Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.24
дисциплины «Основы кроссплатформенного программирования»

Выполнил:
Наумов Никита Викторович
2 курс, группа ИТС-б-о-22-1,
11.03.02 «Инфокоммуникационные технологии и системы связи»,
направленность (профиль) «Инфокоммуникационные системы и сети», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой_____Дата защиты_____

Ставрополь, 2023 г.

Тема: Синхронизация потоков в языке программирования Python

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

Ход работы:

Задание 1. Создал общедоступный репозиторий на GitHub, в котором использована лицензий MIT и язык программирования Python, также добавил файл .gitignore с необходимыми правилами. Клонировал свой репозиторий на свой компьютер. Организовал свой репозиторий в соответствие с моделью ветвления git-flow, появилась новая ветка develop в которой буду выполнять дальнейшие задачи.

```
C:\Users\Gaming-PC>git clone https://github.com/EvgenyEvdakov/Laba_2.24
Cloning into 'Laba_2.24'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 1. Клонирование репозитория

Задание 2. Создал виртуальное окружение conda и активировал его, также установил необходимые пакеты isort, black, flake8.

```
(base) PS C:\Users\Gaming-PC\Laba_2.24> conda create -n 2.24 python=3.10
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 23.1.0
  latest version: 23.10.0

Please update conda by running

    $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

     conda install conda=23.10.0



## Package Plan ##

  environment location: C:\Users\Gaming-PC\.conda\envs\2.24

  added / updated specs:
    - python=3.10
```

Рисунок 2. Создание виртуального окружения

Задание 3. Создал проект PyCharm в папке репозитория. Приступил к работе с примером. Добавил новый файл primer1.py.

