

Лабораторная работа 2.24


Тема: Синхронизация потоков в языке программирования Python

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

Ссылка на GitHub: https://github.com/RomanGorchakov/Py3_11


Порядок выполнения работы

1. Создаём новый общедоступный репозиторий, в котором будет использована лицензия MIT и язык программирования Python.

 Start writing code ...

Start a new repository for RomanGorchakov
A repository contains all of your project's files, revision history, and collaborator discussion.

Repository name *

 Py3_11 is available.

☒ **Public**
Anyone on the internet can see this repository

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

Create a new repository

Introduce yourself with a profile README
Share information about yourself by creating a profile README, which appears at the top of your profile page.

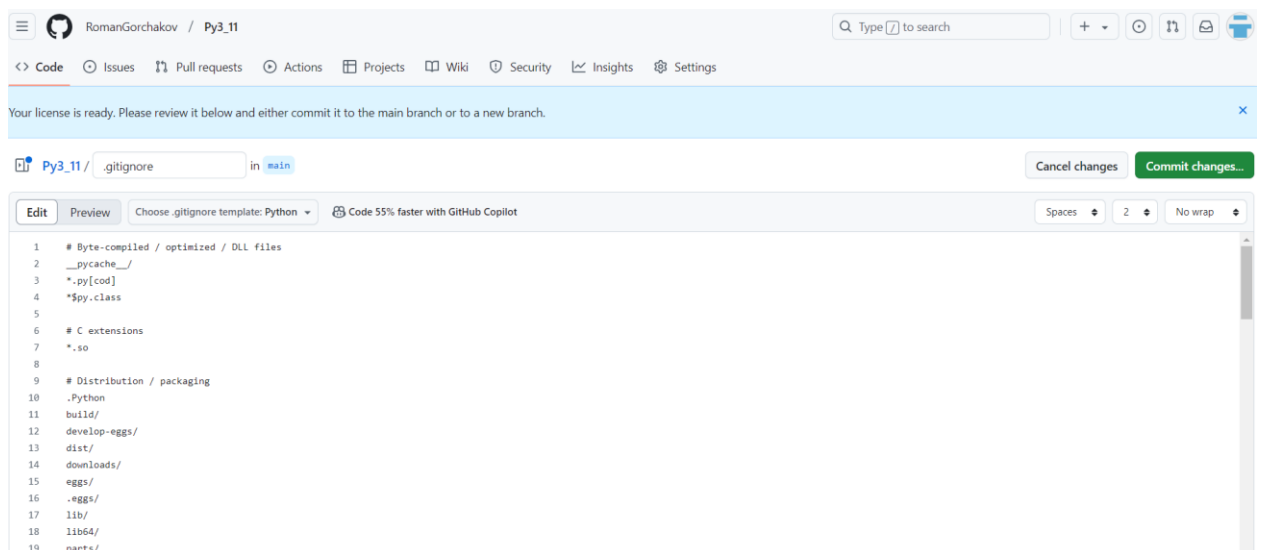
RomanGorchakov / README.md **Create**

```
1 - 🙋 Hi, I'm @RomanGorchakov
2 - 👀 I'm interested in ...
3 - 🌱 I'm currently learning ...
4 - ❤️ I'm looking to collaborate on ...
5 - 💬 How to reach me ...
6 - 🗨️ Pronouns: ...
7 - ⚡ Fun fact: ...
8
```

