

**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(11)

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

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

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * Yana-Kh / Repository name * Lab-2.24-OPJ

✔ Lab-2.24-OPJ is available.

Great repository names are short and memorable. Need inspiration? How about [miniature-rotary-phone?](#)

Description (optional)

☐ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Initialize this repository with:

☒ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set main as the default branch. Change the default name in your [settings](#).

📘 You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

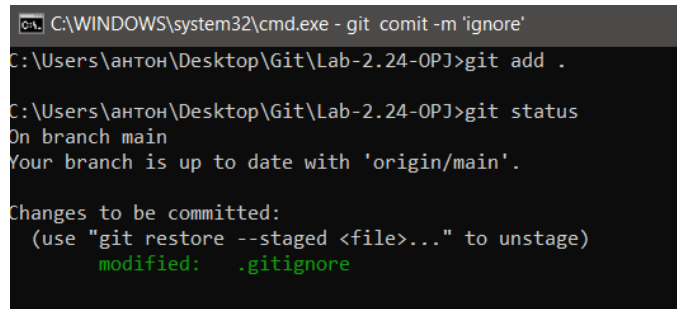
3. Выполните клонирование созданного репозитория.

```
C:\WINDOWS\system32\cmd.exe
Users\anton\Desktop\Git>git clone https://github.com/Yana-Kh/Lab-2.24-OPJ.git
Cloning into 'Lab-2.24-OPJ'...
note: Enumerating objects: 5, done.
note: Counting objects: 100% (5/5), done.
note: Compressing objects: 100% (4/4), done.
note: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

Users\anton\Desktop\Git>
```

Рисунок 2 – Клонирование репозитория

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

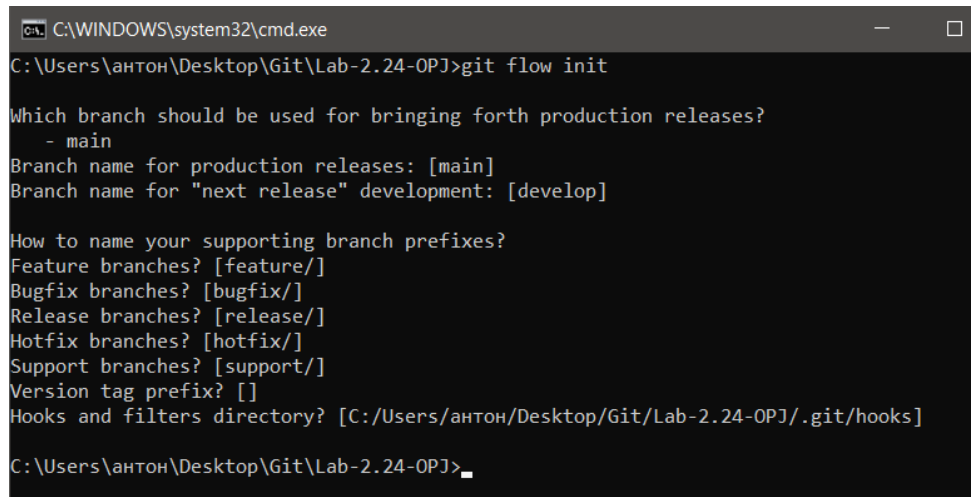


```
C:\WINDOWS\system32\cmd.exe - git commit -m 'ignore'
C:\Users\антон\Desktop\Git\Lab-2.24-0PJ>git add .
C:\Users\антон\Desktop\Git\Lab-2.24-0PJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
```

Рисунок 3 – Дополнение файла .gitignore

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



```
C:\WINDOWS\system32\cmd.exe
C:\Users\антон\Desktop\Git\Lab-2.24-0PJ>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? [C:/Users/антон/Desktop/Git/Lab-2.24-0PJ/.git/hooks]

C:\Users\антон\Desktop\Git\Lab-2.24-0PJ>
```

