

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

Институт цифрового развития
Кафедра инфокоммуникаций

«Синхронизация потоков в языке программирования Python»

ОТЧЕТ
по лабораторной работе №24
дисциплины
«Основы программной инженерии»

Выполнила:

Кувшин Ирина Анатольевна
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

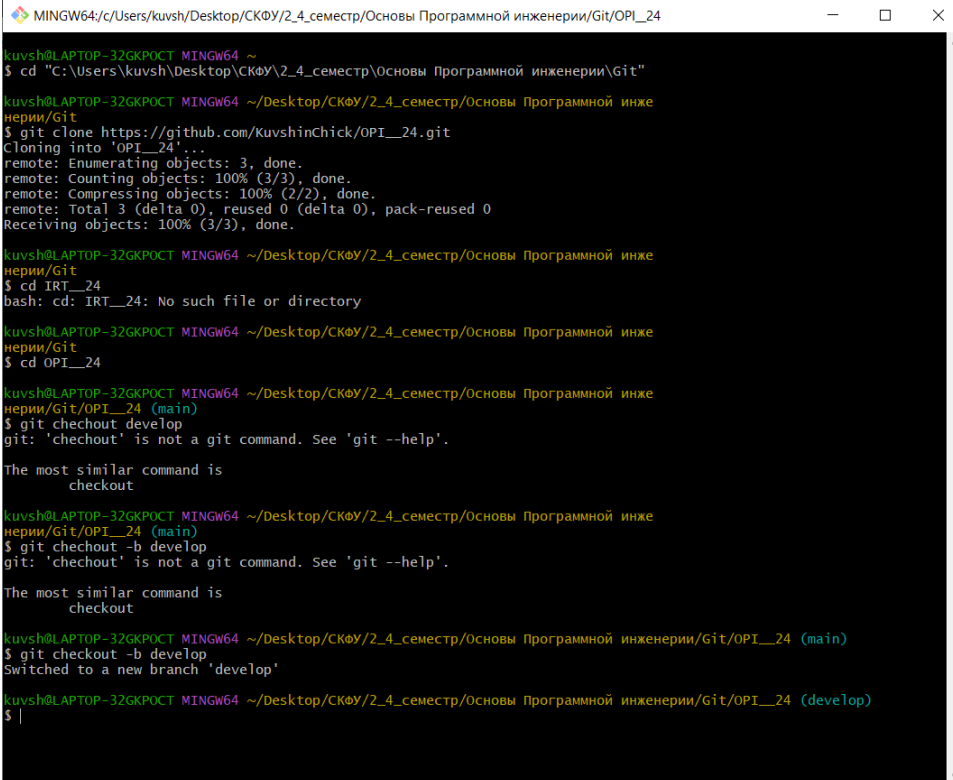
Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

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

Ход работы:

1. Изучить теоретический материал работы.
2. Проработайте примеры лабораторной работы.
3. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
4. Выполните клонирование созданного репозитория.
5. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
6. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.
7. Создайте проект PyCharm в папке репозитория.



```
MINGW64/c/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI_24
kuvsh@LAPTOP-32GKPOCT MINGW64 ~
$ cd "C:\Users\kuvsh\Desktop\СКФУ\2_4_семестр\Основы Программной инженерии\Git"
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ git clone https://github.com/KuvshinChick/OPI_24.git
Cloning into 'OPI_24'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ cd IRT_24
bash: cd: IRT_24: No such file or directory
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ cd OPI_24
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI_24 (main)
$ git checkout develop
git: 'checkout' is not a git command. See 'git --help'.

The most similar command is
  checkout
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI_24 (main)
$ git checkout -b develop
git: 'checkout' is not a git command. See 'git --help'.

The most similar command is
  checkout
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI_24 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI_24 (develop)
$ |
```

Рисунок 24.1 – Клонирование репозитория и создание ветки develop

```
MINGW64; c:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__24
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__24 (develop)
$ git status
On branch develop
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        README.md

nothing added to commit but untracked files present (use "git add" to track)
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__24 (develop)
$ git add .
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__24 (develop)
$ git commit -m "modified .gitignore & readme"
[develop 896c08a] modified .gitignore & readme
2 files changed, 133 insertions(+)
create mode 100644 .gitignore
create mode 100644 README.md
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__24 (develop)
$ git push origin develop
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.44 KiB | 1.44 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/KuvshinChick/OPI__24/pull/new/develop
remote:
To https://github.com/KuvshinChick/OPI__24.git
 * [new branch]      develop -> develop
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__24 (develop)
$
```

Рисунок 24.2 – Обновление .gitignore и readme

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.

