

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Синхронизация потоков в языке программирования Python»**

**Отчет по лабораторной работе № 2.24
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Трушева В. О. _____. « » 2023г.

Подпись студента _____

Работа защищена « _____ 20__ г.

Проверила Воронкин Р.А. _____

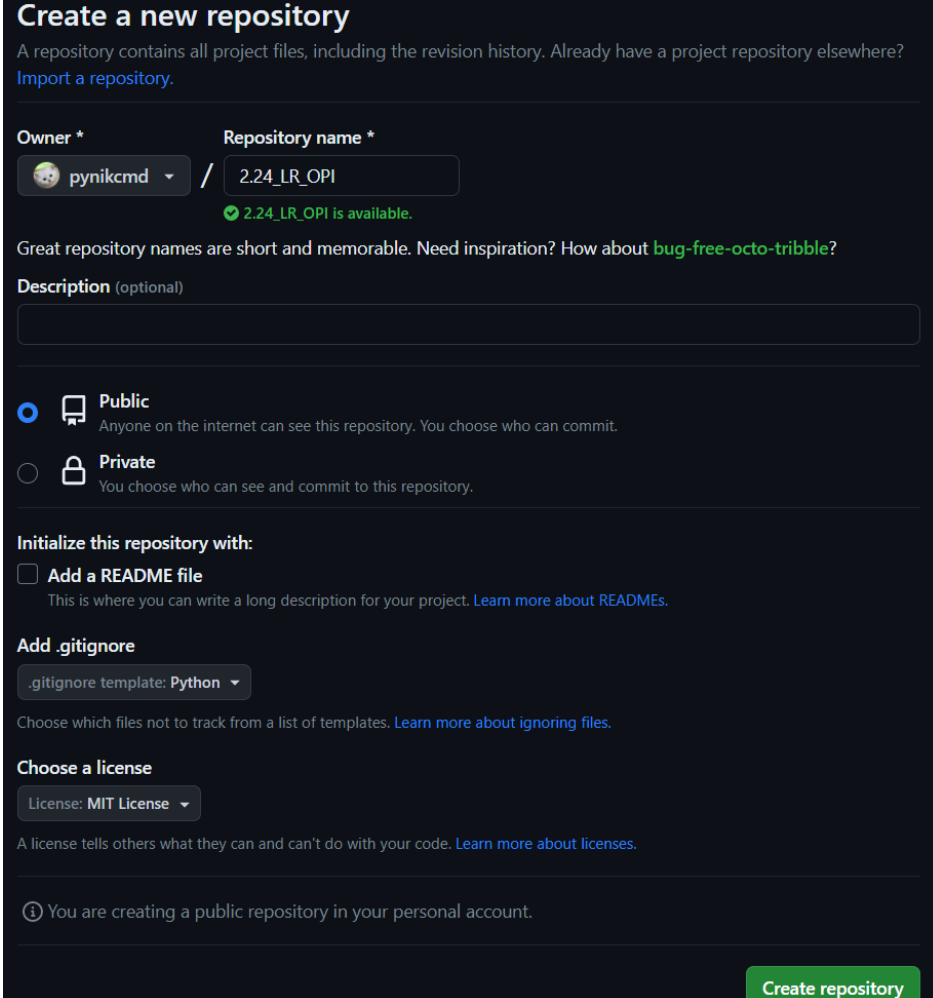
(подпись)

Ставрополь 2023

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

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.




The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two main input fields: 'Owner' and 'Repository name'. The 'Owner' field is set to 'pynikcmd' and the 'Repository name' field is set to '2.24_LR_OPI'. A green checkmark indicates that the repository name is available. Below these fields, there is a section for 'Description' with a text input area. Further down, there are two radio button options for repository visibility: 'Public' (selected) and 'Private'. Below these, there is a section for 'Initialize this repository with:' which includes a checkbox for 'Add a README file'. Underneath, there is a section for 'Add .gitignore' with a dropdown menu set to '.gitignore template: Python'. Below that, there is a section for 'Choose a license' with a dropdown menu set to 'License: MIT License'. At the bottom right, there is a green button labeled 'Create repository'.

3. Выполните клонирование созданного репозитория.

```
D:\fgit>git clone https://github.com/pynikcmd/2.24_LR_OPI.git
Cloning into '2.24_LR_OPI'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 10 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (10/10), 5.67 KiB | 725.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
```

4. Дополните файл `.gitignore` необходимыми правилами для работы с IDE PyCharm.

 `.gitignore` Update .gitignore

5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```
D:\fgit\2.24_LR_OPI>git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [D:/fgit/2.24_LR_OPI/.git/hooks]
```

6. Создайте проект PyCharm в папке репозитория.

7. Проработайте примеры лабораторной работы.

