Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ №5

КОНВЕЕРНАЯ ОБРАБОТКА ДАННЫХ

по дисциплине:

«Анализ алгоритмов»

Руководитель: ст. преп. каф. ИУ7 _____ Волкова Л.Л.

Исполнитель: студ. гр. ИУ7-55Б _____ Муравьев И.А.

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Конвейеризация (конвейерная обработка) – способ организации вычислений, основанный на разделении подлежащей исполнению функции на более мелкие задачи, называемые ступенями (этапами) конвеера, и выделении для каждой из них отдельного блока. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько задач.

В данной работе рассматривается конвейеризация алгоритма медленного хеширования извлеченных из исходной таблицы базы данных паролей с последующим их сохранением в итоговую таблицу базы.

Целью данной работы является изучение и реализация методов асинхронного взаимодействия потоков для конвейерной обработки данных на материале указанного выше алгоритма.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) Разработать схему конвейерной обработки для указанного алгоритма.
2) Получить практические навыки реализации последовательной и конвеерной реализаций.
3) Провести сравнительный анализ последовательной и конвейерной реализации по затрачиваемым ресурсам (времени работы).

1 Аналитическая часть

В данном разделе рассматриваются решаемая задача и выбранный способ её конвейеризации.

1.1 Описание решаемой задачи

Главная часть решаемой задачи состоит в медленном шифровании паролей пользователей. Но перед этим пароли необходимо извлечь из базы данных, а после шифрования сохранить их в эту базу. То есть задача естестественно разбита на 3 этапа. Эти этапы и будут выделены в стадии конвейера.

1.2 Выделенные стадии конвейерной обработки

Было выделено три стадии конвейера:

1) Загрузка логина и пароля пользователя из базы данных.
2) Медленное шифрование пароля.
3) Вставка логина и зашифрованного пароля в базу данных.

1.2.1 Загрузка логина и пароля из базы данных

На данном этапе логин и незашифрованный пароль пользователя загружаются из базы данных.

1.2.2 Медленное шифрование пароля

Медленное шифрование (хеширование) – способ шифрования паролей, основанный на последовательном нахождении хеша от значения, полученного на предыдущей стадии шифрования. Чем больше таких стадий (итераций), тем дольше происходит шифрование. Такой способ усложняет проведение брутфорс атаки (перебор значений) для нахождения пароля выбранного пользователя ввиду низкой скорости проверки каждого пароля.

На данном этапе осуществляется медленное хеширование извлеченного из базы пароля.

1.2.3 Загрузка в базу данных

На данном этапе исходный логин и зашифрованный пароль сохраняются в базу данных.

1.3 Требования к программному обеспечению

К рассмотренному алгоритму выдвигаются следующие требования:

- Входные данные:

    1) База данных с исходной и итоговой таблицами. Таблицы состоят из двух колонок строкового типа (login, password).

- Выходные данные:

    1) Время работы при последовательном выполнения действий (вещественное число).
    2) Время работы конвейера (вещественное число).
    3) Минимальное, среднее и максимальное времена ожидания заявки в очереди каждой из конвейерных лент (вещественные числа).

Вывод

Был рассмотрена решаемая задача. Были определены и описаны этапы конвейерной реализации алгоритма. Были выдвинуты требования к разрабатываемому ПО.

2 Конструкторская часть

Данный раздел содержит схемы конвейерной обработки данных, последовательного и конвейерного алгоритма.

2.1 Схемы алгоритмов

В данном пункте раздела представлены схемы реализуемых в работе алгоритмов.

На рисунке 2.1 представлена схема организации конвейерных вычислений на примере конвейера с тремя лентами.

