него [2].

**Виды**

Перечислим основные типы конвейеров:

1) Арифметический конвейер (arithmetic pipeline) — реализация в АЛУ поэтапного исполнения арифметических операций чаще всего над вещественными числами [3];

2) супер-конвейер, гипер-конвейер, глубокий конвейер (super-pipeline, hyperpipeline, deep pipeline) — вычислительный конвейер с необычно большим количеством стадий. Например, процессор Intel Pentium 4 имел 20 стадий конвейера, а в модификации Prescott получил конвейер из 31 стадии [4];

3) недозагруженный конвейер (underutilized pipeline) — конвейер, в котором в одно и то же время не все стадии конвейера выполняют какуюто операцию. Например ранние процессоры MIPS имели 6-стадийный конвейер, но в каждый момент было занято только 3 его стадии.

**Цель работы:** изучение алгоритма работы конвейера с использованием методов распараллеливания процессов.

1. **Аналитическая часть**

В данном разделе приведено описание алгоритма конвейера.

**1.1 Описание алгоритма.**

На вход алгоритм получает некоторое число. Начальная функция пробрасывает число на первый уровень (уровень А).

Уровень А: для числа n вычисляется n!, после чего к числу добавляется рандомное число от 0 до n!-1. Полученный результат m пробрасывается на уровень B.

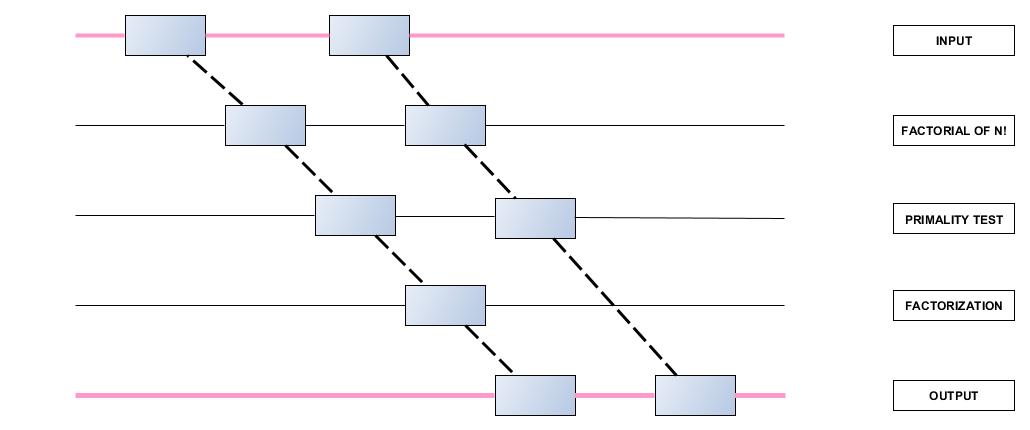
Уровень В: для числа m вычисляется, является ли данное число простым. Если число является простым, оно подается на выход, если нет – пробрасывается на уровень С.

Уровень С: для числа k, полученного на вход, производится факторизация – разбиение числа на множители. Максимальный из полученных множителей подается на выход.

2. **Конструкторская часть**

В данном разделе представлена схема конвейера.

* 1. **Схема конвейера**

****

**Рисунок 1.1 – Схема работы конвейера**

**3. Технологическая часть**

В данном разделе представлена реализация конвейера, указан язык программирования, а также необходимые модули.

**3.1 Средства реализации**

Для выполнения данной лабораторной работы использовался язык Python 3.7.1 в среде Pycharm. Замены времени проводились с использованием функции process\_time\_ns, входящей в библиотеку time Python версии 3.7.

