

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 19

**Синхронизация потоков в языке программирования Python
по дисциплине «Технологии программирования и алгоритмизация»**

Выполнил студент группы ИВТ-б-о-20-1

Бобров Н. В. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

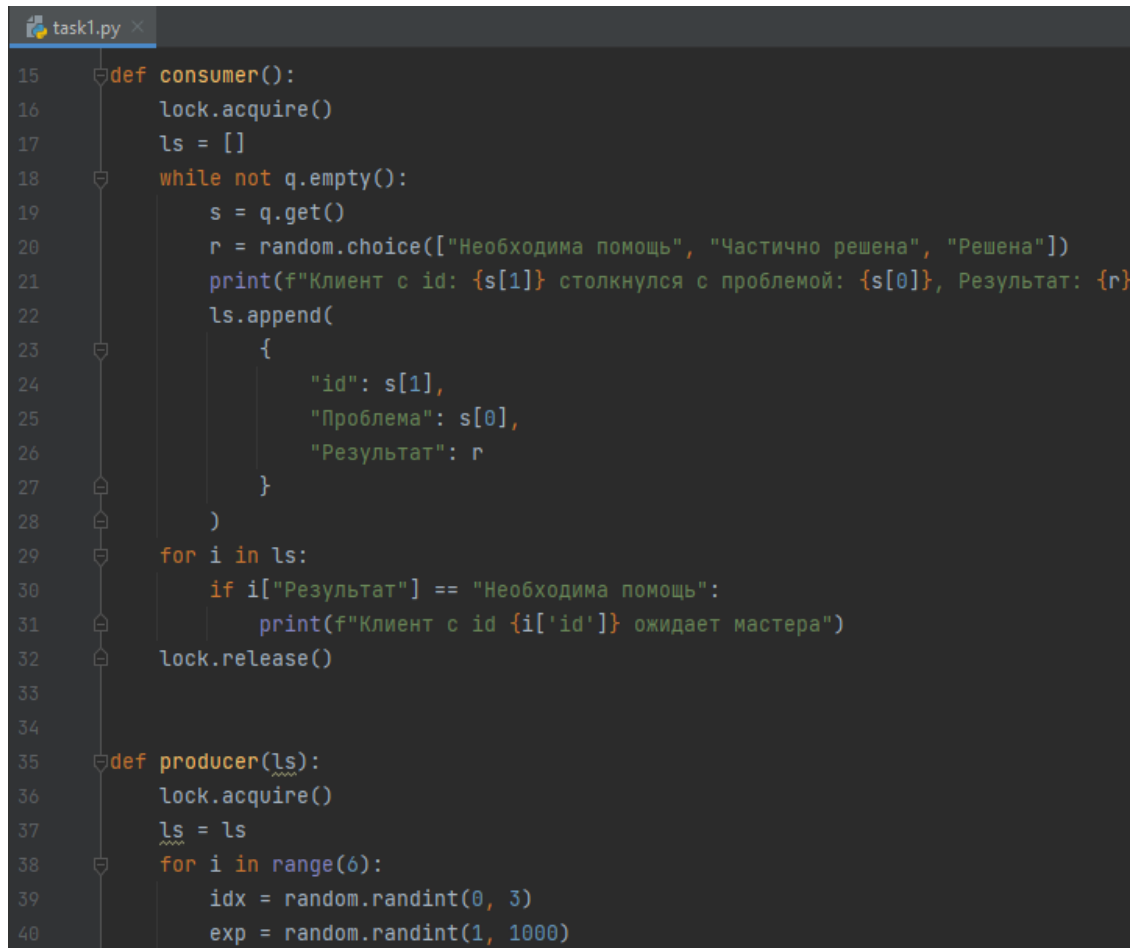
Проверил Воронкин Р. А. _____

(подпись)

Цель работы: приобретение навыков использования примитивов синхронизации в языке программирования Python версии 3.x.

Ход работы:

1. Создал общедоступный репозиторий и клонировал его на своей локальный сервер.
2. Изучив методичку к лабораторной работе, приступил непосредственно к её выполнению.
3. Выполнил первое задание, условием которого является разработка приложения, в котором выполняется решение вычислительной задачи (например, задачи из области физики, экономики, математики, статистики и т. д.) с помощью паттерна “Производитель-Потребитель”.



```
task1.py
15 def consumer():
16     lock.acquire()
17     ls = []
18     while not q.empty():
19         s = q.get()
20         r = random.choice(["Необходима помощь", "Частично решена", "Решена"])
21         print(f"Клиент с id: {s[1]} столкнулся с проблемой: {s[0]}, Результат: {r}")
22         ls.append(
23             {
24                 "id": s[1],
25                 "Проблема": s[0],
26                 "Результат": r
27             }
28         )
29     for i in ls:
30         if i["Результат"] == "Необходима помощь":
31             print(f"Клиент с id {i['id']} ожидает мастера")
32     lock.release()
33
34
35 def producer(ls):
36     lock.acquire()
37     ls = ls
38     for i in range(6):
39         idx = random.randint(0, 3)
40         exp = random.randint(1, 1000)
```