```

def order_processor(name):
    while True:
        with cv:
            # Wait while queue is empty
            while q.empty():
                cv.wait()
            try:
                # Get data (order) from queue
                order = q.get_nowait()
                print(f"{name}: {order}")
                # If get "stop" message then stop thread
                if order == "stop":
                    break
            except:
                pass
        sleep(0.1)

if __name__ == "__main__":
    # Run order processors
    Thread(target=order_processor, args=("thread 1",)).start()
    Thread(target=order_processor, args=("thread 2",)).start()

```

processor() > while True > with cv > except

```

Primer_1 x
thread 1: order 5
thread 1: order 6
thread 1: order 7
thread 1: order 8
thread 1: order 9
thread 1: stop
thread 2: stop
thread 3: stop

```

Пример 1

```

from threading import Thread, BoundedSemaphore
from time import sleep, time

ticket_office = BoundedSemaphore(value=3)

def ticket_buyer(number):
    start_service = time()
    with ticket_office:
        sleep(1)
        print(f"client {number}, service time: {time() - start_service}")

if __name__ == "__main__":
    buyer = [Thread(target=ticket_buyer, args=(i,)) for i in range(5)]
    for b in buyer:
        b.start()

```

== "__main__" > for b in buyer
 Primer_2 ×

```

D:\fgit\2.24_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\2.24_LR_OPI\Task
client 0, service time: 1.0082292556762695
client 1, service time: 1.0082292556762695
client 2, service time: 1.0004193782806396
client 3, service time: 2.0010366439819336
client 4, service time: 2.002129316329956

```

Пример 2

```
event = Event()

def worker(name: str):
    event.wait()
    print(f"Worker: {name}")

if __name__ == "__main__":
    # Clear event
    event.clear()
    # Create and start workers
    workers = [Thread(target=worker, args=(f"wrk {i}",)) for i in range(5)]
    for w in workers:
        w.start()

    print("Main thread")
    event.set()

__name__ == "__main__"
```

Primer_3 x

D:\fgit\2.24_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\2.24_LR_OPI\Tasks\Main thread
Worker: wrk 0
Worker: wrk 2
Worker: wrk 3Worker: wrk 4
Worker: wrk 1

Пример 3

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from threading import Timer

if __name__ == "__main__":
    timer = Timer(interval=3, function=lambda: print("Message from Timer!"))
    timer.start()
```

Primer_4 x

D:\fgit\2.24_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\2.24_LR_OPI\Tasks\FMessage from Timer!

Пример 4

```
br = Barrier(3)
store = []

def f1(x):
    print("Calc part1")
    store.append(x**2)
    sleep(0.5)
    br.wait()

def f2(x):
    print("Calc part2")
    store.append(x*2)
    sleep(1)
    br.wait()

if __name__ == "__main__":
    Thread(target=f1, args=(3,)).start()
    Thread(target=f2, args=(7,)).start()
    br.wait()
    print("Result: ", sum(store))
```

Primer_5 x

D:\fgit\2.24_LR_OPI\Tasks\venv\Scripts\python
Calc part1
Calc part2
Result: 23

Пример 5

8. Разработать приложение, в котором выполнить решение вычислительной задачи (например, задачи из области физики, экономики, математики, статистики и т. д.) с помощью паттерна “Производитель-Потребитель”, условие которой предварительно необходимо согласовать с преподавателем.

```

def waiter():
    lock.acquire() # Захватываем блокировку
    orders = []
    while not q.empty(): # Пока очередь не пуста
        order = q.get() # Извлекаем заказ из очереди
        status = random.choice(["Готовится", "Готов", "Доставляется"]) # Случайным образом выбираем статус заказа
        print(f"Официант принял заказ: {order}, Состояние заказа: {status}")
        orders.append({
            "Заказ": order,
            "Состояние": status
        })

    for order in orders:
        if order["Состояние"] != "Готов":
            print(f"Заказ {order['Заказ']} ожидает доставку")

    lock.release() # Освобождаем блокировку

def customer(menu):
    lock.acquire() # Захватываем блокировку
    menu = menu
    for _ in range(5):
        order = random.choice(menu) # Случайным образом выбираем заказ из меню
        q.put(order) # Помещаем заказ в очередь
    lock.release() # Освобождаем блокировку

```

```

if __name__ == "__main__":
    menu = ['Пицца', 'Салат', 'Суп', 'Стейк']
    lock = Lock() # Создаем объект блокировки
    q = Queue() # Создаем очередь

    # Запускаем поток
    Thread(target=customer, args=(menu,)).start()
    Thread(target=waiter).start()

```

```

Официант принял заказ: Суп, Состояние заказа: Готовится
Официант принял заказ: Суп, Состояние заказа: Готов
Официант принял заказ: Суп, Состояние заказа: Доставляется
Официант принял заказ: Суп, Состояние заказа: Готов
Официант принял заказ: Стейк, Состояние заказа: Готов
Заказ Суп ожидает доставку
Заказ Суп ожидает доставку

```

9. Для своего индивидуального задания лабораторной работы 2.23 необходимо организовать конвейер, в котором сначала в отдельном потоке вычисляется значение первой функции, после чего результаты вычисления

должны передаваться второй функции, вычисляемой в отдельном потоке. Потоки для вычисления значений двух функций должны запускаться одновременно.