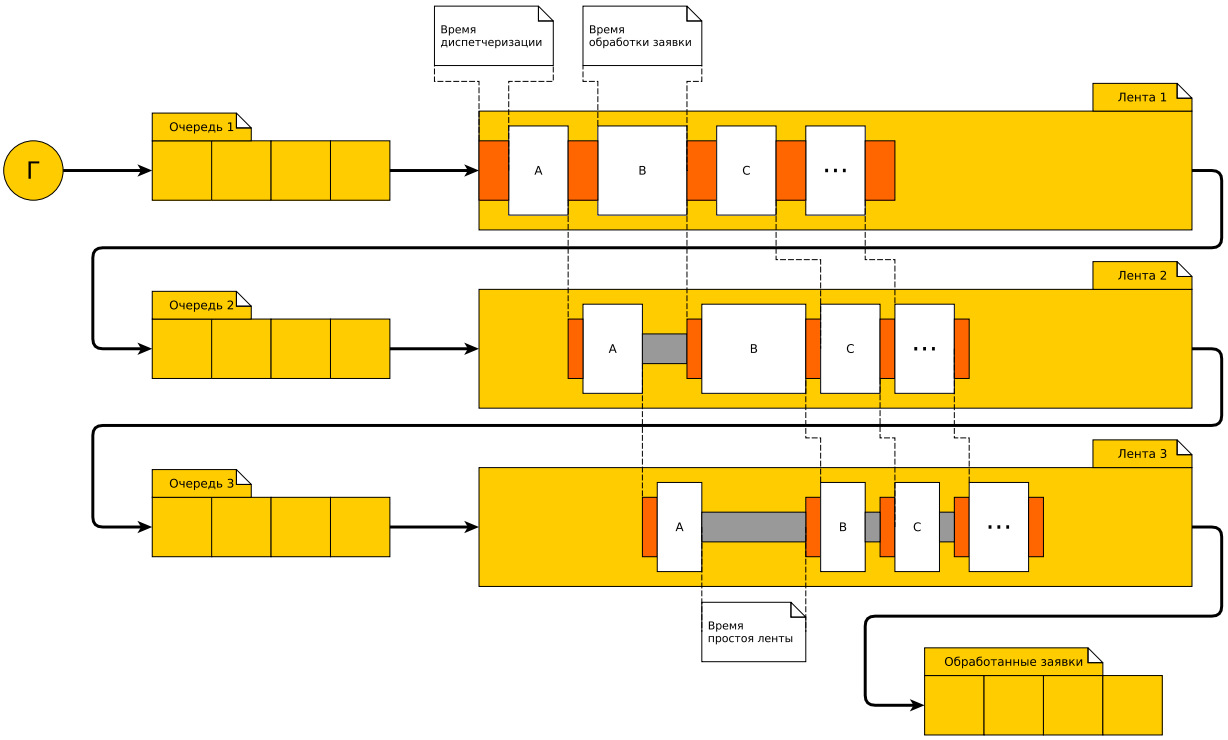

Рис. 2.1. Схема организации конвейера с тремя лентами

2.2 Структуры данных

Для удобства работы были выделены следующие классы:

1) UserStats

- поля
  - times - массив из объектов класса Times, содержащий время поступления и выхода для каждой ленты конвейера;
  - user - объект класса User, содержащий данные пользователя;
  - current - хэш-значение пароля;
  - number - идентификатор объекта класса;

- методы
  - set_time - установка времени входа/выхода для конкретной ленты;
  - get_time - получение времени входа/выхода для конкретной ленты;

2) User

- поля
  - login - логин пользователя;
  - password - пароль пользователя;
- методы
  - generate_pass - генерирует случайный пароль;

3) Time

- поля
  - time_in - время поступления на ленту ковейера;
  - time_out - время выхода с ленты конвейера;
- методы
  - set - установка времени поступления/выхода;
  - get - получение времени поступления/выхода;

Таким образом, программа генерирует заданное в программе количество пользователей и их данных, затем формирует 4 очереди и 3 ленты конвейера и запускает конвейерную обработку. После окончания выводится время конвейерной и последовательной обработки одних и тех же данных, после выводится лог, содержащий время поступления в очередь и выхода задач для каждой ленты конвейера.