Рисунок 1 – Фрагмент кода

```
task1 x
C:\ProgramData\Anaconda3\envs\2.24\python.exe C:/TPA/2.24/task1.py
Клиент с id: 279 столкнулся с проблемой: Бесконечная загрузка, Результат: Необходима помощь
Клиент с id: 674 столкнулся с проблемой: Бесконечная загрузка, Результат: Частично решена
Клиент с id: 929 столкнулся с проблемой: Бесконечная загрузка, Результат: Частично решена
Клиент с id: 976 столкнулся с проблемой: Бесконечная загрузка, Результат: Решена
Клиент с id: 411 столкнулся с проблемой: Бесконечная загрузка, Результат: Необходима помощь
Клиент с id: 268 столкнулся с проблемой: Не проходит оплата, Результат: Необходима помощь

Клиент с id 279 ожидает мастера

Клиент с id 411 ожидает мастера

Клиент с id 268 ожидает мастера

Process finished with exit code 0
```

Рисунок 2 – Результат работы программы

4. Приступил к выполнению второго задания.

```
task2.py x
11 def func2():
12     while True:
13         s = q.get()
14         y = 1/s
15         print(f'y = {y}')
16         if q.empty():
17             break
18
19
20 def func1():
21     lock.acquire()
22     x = 0.3
23     S, n = 0, 1
24     while True:
25         S1 = math.pow(x, n)
26         n += 1
27         S2 = math.pow(x, n)
28         if abs(S2 - S1) < EPS:
29             break
30         S += S2
31         print(f"S = {S}")
32         q.put(S)
33
34     lock.release()
```

Рисунок 3 – Код программы

```

S = 0.1169999999999999y = 11.11111111111111

S = 0.1251y = 8.547008547008547

S = 0.12752999999999998
y = 7.9936051159072745S = 0.12825899999999998
S = 0.12847769999999997

y = 7.841292244961971
y = 7.796723816652244
S = 0.12854330999999997y = 7.783451914223249

S = 0.12856299299999996y = 7.779479149867856

S = 0.12856889789999995y = 7.7782881112607605

S = 0.12857066936999995y = 7.777930870791111

S = 0.12857120081099996y = 7.777823705048977

S = 0.12857136024329996y = 7.777791555902188

S = 0.12857140807298995y = 7.777781911209977

y = 7.777779017806979

```

Рисунок 4 – Результат выполнения программы

Контрольные вопросы:

1. Каково назначение и каковы приемы работы с *Lock-объектом*.

Lock-объект может находиться в двух состояниях: захваченное (заблокированное) и не захваченное (не заблокированное, свободное). После создания он находится в свободном состоянии. Для работы с Lock-объектом используются методы `acquire()` и `release()`. Если Lock свободен, то вызов метода `acquire()` переводит его в заблокированное состояние.

2. В чем отличие работы с *RLock-объектом* от работы с *Lock-объектом*.

В отличие от рассмотренного выше Lock-объекта RLock может освободить только тот поток, который его захватил. Повторный захват потоком уже захваченного RLock-объекта не блокирует его. RLock-объекты поддерживают возможность вложенного захвата, при этом освобождение

происходит только после того, как был выполнен `release()` для внешнего `acquire()`.

3. Как выглядит порядок работы с условными переменными?

На стороне `Consumer`'а: проверить доступен ли ресурс, если нет, то перейти в режим ожидания с помощью метода `wait()`, и ожидать оповещение от `Producer`'а о том, что ресурс готов и с ним можно работать. Метод `wait()` может быть вызван с таймаутом, по истечении которого поток выйдет из состояния блокировки и продолжит работу.

На стороне `Producer`'а: произвести работы по подготовке ресурса, после того, как ресурс готов оповестить об этом ожидающие потоки с помощью методов `notify()` или `notify_all()`. Разница между ними в том, что `notify()` разблокирует только один поток (если он вызван без параметров), а `notify_all()` все потоки, которые находятся в режиме ожидания.

4. Какие методы доступны у объектов условных переменных?

При создании объекта `Condition` вы можете передать в конструктор объект `Lock` или `RLock`, с которым хотите работать. Перечислим методы объекта `Condition` с кратким описанием:

- `acquire(*args)` – захват объекта-блокировки.
- `release()` – освобождение объекта-блокировки.
- `wait(timeout=None)` – блокировка выполнения потока до оповещения о снятии блокировки. Через параметр `timeout` можно задать время ожидания оповещения о снятии блокировки. Если вызвать `wait()` на Условной переменной, у которой предварительно не был вызван `acquire()`, то будет выброшено исключение `RuntimeError`.

5. Каково назначение и порядок работы с примитивом синхронизации “семафор”?

С помощью семафоров удобно управлять доступом к ресурсу, который имеет ограничение на количество одновременных обращений к нему (например, количество подключений к базе данных и т.п.).

6. Каково назначение и порядок работы с примитивом синхронизации “событие”?

События по своему назначению и алгоритму работы похожи на рассмотренные ранее условные переменные. Основная задача, которую они решают – это взаимодействие между потоками через механизм оповещения. Объект класса Event управляет внутренним флагом, который сбрасывается с помощью метода clear() и устанавливается методом set(). Потоки, которые используют объект Event для синхронизации блокируются при вызове метода wait(), если флаг сброшен.

7. Каково назначение и порядок работы с примитивом синхронизации “таймер”?

Модуль threading предоставляет удобный инструмент для запуска задач по таймеру – класс Timer. При создании таймера указывается функция, которая будет выполнена, когда он сработает. Timer реализован как поток, является наследником от Thread, поэтому для его запуска необходимо вызвать start(), если необходимо остановить работу таймера, то вызовите cancel().

8. Каково назначение и порядок работы с примитивом синхронизации “барьер”?

Он позволяет реализовать алгоритм, когда необходимо дождаться завершения работы группы потоков, прежде чем продолжить выполнение задачи.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки использования примитивов синхронизации в языке программирования Python версии 3.x.