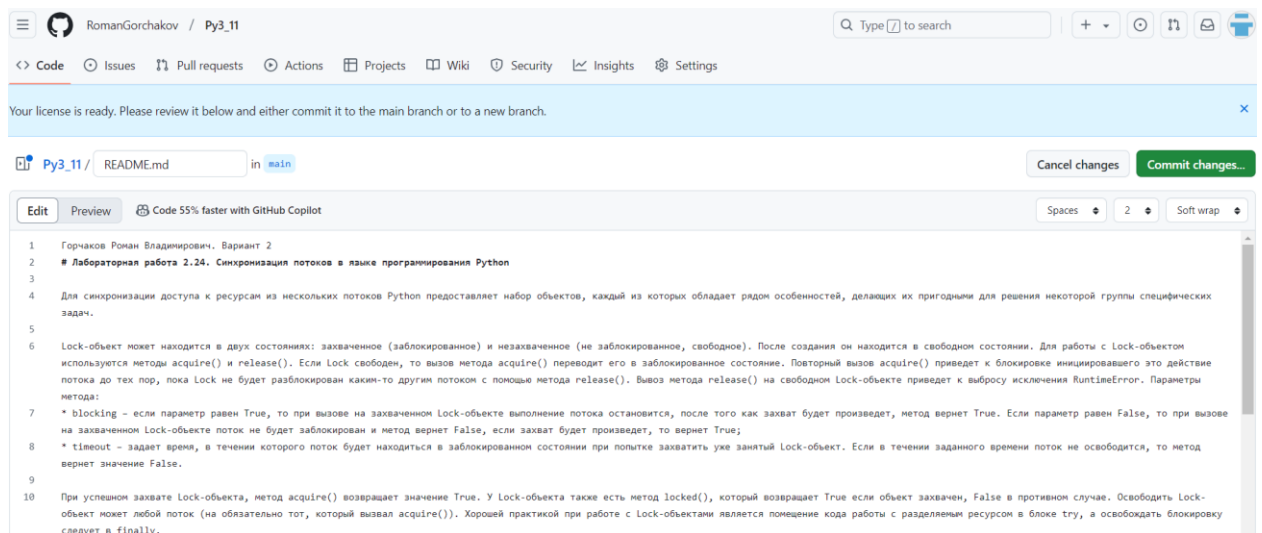
Add a license to your project

Apache License 2.0	<p>A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.</p> <table><thead><tr><th>Permissions</th><th>Limitations</th><th>Conditions</th></tr></thead><tbody><tr><td>✓ Commercial use</td><td>✗ Liability</td><td>① License and copyright notice</td></tr><tr><td>✓ Modification</td><td>✗ Warranty</td><td></td></tr><tr><td>✓ Distribution</td><td></td><td></td></tr><tr><td>✓ Private use</td><td></td><td></td></tr></tbody></table> <p>This is not legal advice. Learn more about repository licenses.</p> <p>MIT License</p> <p>Copyright (c) <input type="text" value="Year"/> <input type="text" value="Full name"/></p> <p>Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:</p>	Permissions	Limitations	Conditions	✓ Commercial use	✗ Liability	① License and copyright notice	✓ Modification	✗ Warranty		✓ Distribution			✓ Private use			<p>To adopt MIT License, enter your details. You'll have a chance to review before committing a <i>LICENSE</i> file to a new branch or the root of your project.</p> <p>Year ① <input type="text" value="2024"/></p> <p>Full name ① <input type="text" value="RomanGorchakov"/></p> <p><input type="button" value="Review and submit"/></p>
Permissions		Limitations	Conditions														
✓ Commercial use		✗ Liability	① License and copyright notice														
✓ Modification		✗ Warranty															
✓ Distribution																	
✓ Private use																	
GNU General Public License v3.0																	
MIT License																	
BSD 2-Clause "Simplified" License																	
BSD 3-Clause "New" or "Revised" License																	
Boost Software License 1.0																	
Creative Commons Zero v1.0 Universal																	
Eclipse Public License 2.0																	
GNU Affero General Public License v3.0																	
GNU General Public License v2.0																	

2. Теперь необходимо дополнить файл `.gitignore` с необходимыми правилами для языка программирования Python. Для этого переходим по ссылке «<https://github.com/github/gitignore>» и скачиваем оттуда файл «Python.gitignore».



3. Теперь создаём файл «README.md», где вносим ФИО и теоретический конспект лекции. Сохраняем набранный текст через кнопку «Commit changes».



4. В окне «Codespace» выбираем опцию «Create codespace on main». Откроется терминал, куда мы введём команду «git clone», чтобы клонировать свой репозиторий. После этого организуем репозиторий в соответствии с моделью ветвления Git-flow. Для этого введём в терминал команды: «git checkout –b develop» для создания ветки разработки; «git branch feature_branch» для создания ветки функций; «git branch release/1.0.0» для создания ветки релиза; «git checkout main» и «git branch hotfix» для создания веток hotfix. Устанавливаем библиотеки isort, black и flake8 и создаём файлы .pre-commit-config.yaml и environment.yml.