Вывод

В данном разделе приведена схема работы конвейера, выделены структуры данных для реализации в технологическим разделе.

3 Технологическая часть

Данный раздел содержит обоснование выбора языка программирования и среды разработки, а также программную реализацию сконструированных алгоритмов.

3.1 Средства реализации

Для реализации программы был выбран язык программирования Python [1]. Такой выбор обусловлен следующими причинами:

- имеются стандартные библиотеки для работы с потоками, хеширования и измерения времени,
- высокая скорость разработки.

В качестве функции для шифрования будет использоваться криптографический хеш SHA-512. На момент написания работы это хеш без коллизий.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.8 представлены реализации рассматриваемых алгоритмов.

Листинг 3.1. Класс User

```
        PC = 'qwertyuiopasdfghjklzxcvbnm1234567890'
        PCS = len(PC)

        class User:
        def __init__(self, login: str = None, password: str =
    None):
        if login is None:
        self.__login = str(uuid.uuid4())
        else:
        self.__login = login
        if password is None:
        self.__password = self.__generate_pass()
        else:
        self.__password = password

        @staticmethod
        def __generate_pass():
        size = random.randint(16, 2**9)
        a = ''
        for i in range(size):
        a += PC[random.randint(0, PCS - 1)]
        return a

        @property
        def login(self):
        return self.__login

        @property
        def password(self):
        return self.__password
```

Листинг 3.2. Класс Time

```
class Time:
def __init__(self):
self.__time_in = None
self.__time_out = None

def set(self, t, is_in: bool):
if is_in:
if self.__time_in is not None:
raise Exception("InTime already recorded!")
self.__time_in = t
elif not is_in:
if self.__time_out is not None:
raise Exception("OutTime already recorded!")
self.__time_out = t
else:
raise Exception("is_in needed!")

def get(self, is_in: bool = None):
if is_in is None:
if self.__time_out is None and self.__time_in is None:
raise Exception("Time's not set!")
elif self.__time_out is None:
raise Exception("Can't get out_time!")
elif self.__time_out is None:
raise Exception("Can't get out_time!")
return [self.__time_in, self.__time_out]
elif is_in:
if self.__time_out is None:
raise Exception("Can't get in_time!")
return self.__time_in
else:
if self.__time_out is None:
raise Exception("Can't get out_time!")
return self.__time_out
```

Листинг 3.3. Класс UserStats

```
        class UserStats:
        cnt = 0

        def __init__(self, login: str = None, password: str =
    None,      do_not_init: bool = True):
        self.__times = [Time() for _ in range(3)]
        self.__number = self.cnt
        if do_not_init:
        self.user = None
        else:
        self.user = User(login, password)
        self.current = ''
        UserStats.cnt += 1

        def set_time(self, time_, stage: int, is_in: bool):
        # is_in - True, in, False, out
        # stage - номер ленты
        if stage < 0 or stage > 2 or stage is None:
        raise Exception("Wrong num!")
        self.__times[stage].set(time_, is_in)

        def get_time(self, is_in: bool = None, stage: int = None)
    :
        # is_in - True, False, out
        # stage - num, None = all

        if stage is not None and (stage < 0 or stage > 2):
        raise ExceptionНеверное(" число!")
        if stage is None:
        if is_in is None:
        return [self.__times[i].get() for i in range(3)]
        else:
        return [self.__times[i].get(is_in) for i in range(3)]
        else:
        return self.__times[stage].get(is_in)

        def get_number(self):
        return self.__number
```