Вычисление чисел Фибоначчи: Производитель генерирует числа Фибоначчи, а потребитель вычисляет и выводит в консоль сумму первых n чисел этой последовательности. Когда список будет заполнен, потребитель вычислит сумму первых n чисел и выведет ее в консоль.

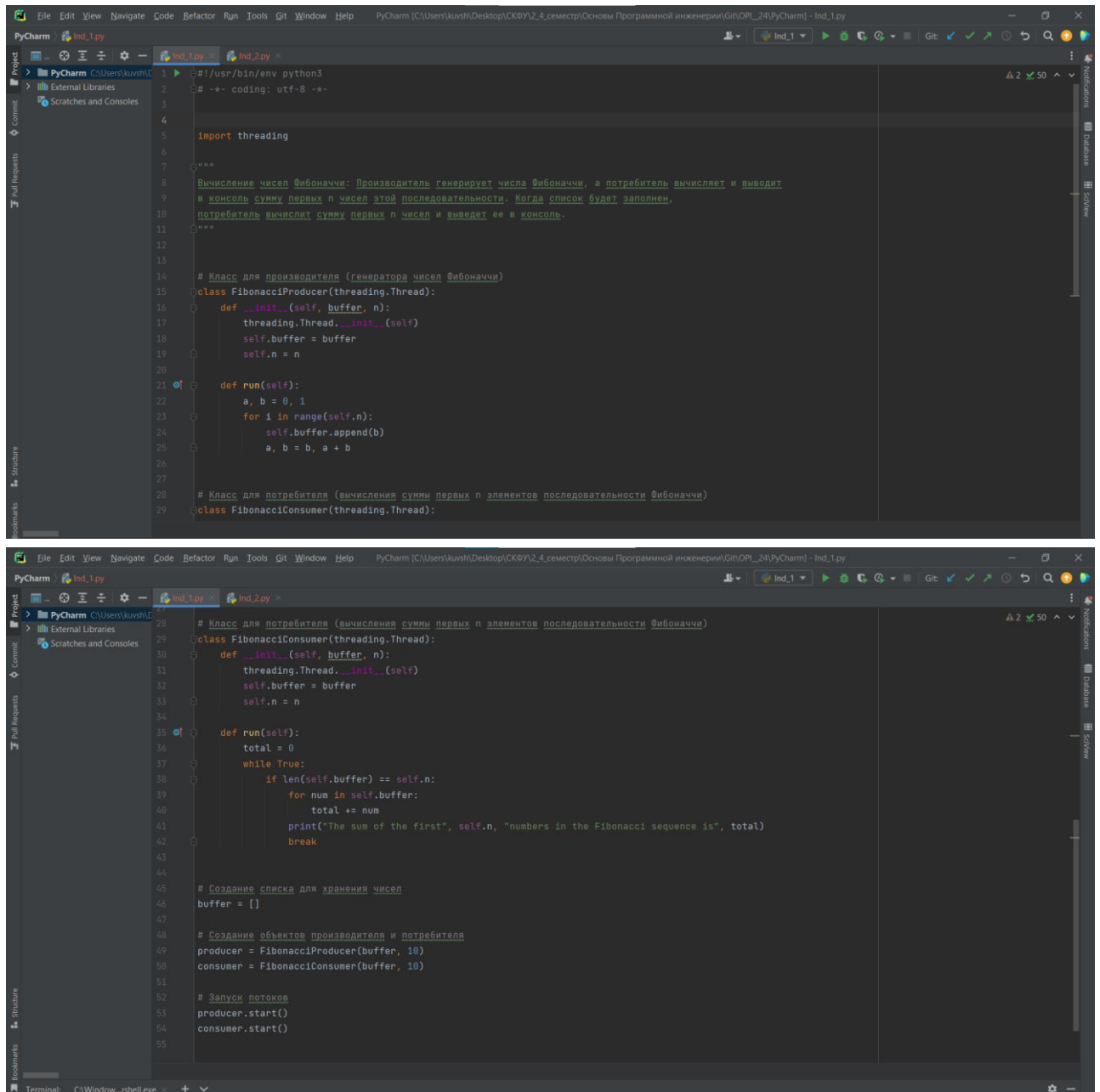