Рисунок 4 – Организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

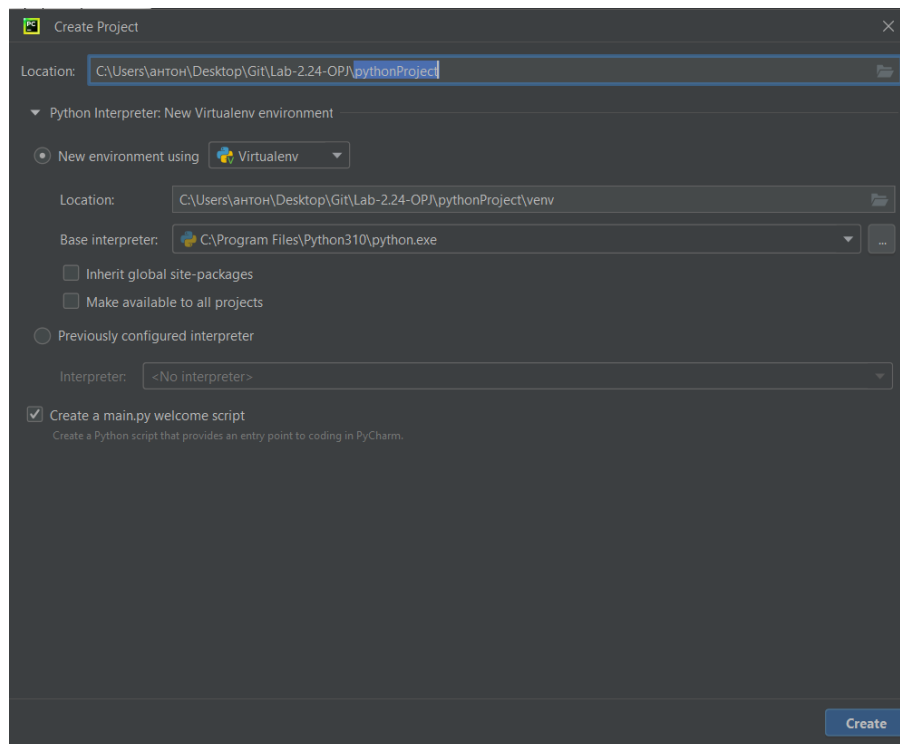


Рисунок 5 – Создание проекта PyCharm

7. Проработать примеры лабораторной работы. Создайте для них отдельные модули языка. Зафиксируйте изменения в репозитории.

Пример 1.

Код:

```
from threading import Condition, Thread
from queue import Queue
from time import sleep
cv = Condition()
q = Queue()

# Consumer function for order processing
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()
    Thread(target=order_processor, args=("thread 3",)).start()

    # Put data into queue
    for i in range(10):
        q.put(f"order {i}")

    # Put stop-commands for consumers
    for _ in range(3):
        q.put("stop")

    # Notify all consumers
    with cv:
        cv.notify_all()

```

```

thread 1: order 0
thread 3: order 1
thread 3: order 2
thread 3: order 3
thread 3: order 4
thread 3: order 5
thread 3: order 6
thread 1: order 7
thread 1: order 8
thread 1: order 9
thread 1: stop
thread 3: stop
thread 2: stop

Process finished with exit code 0

```

Рисунок 6 – Результат работы программы

Пример 2:

Код:

```

from threading import Thread, Event
from time import sleep, time

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()

```

```

Main thread
Worker: wrk 0
Worker: wrk 3
Worker: wrk 2
Worker: wrk 1
Worker: wrk 4

Process finished with exit code 0

```

Рисунок 7 – Результат работы программы

Пример 3.

Код:

```

from threading import Timer
from time import sleep, time

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

```

```

Message from Timer!

Process finished with exit code 0

```

Рисунок 7 – Результат работы программы

Пример 4.

Код:

```

from threading import Barrier, Thread
from time import sleep, time

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))

```

```

Calc part1
Calc part2
Result: 23

Process finished with exit code 0

```

Рисунок 7 – Результат работы программы

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.