Листинг 3.4. Файл master.py (часть 1)

```
        def init_db(con: sqlite3.Connection):
        cur = con.cursor()
        cur.execute('create table if not exists users_out (login
    text, password text)')
        cur.execute('create table if not exists users_in (i int,
    login text, password text)')
        con.commit()

        def fill_input_db(con: sqlite3.Connection, count=30):
        cur = con.cursor()
        for index in range(count):
        m = User()
        cur.execute(f'insert into users_in (i, login, password)
    values ({index}, "{m.login()}", "{m.password()}")')
        con.commit()

        def get_list_size_from_db(con: sqlite3.Connection) -> int
    :
        cur = con.cursor()
        cur.execute('select count(*) from users_in')
        return cur.fetchone()[0]

        def clear_db(con: sqlite3.Connection):
        cur = con.cursor()
        cur.execute('drop table if exists users_out')
        cur.execute('drop table if exists users_in')
        con.commit()

        def generate_users(users_count: int) -> [UserStats]:
        users = []

        for i in range(users_count):
        users.append(UserStats(do_not_init=False))

        return users
```

Листинг 3.5. Файл master.py (часть 2)

```
        def load_user(u: UserStats):
        con = sqlite3.connect('app.db')
        c = con.cursor()
        c.execute(f'select login, password from users_in where i
    = {u.get_number()}')
        login, password = c.fetchone()
        u.user = User(login, password)

        def get_hashed(u: UserStats):
        u.current = u.user.login() + 'my secret key'

        for iter in range(2000):
        u.current = sha512((u.current + str(iter)).encode('Utf
    -8')).hexdigest()

        def insert(u: UserStats):
        con = sqlite3.connect('app.db')
        c = con.cursor()
        c.execute(f"insert into users_out (login, password)
    values ('{u.user.login()}', '{u.current}')")
        con.commit()
        con.close()

        def tex_table(stats, stages):
        print("\\hline")
        print('Stage N & Task M & Start Time & End Time\\\\')
        print("\\hline")

        for stat_num, stat in enumerate(stats):
        for stage in range(stages):
        times = stat.get_time(stage=stage)
        print(
        f'Stage: {stage + 1} & Task: {stat_num + 1} & {times[0] -
     start_time:.6f} & {times[1] - start_time:.6f} \\\\')

        print("\\hline")
```

Листинг 3.6. Файл master.py (часть 3)

```
        def job(task: Callable, in_queue: SimpleQueue, out_queue:
    SimpleQueue, stage: int):
        while True:
        data: UserStats = in_queue.get()

        if data is None:
        out_queue.put(data)
        break

        data.set_time(time.time(), stage, True)
        task(data)
        data.set_time(time.time(), stage, False)
        out_queue.put(data)

        def test_serial(users):
        for user in users:
        load_user(user)
        get_hashed(user)
        insert(user)

        if __name__ == '__main__':
        stages_count = 3

        connection = sqlite3.connect('app.db')
        clear_db(connection)
        init_db(connection)
        fill_input_db(connection)

        _users = generate_users(get_list_size_from_db(connection)
    )

        pipeline_time = 0
        serial_time = 0
        cnt = 10
        for i in range(cnt):
        clear_db(connection)
        init_db(connection)
        fill_input_db(connection)

```

```
38          passwords_queue = SimpleQueue()
39          salt_queue = SimpleQueue()
```

Листинг 3.7. Файл master.py (часть 4)

```
        hash_queue = SimpleQueue()
        result_queue = SimpleQueue()

        add_salter = Process(target=job, args=(load_user,
    passwords_queue, salt_queue, 0))
        hasher = Process(target=job, args=(get_hashed, salt_queue
    , hash_queue, 1))
        inserter = Process(target=job, args=(insert, hash_queue,
    result_queue, 2))
        pipeline = [add_salter, hasher, inserter]

        for u in _users:
        passwords_queue.put(u)

        passwords_queue.put(None)
        start_time = time.time()
        for worker in pipeline:
        worker.start()

        for worker in pipeline:
        worker.join()
        end_time = time.time()
        pipeline_time += end_time - start_time

        start_time_ = time.time()
        test_serial(_users)
        end_time_ = time.time()
        serial_time += end_time_ - start_time_

        pipeline_time /= cnt

        print(f'Pipeline time = {pipeline_time * 1e6} mks')

        serial_time /= cnt
        print(f'Serial time = {serial_time * 1e6} mks')
        print('Press to get log')
        input()

        stats = []
```

