Нарисовать на доске схему:

---------------------------🡪Time

Pr. 1 === == =

Pr. 2 ======= ==

Pr. 3 = = = == == = - потоки

Внутри потока ====: ---====----==-----= - асинхр.ф-ии

Значительная часть теоретических основ параллельного программирования фактически была заложена в 1960е. Самый ранний алгоритмический язык ALGOL 68, который впервые был разработан в 1959, содержал средства, которые выполняли поддержку совместного программирования. Академическое изучение параллелизма официально началось с основополагающей работы Эдсгера Дейкстры, который был пионером в области науки о вычислениях, наиболее известным благодаря алгоритму поиска пути, названному в его честь.

Эта плодотворная статья, рассматривается как самый первый документ в области параллельного программирования, в котором Дейкстра определил и разрешил проблему взаимного исключения. Взаимное исключение, которое является неким свойством управления одновременностью, предотвращающим состояние конкурентности, стало одной из самых обсуждаемых тем в параллельной обработке.

Тем не менее, после этого не было значительного интереса. Начиная примерно с 1970 до ранних 2000, о процессорах говорилось, что они удваивают скорость исполнения каждые 18 месяцев. На протяжении этого периода программистам не было нужды заниматься параллельным программированием, поскольку всё, что им нужно было сделать чтобы их программы работали быстрее, это подождать. Однако в начале 2000-х произошла смена парадигмы в процессорном бизнесе; вместо того чтобы выпускать всё более большие и быстрые процессоры для компьютеров, производители стали сосредотачиваться на более медленных процессорах меньшего размера, которые помещались вместе в группы. Именно в это время компьютеры начали иметь процессоры со множеством ядер.

В наши дни среднестатистический компьютер имеет более одного ядра. Поэтому, если программист создаёт все свои программы как не применяющие параллелизм в каком- то виде, он обнаружит что его программы используют только одно ядро или один поток для обработки данных, в то время как остальной ЦПУ простаивает, не делая. Именно это является одной из причин недавнего рывка в развитии параллельного программирования (язык Go).

Другой причиной для роста популярности параллельной обработки является увеличение области разработки графических, мультимедийных и основанных на веб приложений, в которых широко применяются приложения одновременности для решения сложных и многогранных задач. Например, параллельность является основным игроком в веб разработке: всякий выполняемый пользователем новый запрос превращается в свой собственный процесс или асинхронно координируется с прочими запросами; если какой- либо из этих запросов требует доступа к некоему общему ресурсу (например, к базе данных), в котором могут быть изменены данные, следует принимать во внимание совместную обработку.

**GIL.**

Рассмотрим те проблемы, с которыми сталкивались разработчики ядра Python в ранние дни Python и что породило необходимость самой GIL. В частности, существует существенная разница между программированием на Python и программированием на прочих популярных языках программирования относительно управления объектами в имеющемся пространстве оперативной памяти.

Например, в программном языке C++ некая переменная - на самом деле какое-то местоположение в имеющемся пространстве оперативной памяти в котором будет записано некое значение. Такая установка ведёт к тому факту, что при назначении некоторого определённого значения переменной не являющейся указателем, данный язык программирования действенно копирует это определённое значение в данное местоположение оперативной памяти (то есть в саму переменную). Кроме того, когда некоей переменной назначается другая переменная (которая не является указателем), само местоположение оперативной памяти последней переменной будет скопировано в местоположение первой; никакой связи между этими двумя переменными не будет поддерживаться после данного назначения.

С другой стороны Python рассматривает некую переменную просто как некое название, в то время как реальные значения его переменных изолированы в некоторой другой области в имеющемся пространстве памяти. Когда некое значение назначается какой-то переменной, эта переменная действенно получает некую ссылку на то самое местоположение в имеющемся пространстве памяти, где содержится это значение (даже хотя сам термин ссылки не применяется в том же самом смысле, как это имеет место в случае ссылки C++). Управление памятью в Python таким образом фундаментально отличается от той модели помещения некоторого значения в пространстве оперативной памяти, которое мы наблюдаем в C++.

Это означает, что при исполнении некоторой операции назначения Python просто взаимодействует со ссылками и переключается между ними - вместо того чтобы делать это со значениями. Кроме того, по этой же причине множество переменных может ссылаться на одно и то же значение, а изменения выполняемые одной переменной отражаются во всех прочих связанных переменных.

(дальше на доске нарисовать:

a = [1,2,3]

b = a

b.append(4)

print(a)

output: ?

)

**Развитие библиотек и фреймворков в сторону асинхронности**

Лучшей иллюстрацией движения мира к асинхронному подходу, пожалуй, является Django. Еще совсем недавно это был полностью синхронный фреймворк. С 3-й версии добавилась поддержка асинхронности и сейчас она становится все более полной. То есть, даже если вы будете писать проекты на скорее всего самом распространенном Python-фреймворке, то вам надо будет знать принципы асинхронности.

Многие сервисы изначально были написаны в синхронной парадигме. Например, возьмем Instagram, который был написан именно на синхронной версии Django. Казалось бы, как он справлялся с большой нагрузкой? Все благодаря команде Instagram, которая еще задолго до становления Django асинхронной, переписала его ядро и добавила поддержку асинхронности.

Вообще сам факт того, что Python добавил в 2015 году в свою стандартную библиотеку для работы с асинхронными операциями - asyncio, уже говорит о многом: команда разработчиков ядра Python понимает, что за этим будущее разработки. На самом деле — уже и настоящее. С каждым годом выпускается всё больше и больше фреймворков и библиотек для асинхронного программирования. Для «старых» синхронных модулей либо пишется аналог (например, asyntnt), либо в них добавляется асинхронная логика.