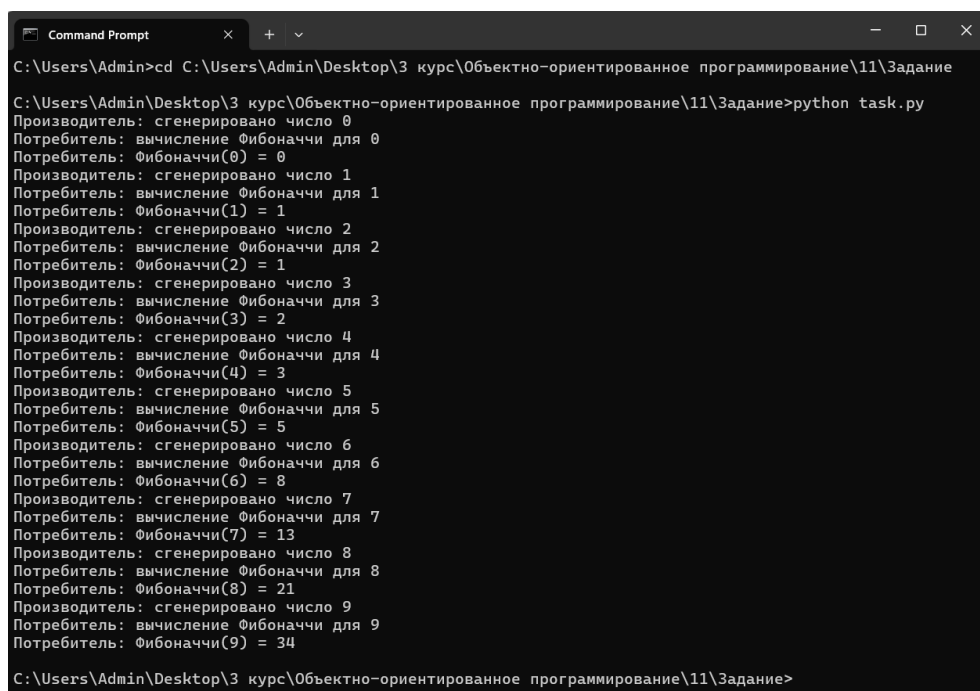
```
@RomanGorchakov →/workspaces/Py3_11 (main) $ git checkout -b develop
Switched to a new branch 'develop'
● @RomanGorchakov →/workspaces/Py3_11 (develop) $ git branch feature_branch
● @RomanGorchakov →/workspaces/Py3_11 (develop) $ git branch release/1.0.0
● @RomanGorchakov →/workspaces/Py3_11 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov →/workspaces/Py3_11 (main) $ git branch hotfix
● @RomanGorchakov →/workspaces/Py3_11 (main) $ git checkout develop
Switched to branch 'develop'
○ @RomanGorchakov →/workspaces/Py3_11 (develop) $
```

```

@RomanGorchakov → /workspaces/Py3_11 (develop) $ pip install black
Collecting black
  Downloading black-24.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl.metadata (79 kB)
Collecting click>=8.0.0 (from black)
  Downloading click-8.1.8-py3-none-any.whl.metadata (2.3 kB)
Collecting mypy_extensions>=0.4.3 (from black)
  Downloading mypy_extensions-1.0.0-py3-none-any.whl.metadata (1.1 kB)
Requirement already satisfied: packaging>=22.0 in /home/codespace/.local/lib/python3.12/site-packages (from black) (24.2)
Collecting pathspec>=0.9.0 (from black)
  Downloading pathspec-0.12.1-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: platformdirs>=2 in /home/codespace/.local/lib/python3.12/site-packages (from black) (4.3.6)
Downloading black-24.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (1.8 MB)
1.8/1.8 MB 49.0 MB/s eta 0:00:00
Downloading click-8.1.8-py3-none-any.whl (98 kB)
Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Downloading pathspec-0.12.1-py3-none-any.whl (31 kB)
Installing collected packages: pathspec, mypy_extensions, click, black
Successfully installed black-24.10.0 click-8.1.8 mypy_extensions-1.0.0 pathspec-0.12.1
@RomanGorchakov → /workspaces/Py3_11 (develop) $ pip install flake8
Collecting flake8
  Downloading flake8-7.1.1-py2.py3-none-any.whl.metadata (3.8 kB)
Collecting mccabe<0.8.0,>=0.7.0 (from flake8)
  Downloading mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting pycodestyle<2.13.0,>=2.12.0 (from flake8)
  Downloading pycodestyle-2.12.1-py2.py3-none-any.whl.metadata (4.5 kB)
Collecting pyflakes<3.3.0,>=3.2.0 (from flake8)
  Downloading pyflakes-3.2.0-py2.py3-none-any.whl.metadata (3.5 kB)
Downloading flake8-7.1.1-py2.py3-none-any.whl (57 kB)
Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Downloading pycodestyle-2.12.1-py2.py3-none-any.whl (31 kB)
Downloading pyflakes-3.2.0-py2.py3-none-any.whl (62 kB)
Installing collected packages: pyflakes, pycodestyle, mccabe, flake8
Successfully installed flake8-7.1.1 mccabe-0.7.0 pycodestyle-2.12.1 pyflakes-3.2.0
@RomanGorchakov → /workspaces/Py3_11 (develop) $ pre-commit sample-config > .pre-commit-config.yaml
@RomanGorchakov → /workspaces/Py3_11 (develop) $ conda env export > environment.yml