Листинг 3.8. Файл master.py (часть 5)

```
        while not result_queue.empty():
        stats.append(result_queue.get())
        stats.pop()

        deltas = [[], [], []]
        for stat in stats:
        for stage in range(stages_count):
        stage_stat = stat.get_time(stage=stage)
        deltas[stage].append(stage_stat[1] - stage_stat[0])

        for stage in range(stages_count):
        print(f'Max time on stage {stage + 1} = {max(deltas[stage
    ]) * 1e6} mks')
        print(f'Min time on stage {stage + 1} = {min(deltas[stage
    ]) * 1e6} mks')
        print(f'Avg time on stage {stage + 1} = {sum(deltas[stage
    ]) / len(deltas[stage]) * 1e6} mks\n')

        connection.close()

        tex_table(stats, stages_count)
```

Вывод

В данном разделе была реализована конвейерная обработка данных: извлечение из базы данных, хеширование и сохранение паролей пользователей.

4 Экспериментальная часть

В данном разделе сравниваются реализованные алгоритмы, дается сравнительная оценка затрат на время.

4.1 Пример работы программы

Пример работы программы представлен на рисунке 4.1.

```
Pipeline time = 133182.73544311523 mks
Serial time = 228827.65769958496 mks
Press to get log

Max time on stage 1 = 812.7689361572266 mks
Min time on stage 1 = 133.99124145507812 mks
Avg time on stage 1 = 244.4903055826823 mks

Max time on stage 2 = 12578.487396240234 mks
Min time on stage 2 = 3556.489944458008 mks
Avg time on stage 2 = 9844.144185384115 mks

Max time on stage 3 = 3536.4627838134766 mks
Min time on stage 3 = 2316.9517517089844 mks
Avg time on stage 3 = 3023.9184697469077 mks

\hline
Stage N & Task M & Start Time & End Time\\
\hline
Stage: 1 & Task: 1 & 0.002179 & 0.002991 \\
Stage: 2 & Task: 1 & 0.003429 & 0.016008 \\
Stage: 3 & Task: 1 & 0.016438 & 0.019504 \\
\hline
Stage: 1 & Task: 2 & 0.003314 & 0.003629 \\
Stage: 2 & Task: 2 & 0.016270 & 0.026802 \\
Stage: 3 & Task: 2 & 0.027042 & 0.030579 \\
\hline
Stage: 1 & Task: 3 & 0.003776 & 0.004056 \\
Stage: 2 & Task: 3 & 0.026944 & 0.037537 \\
Stage: 3 & Task: 3 & 0.037839 & 0.040872 \\
\hline
Stage: 1 & Task: 4 & 0.004245 & 0.004612 \\
Stage: 2 & Task: 4 & 0.037680 & 0.048180 \\
Stage: 3 & Task: 4 & 0.048373 & 0.051491 \\
\hline
Stage: 1 & Task: 5 & 0.004766 & 0.005046 \\
Stage: 2 & Task: 5 & 0.048321 & 0.058671 \\
Stage: 3 & Task: 5 & 0.058857 & 0.062011 \\
\hline
Stage: 1 & Task: 6 & 0.005198 & 0.005489 \\
Stage: 2 & Task: 6 & 0.058783 & 0.069260 \\
Stage: 3 & Task: 6 & 0.069451 & 0.071884 \\
\hline
Stage: 1 & Task: 7 & 0.005622 & 0.005898 \\
Stage: 2 & Task: 7 & 0.069384 & 0.079809 \\
Stage: 3 & Task: 7 & 0.080004 & 0.083281 \\
\hline
Stage: 1 & Task: 8 & 0.006060 & 0.006336 \\
Stage: 2 & Task: 8 & 0.079923 & 0.090405 \\
Stage: 3 & Task: 8 & 0.090638 & 0.093818 \\
\hline
Stage: 1 & Task: 9 & 0.006461 & 0.006736 \\
Stage: 2 & Task: 9 & 0.090529 & 0.094146 \\
Stage: 3 & Task: 9 & 0.094260 & 0.096577 \\
\hline
```

Рис. 4.1. Пример работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- операционная система: Fedora 35 [3];
- оперативная память: 16 Гб;
- процессор: AMD Ryzen5 2500U [2]:
  - количество физических ядер: 4;
  - количество логических ядер: 8.