**3.2 Листинг кода**

1. **from** random **import** randint
2. **import** threading
3. **import** time
5. **def keyf(e):**
6. **return** e[1]
8. **def** log\_sort(log, n):
9. res = []
10. **for i in range(1,n+1):**
11. tmp = []
12. **for** j **in** log:
13. **if** int((j[1].split())[0]) == i:
14. tmp.append(j)
15. **tmp.sort**
17. **for** j **in** tmp:
18. res.append(j)
19. **return** res
21. **def** get\_log(log1, log2, log3):
22. log = []
23. **for** i **in** log1:
24. log.append(i)
25. **for i in log2:**
26. log.append(i)
27. **for** i **in** log3:
28. log.append(i)
29. log.sort()
30. **return log**
32. **def** factorial(n):
33. **if** n == 0:
34. **return** 1
35. **else:**
36. **return** n\*factorial(n-1)
38. **def** plus\_random(n):
39. **return** n+randint(1,n\*2-1)
41. **def** check\_prime(n):
42. **for** i **in** range(2, n//2+1):
43. **if** n % i == 0:
44. **return** False
45. **return True**
47. **def** next\_prime(n):
48. n += 1
49. **if** check\_prime(n):
50. **return n**
51. **return** next\_prime(n)
53. **def** factorize(n):
54. res = []
55. **if check\_prime(n) == True:**
56. res.append(n)
57. **else**:
58. tmp = n
59. i = 2
60. **while tmp >= i:**
61. **if** tmp % i == 0:
62. res.append(i)
63. tmp //= i
64. **else**:
65. **i = next\_prime(i)**
67. **return** res
69. mutex1 = threading.Lock()
70. **mutex2 = threading.Lock()**
71. mutex3 = threading.Lock()
72. fst\_q = []
73. snd\_q = []
74. trd\_q = []
75. **fin\_q = []**
77. **def** fst\_conv(n):
78. **global** fst\_q
79. **global** log1
80. **global tb**
82. **for** i **in** range(n):
83. tmp = randint(1,8)
84. log1.append([time.perf\_counter()-tb, '{} объект был создан первым конвеером'.format(i+1)])
85. **mutex1.acquire()**
86. fst\_q.append(tmp)
87. mutex1.release()
88. log1.append([time.perf\_counter()-tb, '{} объект помещен в очередь второго конвеера'.format(i+1)])
89. mutex1.acquire()
90. **fst\_q.append(-1)**
91. mutex1.release()
92. **return**
94. **def** snd\_conv():
95. **global fst\_q**
96. **global** snd\_q
97. **global** log2
98. **global** tb
99. i = 0
101. **while** True:
102. time.sleep(10\*\*-5)
103. **if** len(fst\_q)>0:
104. **if** fst\_q[0] == -1:
105. **mutex2.acquire()**
106. snd\_q.append(-1)
107. mutex2.release()
108. **return**
109. tmp = 0
111. mutex1.acquire()
112. tmp = fst\_q[0]
113. fst\_q = fst\_q[1:]
114. mutex1.release()
115. **log2.append([time.perf\_counter()-tb, '{} объект был взят на обработку вторым конвеером'.format(i+1)])**

118. tmp = plus\_random(factorial(tmp))
120. **mutex2.acquire()**
121. snd\_q.append(tmp)
122. mutex2.release()
123. log2.append([time.perf\_counter()-tb, '{} объект помещен в очередь третьего конвеера'.format(i+1)])
124. i+=1

127. **def** trd\_conv():
128. **global** snd\_q
129. **global** trd\_q
130. **global log3**
131. **global** tb
132. j = 0
133. **while** True:
134. time.sleep(10\*\*-5)
135. **if len(snd\_q)>0:**
136. **if** snd\_q[0] == -1:
137. **return**
138. tmp = 0
140. **mutex2.acquire()**
141. tmp = snd\_q[0]
142. snd\_q = snd\_q[1:]
143. mutex2.release()
144. log3.append([time.perf\_counter()-tb,'{} объект был взят на обработку третьим конвеером'.format(j+1)])
145. **tmp = factorize(tmp)**
147. mutex3.acquire()
148. trd\_q.append(tmp)
149. mutex3.release()
150. **log3.append([time.perf\_counter()-tb,'{} объект помещен в очередь четвертого конвеера'.format(j+1)])**
151. j+=1
153. n = 100
154. tb = time.perf\_counter()
155. **log1 = []**
156. log2 = []
157. log3 = []
159. t1 = threading.Thread(target=fst\_conv, args=(n,))
160. **t2 = threading.Thread(target=snd\_conv)**
161. t3 = threading.Thread(target=trd\_conv)
163. t3.start()
164. t2.start()
165. **t1.start()**
167. t1.join()
168. t2.join()
169. t3.join()
171. choice = 0
172. **print**('1. Вывести обычный лог')
173. **print**('2. Вывести лог, сортированный по объекту')
174. **print**('3. Вывести лог для определенного объекта')
176. choice = int(input('Выберите команду: '))
177. **while** choice < 1 **or** choice > 3:
178. choice = int(input('Выберите команду: '))
179. log = get\_log(log1, log2, log3)
180. **f = open('log.txt', 'w')**
182. **if** (choice >1):
183. log = log\_sort(log, n)
184. **if** choice == 3:
185. **i = int(input('Выберите объект от 1 до {}: '.format(n)))**
186. bot = 0
187. top = 0
188. **for** k **in** range(0,len(log)-1):
189. **if** int((log[k][1].split())[0]) == i **and** int((log[k+1][1].split())[0]) != i:
190. **top = k**
191. **if** int((log[k][1].split())[0]) != i **and** int((log[k+1][1].split())[0]) == i:
192. bot = k+1
193. log = log[bot:top+1]