```

5. Создаём файл «task.py», в котором нужно разработать приложение, в котором выполнить решение вычислительной задачи с помощью паттерна «Производитель-Потребитель», условие которой предварительно необходимо согласовать с преподавателем.



```

C:\Users\Admin>cd C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\11\Задание
C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\11\Задание>python task.py
Производитель: сгенерировано число 0
Потребитель: вычисление Фибоначчи для 0
Потребитель: Фибоначчи(0) = 0
Производитель: сгенерировано число 1
Потребитель: вычисление Фибоначчи для 1
Потребитель: Фибоначчи(1) = 1
Производитель: сгенерировано число 2
Потребитель: вычисление Фибоначчи для 2
Потребитель: Фибоначчи(2) = 1
Производитель: сгенерировано число 3
Потребитель: вычисление Фибоначчи для 3
Потребитель: Фибоначчи(3) = 2
Производитель: сгенерировано число 4
Потребитель: вычисление Фибоначчи для 4
Потребитель: Фибоначчи(4) = 3
Производитель: сгенерировано число 5
Потребитель: вычисление Фибоначчи для 5
Потребитель: Фибоначчи(5) = 5
Производитель: сгенерировано число 6
Потребитель: вычисление Фибоначчи для 6
Потребитель: Фибоначчи(6) = 8
Производитель: сгенерировано число 7
Потребитель: вычисление Фибоначчи для 7
Потребитель: Фибоначчи(7) = 13
Производитель: сгенерировано число 8
Потребитель: вычисление Фибоначчи для 8
Потребитель: Фибоначчи(8) = 21
Производитель: сгенерировано число 9
Потребитель: вычисление Фибоначчи для 9
Потребитель: Фибоначчи(9) = 34
C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\11\Задание>

```

6. Создаём файл «individual.py», в котором нужно организовать конвейер, в котором сначала в отдельном потоке вычисляется значение первой функции, после чего результаты вычисления должны передаваться второй функции, вычисляемой в отдельном потоке, для индивидуального задания лабораторной работы 2.23.

```
Command Prompt
C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\11\Задание>cd C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\11\Индивидуальное задание
C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\11\Индивидуальное задание>python individual.py
Сумма ряда с точностью 1e-06: 0
Контрольное значение функции: 3.333333333333333
Разница: 3.333333333333333
C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\11\Индивидуальное задание>
```