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов измерялось на автоматически генерируемых в необходимом количестве пользовательских данных с использованием функции time библиотеки time. Замеры производились на компьютере, подключенном к сети электропинания с отключенным режимом энергосбережения. Усредненные результаты 10 замеров реального времени работы приведены в таблице ниже.

На рисунке 4.2 представлена зависимость времени выполнения алгоритма в зависимости от «плана на день» – количества пользователей для обработки - на основе таблицы 4.1. Последовательное выполнение занимает в среднем в 1.5-1.7 раз больше времени, чем конвейерное с незначительным увеличением выигрыша обработки на конвейере при увеличении количества пользователей. Таким образом, выигрыш от использования конвейерной обработки при количестве пользователей от 15 до 85 приблизительно одинаков и составляет около 1.6 раз.

Таблица 4.1. Время выполнения алгоритма для разного количества пользователей в микросекундах

Размер	Последовательное выполнение	Конвейерная обработка
5	38291.287	36348.152
15	109756.374	73250.5798
25	181147.265	113172.388
35	252654.957	152421.474
45	322310.686	197659.373
55	399007.916	239235.663
65	461043.334	286625.170
75	541337.20	321961.14

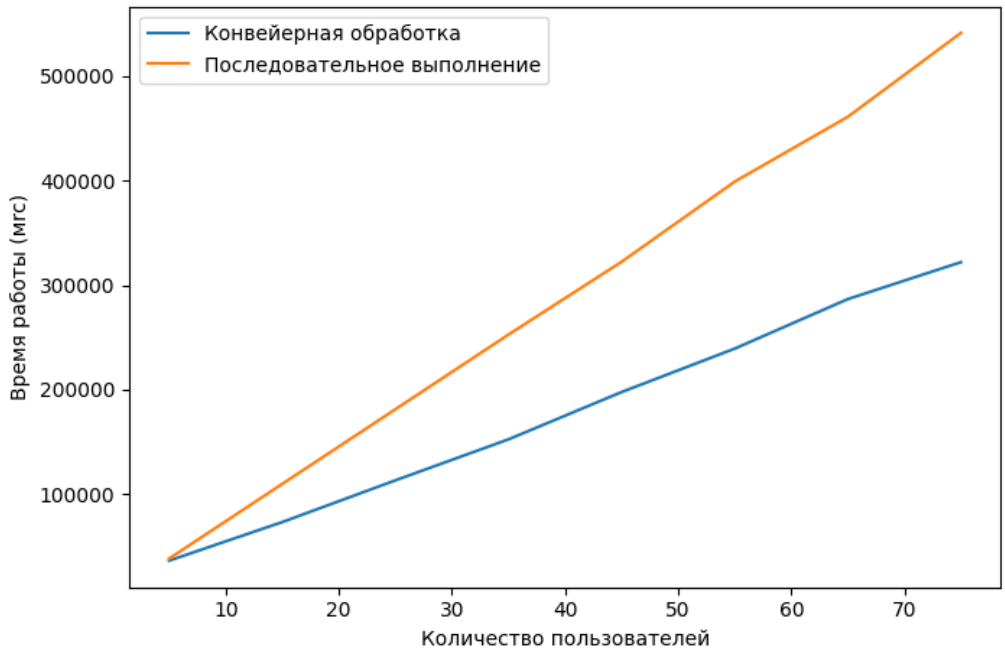

Рис. 4.2. Зависимость времени выполнения от количества пользователей

4.4 Тестирование

Для тестирования корректности работы ПО используется анализ логов. Полученная для последних 6 заявок таблица приведена ниже. На рисунке 4.3 представлены данные о максимальном, минимальном и среднем времени обработки заявок на каждой ленте конвейера.