196. **for** i **in** log:
197. f.write('{:.4f} :'.format(i[0])+i[1]+'**\n**')
198. f.close()

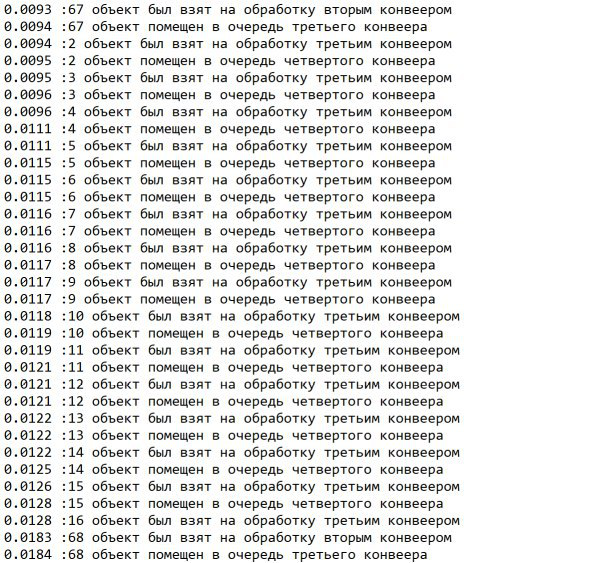
201. **print**(trd\_q)

Листинг 1.1 – Реализация функций потоков

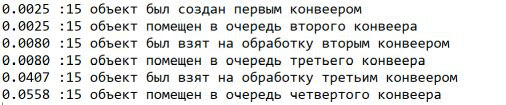
1. **Экспериментальная часть**

В данном разделе будут произведены замеры времени работы алгоритмов. Для исследования скоростных характеристик был использован компьютер на базе процессора Intel Core i7-6600U, содержащий 16 гигабайт оперативной памяти. Модуль тестирования запускался с жестокого диска под операционной системой Windows. Жесткий диск имел среднюю скорость передачи данных при чтении 2629 Мбайт/с и 798 Мбайт/с при записи.

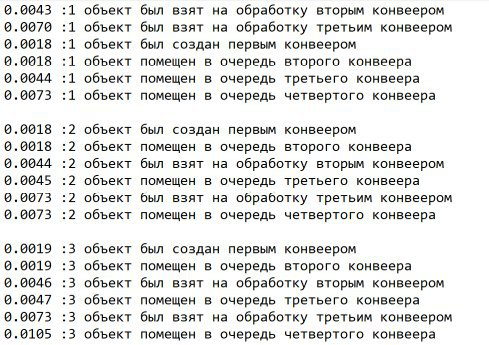
**4.1 Тестирование алгоритмов**



**Рисунок 4.1 – Пример лога программы**



**Рисунок 4.2 – Пример лога программы для определенного объекта**



**Рисунок 4.3 – Пример лога программы с сортировкой по объектам**

**Заключение**

В ходе выполнения лабораторной работы был изучен алгоритм работы конвейера с использованием методов распараллеливания процессов.

**Список литературы:**

[1] Конвейерная обработка данных <https://studfiles.net/preview/1083252/page:25/>.

[2] Описание RISC-процессора http://www.metods-rgrtu.ru/index.php/metods-3300-3399/309-3330.

[3] Арифметический конвейер <https://studfiles.net/preview/364715/page:3/>.

[4] Суперконвейер https://studopedia.su/15132364superkonveyernie − protsessori.html.