7. Выполняем коммит файлов в репозиторий Git в ветку разработки, сливаем её с веткой main и отправляем изменения на сервер GitHub.

```
Fast-forward
 .pre-commit-config.yaml      | 38 ++++++
 .python-version              | 1 +
 edu.pyoop.code-workspace     | 47 +++++++
 environment.yml              | 80 ++++++++
 pyproject.toml               | 88 ++++++++
 setup.cfg                   | 43 ++++++
 uv.lock                     | 303 ++++++++
 "\320\227\320\260\320\264\320\260\320\275\320\270\320\265\task.py" | 55 +++++++
 ...individual.py"           | 48 ++++++
 ...320\2402.23_320\223\320\276\321\200\321\207\320\260\320\272\320\276\320\262\320\240\320\222.pdf" | Bin 0 -> 896620 bytes
10 files changed, 703 insertions(+)
create mode 100644 .pre-commit-config.yaml
create mode 100644 .python-version
create mode 100644 edu.pyoop.code-workspace
create mode 100644 environment.yml
create mode 100644 pyproject.toml
create mode 100644 setup.cfg
create mode 100644 uv.lock
create mode 100644 "\320\227\320\260\320\264\320\260\320\275\320\270\320\265\task.py"
create mode 100644 "\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\320\276\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\320\265\individual.py"
create mode 100644 "\320\236\321\202\321\207\321\221\321\202\320\233\320\2402.23_320\223\320\276\321\200\321\207\320\260\320\272\320\276\320\262\320\240\320\222.pdf"
@RomanGorchakov ->/workspaces/Py3_11 (main) $ git push -u
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 2 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (15/15), 842.16 KiB | 21.59 MiB/s, done.
Total 15 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/RomanGorchakov/Py3_11
 9299fe6..970e592 main -> main
branch 'main' set up to track 'origin/main'.
@RomanGorchakov ->/workspaces/Py3_11 (main) $
```

Py3_11Public

Pin
Unwatch1
Fork0
Star0

main1 Branch0 Tags
Go to file
Add file
Code

RomanGorchakovThread Sync970e592 · now4 Commits

Задание	Thread Sync	now
Индивидуальное задание	Thread Sync	now
Отчёт	Thread Sync	now
.gitignore	Create .gitignore	3 hours ago
.pre-commit-config.yaml	Thread Sync	now
.python-version	Thread Sync	now
LICENSE	Create LICENSE	3 hours ago
README.md	Create README.md	2 hours ago
edu.pyoop.code-workspace	Thread Sync	now
environment.yml	Thread Sync	now
pyproject.toml	Thread Sync	now
setup.cfg	Thread Sync	now
uv.lock	Thread Sync	now

README
MIT license

Горчаков Роман Владимирович. Вариант 2

Лабораторная работа 2.24. Синхронизация потоков в языке программирования Python

Для синхронизации доступа к ресурсам из нескольких потоков Python предоставляет набор объектов, каждый из которых обладает рядом особенностей, делающих их пригодными для решения некоторой группы специфических задач.

Python packageConfigure
Create and test a Python package on multiple Python versions.

Python Package using AnacondaConfigure
Create and test a Python package on multiple Python versions using Anaconda for package management.

PylintConfigure

Контрольные вопросы

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

Lock-объект может находится 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.

В отличие от 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()`. Сигнатуры и назначение методов `release()` и `acquire()` RLock-объектов совпадают с приведенными для Lock, но в отличие от него у RLock нет метода `locked()`. RLock-объекты поддерживают протокол менеджера контекста.

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”?

Реализация классического семафора, предложенного Дейкстрой. 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”?

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”?

Последний инструмент для синхронизации работы потоков, который мы рассмотрим является Barrier. Он позволяет реализовать алгоритм, когда необходимо дождаться завершения работы группы потоков, прежде чем продолжить выполнение задачи.

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

Блокировки (Lock) используются для синхронизации доступа к общим ресурсам. Они предотвращают противоречивость в выходных данных, позволяя каждый раз только одному потоку изменять данные.

Семафоры применяются для ограничения ресурсов, например, доступа к серверу, допуская обрабатывать только определённое количество клиентов за раз.

События (Event) используются для простой связи между несколькими потоками, работающими одновременно. Один поток сигнализирует о том, что произошло событие, а другие потоки активно прослушивают этот сигнал.

Объект Condition работает как коммуникатор между потоками и применяется для уведомления других потоков об изменении состояния программы. Например, его можно использовать для сигнализации доступности ресурса.

Барьеры используются для синхронизации работы сервера и клиента, поскольку серверу часто приходится ожидать клиента после инициализации.