Таблица 4.2. Полученный лог

Stage N	Task M	Start Time	End Time
Stage: 1	Task: 24	0.012759	0.012983
Stage: 2	Task: 24	0.096859	0.100533
Stage: 3	Task: 24	0.100766	0.103546
Stage: 1	Task: 25	0.013081	0.013289
Stage: 2	Task: 25	0.100624	0.104318
Stage: 3	Task: 25	0.104567	0.107468
Stage: 1	Task: 26	0.013381	0.013590
Stage: 2	Task: 26	0.104418	0.108121
Stage: 3	Task: 26	0.108363	0.111242
Stage: 1	Task: 27	0.013669	0.017236
Stage: 2	Task: 27	0.108238	0.111867
Stage: 3	Task: 27	0.112061	0.114830
Stage: 1	Task: 28	0.017447	0.017701
Stage: 2	Task: 28	0.111984	0.115609
Stage: 3	Task: 28	0.115740	0.118498
Stage: 1	Task: 29	0.017766	0.017931
Stage: 2	Task: 29	0.115671	0.119328
Stage: 3	Task: 29	0.119419	0.122284
Stage: 1	Task: 30	0.017987	0.018134
Stage: 2	Task: 30	0.119397	0.123048
Stage: 3	Task: 30	0.123159	0.126240

```
Max time on stage 1 = 3566.265106201172 mks
Min time on stage 1 = 146.38900756835938 mks
Avg time on stage 1 = 374.62711334228516 mks

Max time on stage 2 = 6560.087203979492 mks
Min time on stage 2 = 3587.484359741211 mks
Avg time on stage 2 = 3821.8100865681968 mks

Max time on stage 3 = 4501.3427734375 mks
Min time on stage 3 = 2593.517303466797 mks
Avg time on stage 3 = 2849.3324915568037 mks
```

Рис. 4.3. Время выполнения заявок на каждой из лент конвейера

Вывод

В данном разделе были проведены измерения времени, затрачиваемого на загрузку данных о пользователе из базы данных, хеширование пароля и сохранения в базу данных

полученного хэша. Для размера входной очереди конвейера, принадлежащего интервалу [5, 85], выигрыш по сравнению с последовательной обработкой составил приблизительно 1.5-1.7 раз с небольшими колебаниями при увеличении количества пользователей.

ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы было реализовано асинхронное взаимодействие потоков на примере конвейерной обработки данных. В качестве конвейера рассмотрена работа с логинами и паролями пользователей: загрузка из базы данных, медленное хеширование (с использование криптографического хеша SHA-512) и сохранение полученных данных в базу данных.

В результате исследования времени выполнения алгоритма было выявлено, что в среднем для количества пользователей от 5 до 85 выигрыш при использования конвейерной обработки составляет 1.6 раз.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] About Python [Электронный ресурс]. Режим доступа: https://www.python.org/about/. Дата обращения: 20.10.2021.

[2] AMD Ryzen™ 5 2500U. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-5-2500u. Дата обращения: 20.10.2021.

[3] Fedora 35 User Docs. Режим доступа: https://docs.fedoraproject.org/en-US/fedora/f35/. Дата обращения: 20.10.2021.

[4] Подробное объяснение хеш-шифрования и инструментов md5, sha1, sha256, Java. Режим доступа: https://russianblogs.com/article/80151483863/. Дата обращения: 21.11.2021.

[5] Риски и проблемы хеширования паролей. Режим доступа: https://habr.com/ru/company/vk/blog/271245/. Дата обращения: 21.11.2021.

[6] Параллельная обработка данных. Режим доступа: https://parallel.ru/vvv/lec1.html. Дата обращения: 21.11.2021.