Способы синхронизации потоков в Python.

**Semaphore/BoundedSemaphore**

Семафоры часто используются для защиты ресурсов с ограниченной пропускной способностью, например, сервера баз данных. В любой ситуации, когда размер ресурса фиксирован, вам следует использовать ограниченный семафор. Семафор управляет атомарным счетчиком, представляющим количество вызовов release() минус количество вызовов acquire() плюс начальное значение. Ограниченный семафор проверяет, не превышает ли его текущее значение его начального значения. Если это происходит, то возникает ValueError.

**Условием (Condition)**

Этот класс реализует объекты переменной состояния. Переменная условия позволяет одному или нескольким потокам ожидать, пока они не получат уведомление от другого потока.

Основное назначение условных переменных – это синхронизация работы потоков, которая предполагает ожидание готовности некоторого ресурса и оповещение об этом событии. Наиболее явно такой тип работы выражен в паттерне Producer-Consumer (Производитель – Потребитель).

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.

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.

**Событием (Event)**

События по своему назначению и алгоритму работы похожи на 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.

**Барьером (Barrier)**

Он позволяет реализовать алгоритм, когда необходимо дождаться завершения работы группы потоков, прежде чем продолжить выполнение задачи.

Параметры:

Количество потоков, которые будут работать в рамках барьера.

Определяет функцию, которая будет вызвана, когда потоки будут освобождены (достигнут барьера).

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 равное *True*указывает на то, что барьер находится в “разрушенном” состоянии.

**Очередью (Queue)**

Queue предоставляет очередь FIFO "первый вход" и "первый выход", что означает, что элементы извлекаются из очереди в том порядке, в котором они были добавлены. Первые элементы, добавленные в очередь, будут первыми извлеченными элементами.

Put - Поместите элемент в очередь. Если необязательный блок аргументов имеет значение true, а тайм-аут равен None (по умолчанию), при необходимости заблокируйте до тех пор, пока не освободится свободное место.

Get - Удалите и верните элемент из очереди. Если необязательный блок аргументов имеет значение true, а тайм-аут равен None (по умолчанию), при необходимости заблокируйте до тех пор, пока элемент не станет доступен.