Рисунок 24.3 – Проработка программы

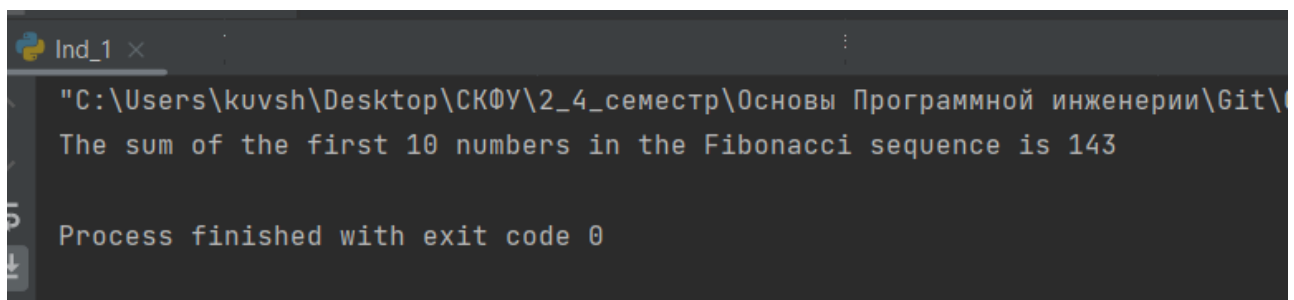


Рисунок 24.4 – Результат работы программы

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.

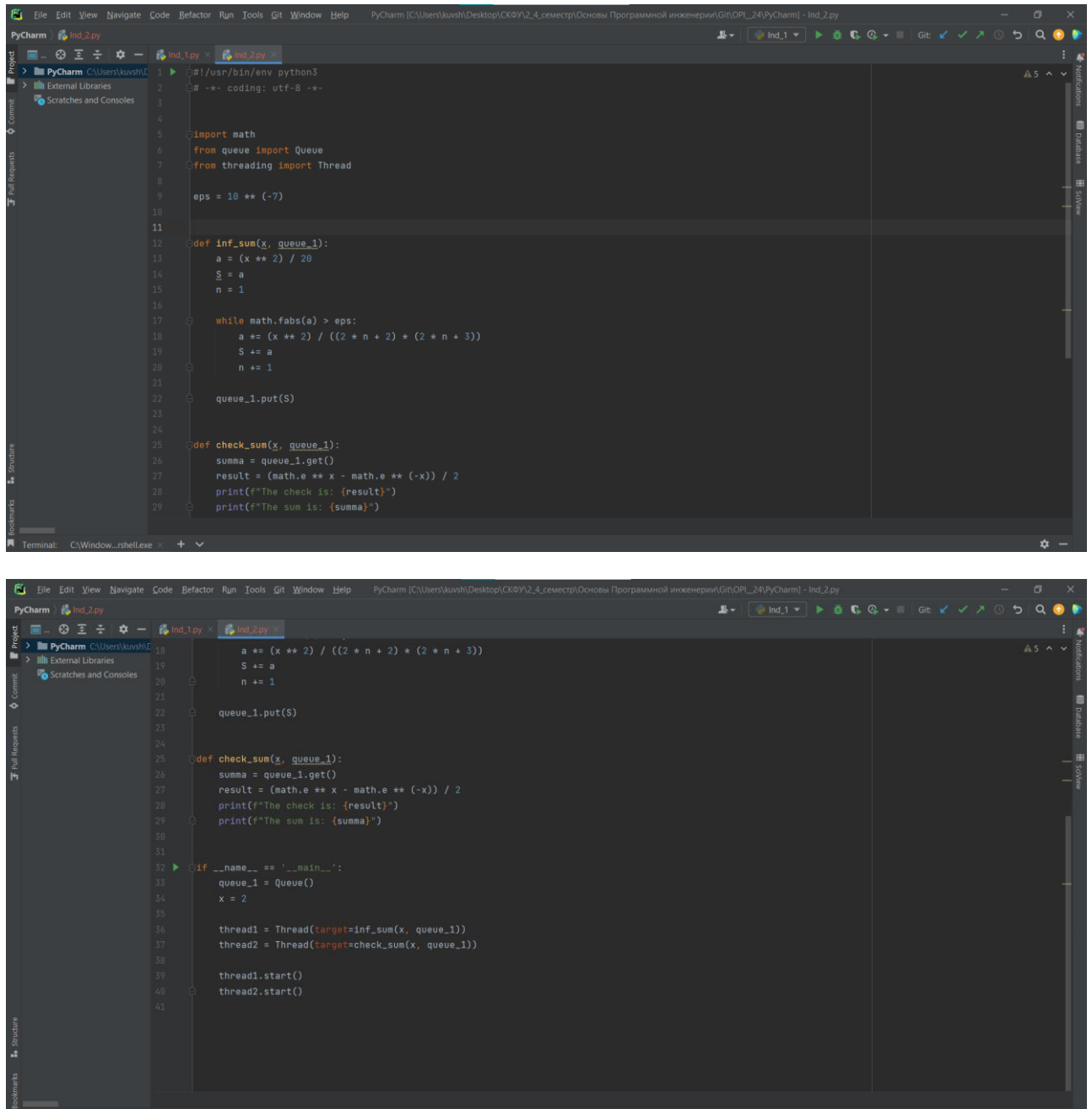
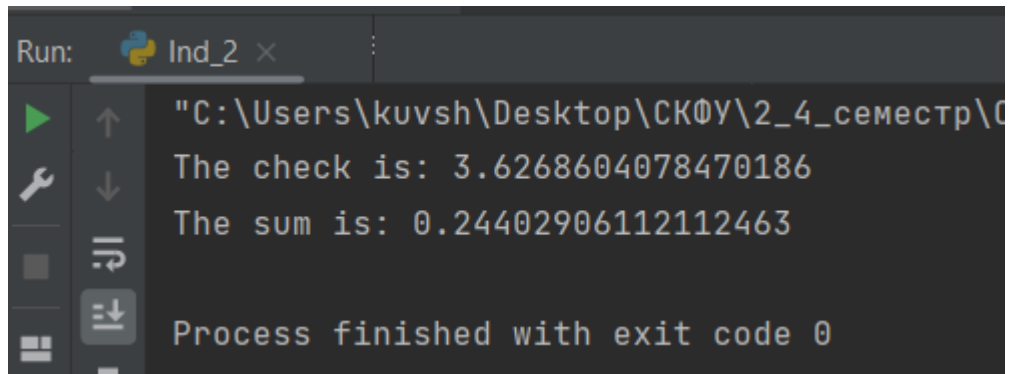