Условие примера: В этом примере мы создаем функцию order_processor, которая может реализовывать в себе бизнес логику, например, обработку заказа. При этом, если она получает сообщение stop, то прекращает свое выполнение. В главном потоке мы создаем и запускаем три потока для обработки заказов. Запущенные потоки видят, что очередь пуста и“встают на блокировку” при вызове wait(). В главном потоке в очередь добавляются десять заказов и сообщения для остановки обработчиков, после этого вызывается метод notify_all() для оповещения всех заблокированных потоков о том, что данные для обработки есть в очереди.

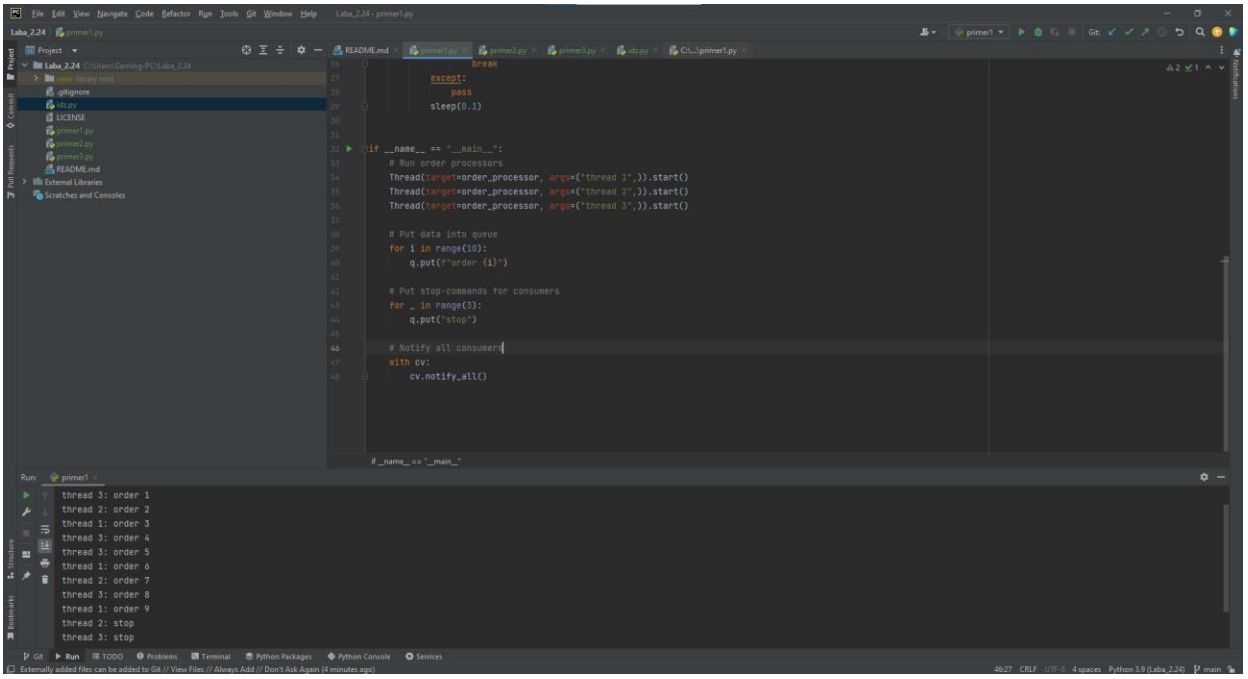

Рисунок 3. Выполнение первого примера

Добавил новый файл primer2.py.

Пример работы с Event-объектом:

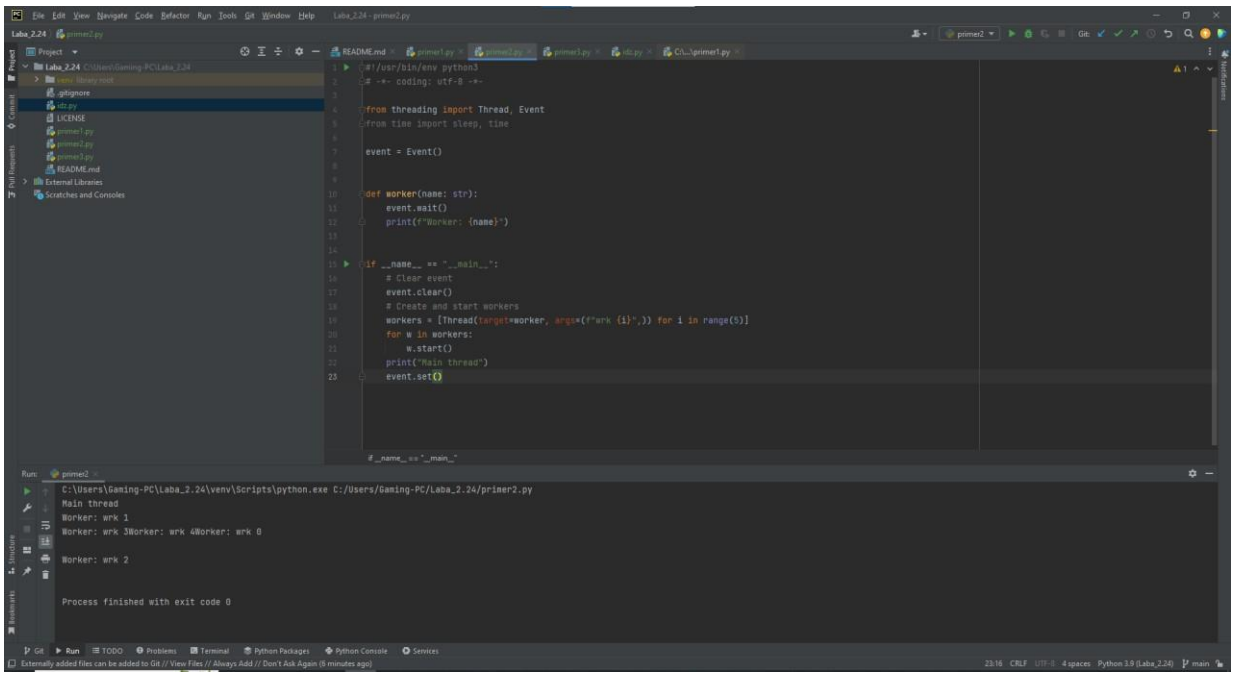

Рисунок 4. Выполнение второго примера

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.

Добавил новый файл primer3.py.

Пример работы с таймером:

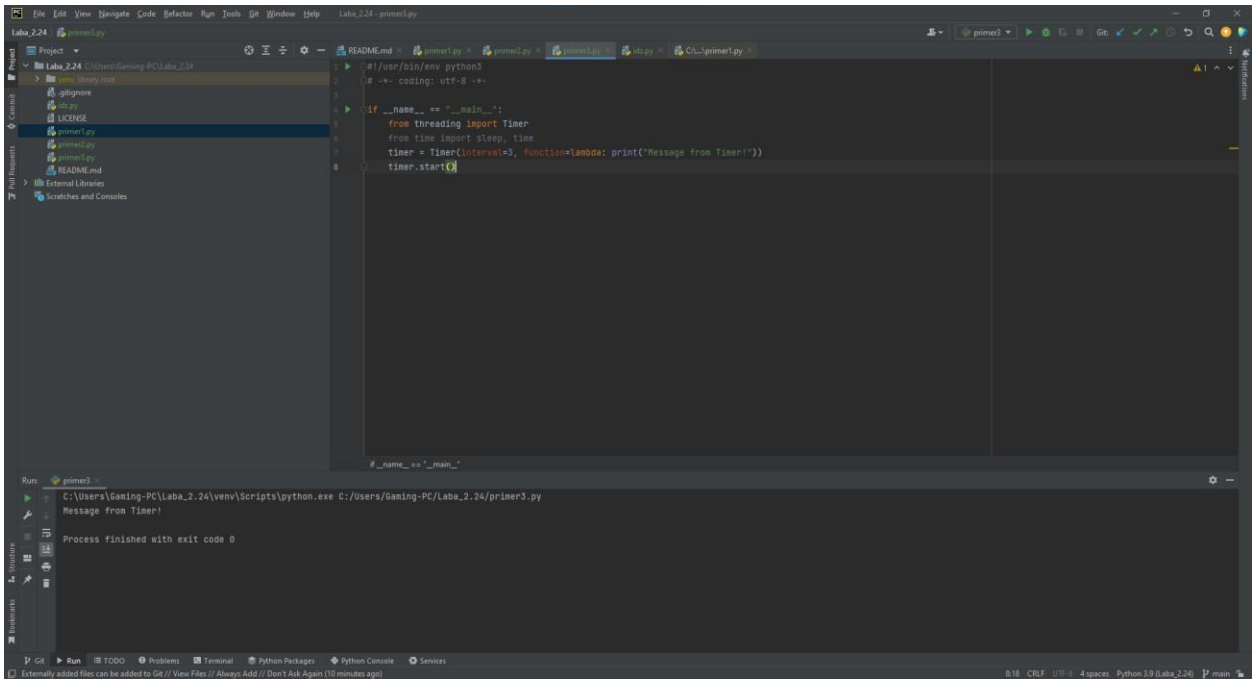

Рисунок 5. Выполнение третьего примера