Вариант – 2 (27). Условие

2.
$$S = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \dots; x = 0,7; y = \frac{1}{1-x}.$$

```
def sum(x):
    lock.acquire()
    n = 1
    sum = 1
    temp = x
    while abs(temp) >= EPSILON:
        sum += temp
        n += 1
        temp *= x
    q.put(sum)
    lock.release()

def check_res(res, check_res):
    print(f"Значение суммы для x = 0.7: {res}")
    print(f"Проверка: {check_res}")

def main():
    x = 0.7
    check = 1 / (1 - x)

    # Создаем потоки
    thread1 = Thread(target=sum, args=(x,)).start()
    thread2 = Thread(target=check_res, args=(q.get(), check)).start()

s0
Ind_2 (1) x
D:\fgit\2.24_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\2.24_LR_OPI\Ta
Значение суммы для x = 0.7: 3.333333083650555
Проверка: 3.333333333333333
```

10. Зафиксируйте сделанные изменения в репозитории.

11. Выполните слияние ветки для разработки с веткой main (master).

12. Отправьте сделанные изменения на сервер GitHub.

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. Повторный вызов `acquire()` приведет к блокировке инициировавшего это действие потока до тех пор, пока Lock не будет разблокирован каким-то другим потоком с помощью метода `release()`. Вывоз метода `release()` на свободном Lock-объекте приведет к выбросу исключения `RuntimeError`.

2. В чем отличие работы с RLock-объектом от работы с Lock-объектом. 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()`, у RLock нет метода `locked()`.

3. Как выглядит порядок работы с условными переменными? Порядок работы с условными переменными выглядит так:

1. 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.

2. 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? 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`.

`wait_for(predicate, timeout=None)` – метод позволяет сократить количество кода, которое нужно написать для контроля готовности ресурса и ожидания оповещения.

`notify(n=1)` – снимает блокировку с остановленного методом `wait()` потока. Если необходимо разблокировать несколько потоков, то для этого следует передать их количество через аргумент `n`.

`notify_all()` – снимает блокировку со всех остановленных методом `wait()` потоков.

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

Суть его идеи заключается в том, при каждом вызове метода `acquire()` происходит уменьшение счетчика семафора на единицу, а при вызове `release()` – увеличение. Значение счетчика не может быть меньше нуля, если на момент вызова `acquire()` его значение равно нулю, то происходит блокировка потока до тех пор, пока не будет вызван `release()`. Семафоры поддерживают протокол менеджера контекста.

Для работы с семафорами в Python есть класс `Semaphore`, при создании его объекта можно указать начальное значение счетчика через параметр `value`. `Semaphore` предоставляет два метода:

`acquire(blocking=True, timeout=None)` – если значение внутреннего счетчика больше нуля, то счетчик уменьшается на единицу и метод возвращает `True`. Если значение счетчика равно нулю, то вызвавший данный метод поток блокируется, до тех пор, пока не будет кем-то вызван метод `release()`. Дополнительно при вызове метода можно указать параметры

`blocking` и `timeout`, их назначение совпадает с `acquire()` для `Lock`. `release()` – увеличивает значение внутреннего счетчика на единицу.

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

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.

Методы класса Event:

`is_set()` – возвращает `True` если флаг находится в взведенном состоянии. `set()` – переводит флаг в взведенное состояние.

`clear()` – переводит флаг в сброшенное состояние.

`wait(timeout=None)` – блокирует вызвавший данный метод поток если флаг соответствующего Event-объекта находится в сброшенном состоянии. Время нахождения в состоянии блокировки можно задать через параметр `timeout`.

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

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()`.

Конструктор класса `Timer`:

`Timer(interval, function, args=None, kwargs=None)` Параметры:

`interval` – количество секунд, по истечении которых будет вызвана функция `function`.

`function` – функция, вызов которой нужно осуществить по таймеру.
`args, kwargs` – аргументы функции `function`.

Методы класса `Timer`:

`cancel()` – останавливает выполнение таймера

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.

Конструктор класса:

`Barrier(parties, action=None, timeout=None)` Параметры:

`parties` – количество потоков, которые будут работать в рамках барьера.

`action` – определяет функцию, которая будет вызвана, когда потоки будут освобождены (достигнут барьера).

`timeout` – таймаут, который будет использовать как значение по умолчанию для методов `wait()`.

Свойства и методы класса:

`wait(timeout=None)` – блокирует работу потока до тех пор, пока не будет получено уведомление либо не пройдет время указанное в `timeout`.

`reset()` – переводит `Barrier` в исходное (пустое) состояние. Потокам, ожидающим уведомления, будет передано исключение `BrokenBarrierError`.

`abort()` – останавливает работу барьера, переводит его в состояние “разрушен” (`broken`). Все текущие и последующие вызовы метода `wait()` будут завершены с ошибкой с выбросом исключения `BrokenBarrierError`.

`parties` – количество потоков, которое нужно для достижения барьера. `n_waiting` – количество потоков, которое ожидает срабатывания барьера.

`broken` – значение флага равно `True` указывает на то, что барьер находится в “разрушенном” состоянии.

9. Сделайте общий вывод о применении тех или иных примитивов синхронизации в зависимости от решаемой задачи.

Для решения определенного вида задач удобным будет каждый из способов, в зависимости от условий задачи, универсального способа нет.