Рисунок 24.5 – Проработка программы



```
Run: Ind_2 x
"C:\Users\kuvsh\Desktop\СКФУ\2_4_семестр\0
The check is: 3.6268604078470186
The sum is: 0.24402906112112463
Process finished with exit code 0
```

Рисунок 24.6 – Результат работы программы

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 используется для синхронизации доступа к общим ресурсам из нескольких потоков. Приемы работы с Lock-объектом включают вызов метода `acquire()` для получения блокировки и `release()` для освобождения блокировки.

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

RLock-объект представляет собой рекурсивную блокировку и может быть захвачен несколько раз одним и тем же потоком. В отличие от Lock-объекта, приемы работы с RLock-объектом включают вызов метода `acquire()` и `release()` в тех же потоках в любом количестве.

3. Как выглядит порядок работы с условными переменными?

Для работы с условными переменными необходимо создать объект класса `Condition`. Затем потоки могут вызывать методы `wait()`, `notify()` и `notify_all()`, чтобы ожидать определенного условия и уведомлять другие потоки о его выполнении.

4. Какие методы доступны у объектов условных переменных?

У объектов условных переменных доступны методы `wait()`, `notify()` и `notify_all()`.

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

Семафор используется для синхронизации доступа к ограниченному количеству ресурсов. Для создания семафора в Python используется класс `Semaphore`. При получении блокировки при помощи метода `acquire()` счетчик семафора уменьшается, а при освобождении блокировки методом `release()` увеличивается.

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

Событие используется для синхронизации потоков на выполнении какого-то события, например, завершении задания. В Python для этого используется класс `Event`. Потоки могут ждать на выполнение события при помощи метода `wait()`, а событие можно установить методом `set()` и снять методом `clear()`.

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

Таймер используется для выполнения задач через определенный промежуток времени. В Python для этого можно использовать класс `Timer`. При создании объекта таймера необходимо указать интервал времени, после которого выполнится задача. Путем вызова метода `start()` таймер запускается, а метод `cancel()` останавливает выполнение задачи.

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

Барьер используется для ожидания завершения выполнения задач несколькими потоками. Для этого в Python используется класс `Barrier`. Создается

объект барьера с указанием количества потоков и методами `wait()` для ожидания завершения задач и `reset()` для возврата барьера в исходное состояние.

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

Выбор конкретного примитива синхронизации зависит от решаемой задачи. Lock-объекты и RLock-объекты используются для синхронизации доступа к общим ресурсам, семафоры - для синхронизации ограниченного количества доступа к ресурсам, события - для уведомления потоков об определенных событиях, таймеры - для отложенного выполнения задач, а барьеры - для синхронизации одновременного завершения работы нескольких потоков.