Добавил новый файл primer4.py.

Пример работы с классом Barrier:

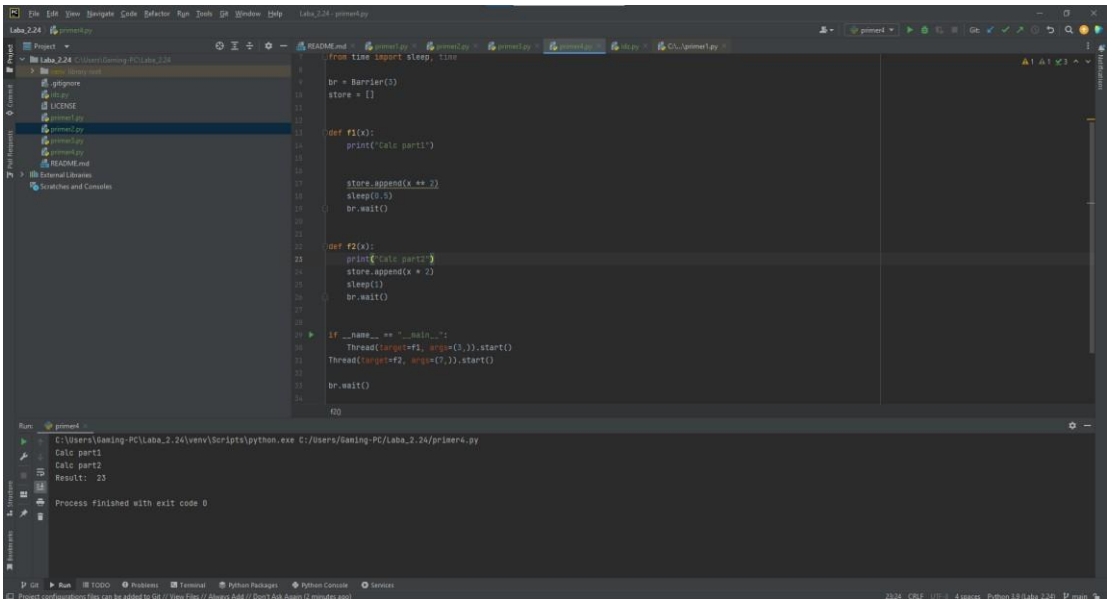

Рисунок 6. Выполнение четвертого примера

Задание 4. Необходимо 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.

Условие задания: разработать программу, в которой есть два видазадач - генерация чисел и проверка на чётность. Производитель генерирует числа, а потребитель проверяет их на чётность.

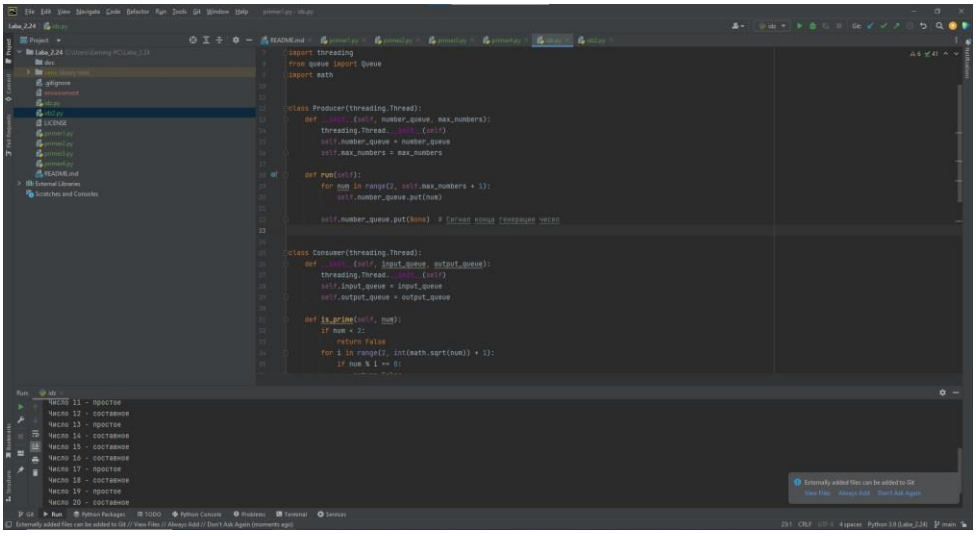

Рисунок 7. Производитель-Потребитель

Задание 5.

Индивидуальное задание

Создал новый файл под названием idz2.py.

Условие задания: 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.

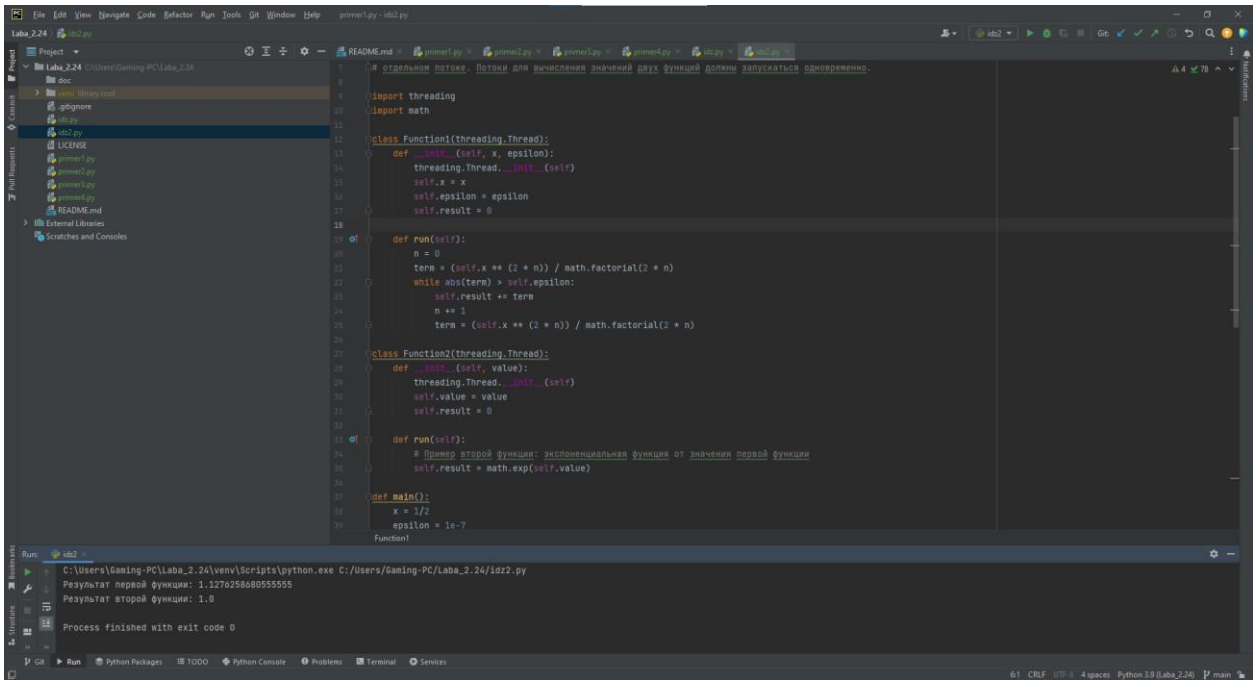

Рисунок 8. Выполнение индивидуального задания

Задание 5.

После выполнения работы на ветке develop, слил ее с веткой main и отправил изменения на удаленный сервер. Создал файл envirement.yml и деактивировал виртуальное окружение.