Код:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from random import randint
from threading import Condition, Thread
from queue import Queue
from time import sleep
from typing import List

cv = Condition()
q = Queue()

def customer(books):
    while True:
        with cv:
            while not q.empty():
                cv.wait()
            try:
                for i in range(10):
                    n_book = randint(0, 5)
                    q.put(f"заказ №{i + 1}: {books[n_book]}")
                    if i % 4 == 0:
                        q.put("full order")

```

```

        print(f"Новый заказ!")
        break
    except:
        pass

# Consumer function for order processing
def order_processor(shop):
    while True:
        with cv:
            # Wait while queue is empty
            while q.empty():
                cv.wait()
            try:
                # Get data (order) from queue
                order = q.get_nowait()
                if order == "full order":
                    print("full order!")
                    break
                print(f"В магазин {shop} {order}")

            except:
                pass

if __name__ == "__main__":
    # Run order processors
    books: list[str] = ['Алые паруса', 'Недоросль', 'До встречи с тобой',
                        'Заводной апельсин', 'Три товарища', 'Ася']
    Thread(target=customer, args=(books,)).start()
    Thread(target=order_processor, args=("Князь Мышкин",)).start()
    Thread(target=order_processor, args=("Читай-Город",)).start()
    Thread(target=order_processor, args=("Лабиринт",)).start()
    # Notify all consumers
    with cv:
        cv.notify_all()

```

```

Новый заказ!
Новый заказ!
Новый заказ!
Новый заказ!
Новый заказ!
Новый заказ!
Новый заказ!
Новый заказ!
Новый заказ!
Новый заказ!
Новый заказ!
В магазин Князь Мышкин заказ №1: Ася
full order!
В магазин Читай-Город заказ №2: Недоросль
В магазин Читай-Город заказ №3: Заводной апельсин
В магазин Читай-Город заказ №4: Заводной апельсин
В магазин Читай-Город заказ №5: Алые паруса
full order!
В магазин Лабиринт заказ №6: Три товарища
В магазин Лабиринт заказ №7: Недоросль
В магазин Лабиринт заказ №8: Три товарища
В магазин Лабиринт заказ №9: Алые паруса
full order!

```

Рисунок 8 – Результат работы программы

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.

Код:

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

from threading import Condition, Thread
from queue import Queue

cv = Condition()
q = Queue()

def check_sum(x):
    while True:
        with cv:
            # Wait while queue is empty
            while q.empty():
                cv.wait()
            try:
                res = q.get_nowait()
                y = (5-2*x) / (6 - 5*x + x**2)
                print(f"The result check-function is: {y}")
                print(f"The sum S is: {res}")
                break
            except:
                pass

def s_sum(x):
    while True:
        with cv:
            while not q.empty():
                cv.wait()
            try:
                n = 1
                term = (1 / (2 ** n) + 1 / (3 ** n)) * (x ** (n - 1))
                epsilon = 1e-7
                ssum = term

                while abs(term) >= epsilon:
                    n += 1
                    term = (1 / (2 ** n) + 1 / (3 ** n)) * (x ** (n - 1))
                    ssum += term
                q.put(ssum)
                break
            except:
                pass

def main():
```

```

x = -0.8
thread1 = Thread(target=s_sum, args=(x,))
thread1.start()
thread2 = Thread(target=check_sum, args=(x,))
thread2.start()
with cv:
    cv.notify_all()

if __name__ == '__main__':
    main()

```

```

The result check-function is: 0.6203007518796991
The sum S is: 0.6203007273247924

Process finished with exit code 0

```

Рисунок 9 – Результат работы программы

10. Зафиксируйте сделанные изменения в репозитории.







 ex1.py	ex
 ex2.py	ex
 ex3.py	ex
 ex4.py	ex
 id1.py	id
 id2.py	id

Рисунок 10 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

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`.

У `Lock`-объекта также есть метод `locked()`, который возвращает `True` если объект захвачен, `False` в противном случае. Освободить `Lock`-объект может любой поток (на обязательно тот, который вызвал `acquire()`). Хорошей практикой при работе с `Lock`-объектами является помещение кода работы с разделяемым ресурсом в блоке `try`, а освобождать блокировку следует в `finally`.

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?

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?

`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. Он заменяет собой следующую конструкцию:

```
while not predicate():
```

```
    cv.wait()
```

`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()`.

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

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()`.

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()`.

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

Выбор примитива синхронизации зависит от требований и особенностей задачи. `Locks/Mutexes` обеспечивают взаимное исключение, `Condition`

Variables позволяют потокам синхронизироваться на условиях, Semaphores управляют доступом к ресурсу, Barriers синхронизируют группы потоков, а Atomic Operations выполняют безопасные операции над общими данными.