```
(2.24) PS C:\Users\Gaming-PC\Laba_2.24> conda env export > environment
(2.24) PS C:\Users\Gaming-PC\Laba_2.24> conda deactivate
```

Рисунок 9. Деактивация ВО

Ответы на контрольные вопросы:

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

Назначение: Lock (или блокировка) - это механизм синхронизации, предназначенный для предотвращения конфликтов доступа к общим ресурсам из нескольких потоков.

Приемы работы:

- acquire(): Захватывает блокировку. Если блокировка уже захвачена другим потоком, текущий поток блокируется до ее освобождения.
- release(): Освобождает блокировку. Если есть ожидающие потоки, один из них получит блокировку.

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

RLock (Reentrant Lock) - это вариант Lock, который может быть захвачен несколько раз одним и тем же потоком. Однако, чтобы успешно освободить RLock, его также необходимо освободить столько раз, сколько было сделано захватов.

3. **Как выглядит порядок работы с условными переменными?**

- Создать объект Condition, связанный с блокировкой.
- Использовать методы wait(), notify(), и notify_all() для организации ожидания и оповещения.

4. **Какие методы доступны у объектов условных переменных?**

- acquire(): Захватывает связанную блокировку.
- release(): Освобождает связанную блокировку.
- wait(timeout=None): Ожидает оповещения, освобождая блокировку. Может быть прерван методом notify() или по истечении времени.
- notify(n=1): Будит один поток, ожидающий условную переменную.
- notify_all(): Будит все потоки, ожидающие условную переменную.

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

Назначение: Семафор - это объект синхронизации, ограничивающий доступ к общему ресурсу. Он имеет счетчик, который уменьшается при захвате и увеличивается при освобождении.

Порядок работы:

- Создать семафор с начальным значением.
- Использовать acquire() для захвата и release() для освобождения.

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

Назначение: Событие - это сигнальный механизм, позволяющий одному потоку оповещать другие о том, что что-то произошло.

Порядок работы:

- Создать событие.
- Использовать set() для установки события и clear() для сброса.
- Другие потоки могут использовать wait() для блокировки до установки события.

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

Назначение: Таймер - это событие, которое срабатывает через определенный интервал времени.

Порядок работы:

- Создать таймер с указанием интервала и функции, которая будет выполнена по истечении времени.
- Запустить таймер.

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

Назначение: Барьер - это точка синхронизации, где несколько потоков могут встречаться и дожидаться друг друга.

Порядок работы:

- Создать барьер с указанием количества потоков, которые должны встретиться.

- Каждый поток использует wait() для ожидания других потоков.
- После того как все потоки встретились, они могут продолжить выполнение.

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

Выбор примитива синхронизации зависит от конкретной задачи. Например, Lock подходит для простых критических секций, Condition для ожидания определенного состояния, Semaphore для управления ресурсами, Event для сигнализации, Timer для отложенных действий, и Barrier для синхронизации нескольких потоков в одной точке.

Вывод: приобрел навыки использования примитивов синхронизации в языке программирования Python версии 3.x.