**به نام خدای رنگین کمان**

*خب اول از همه باید بدونیم که این مفاهیم برای کسانی هستش که حداقل یکسالی با پایتون کار کردند، و میخان دانش خودشون رو افزایش بدن.*

**: Multithreading in python**

**به صورت خلاصه یعنی ما چندین ترد را همزمان انجام بدیم و لازم نیست از بالا شروع کنی به کامپایل کردن تا پایین.(میتونی بگی اول بالا بعد آخری بعد وسطی و غیره)**

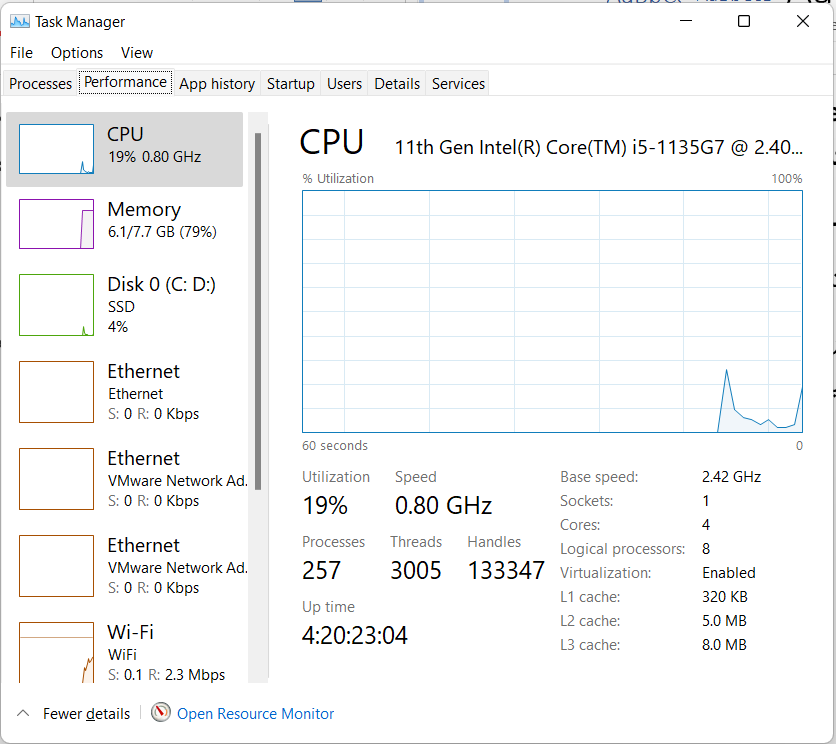
**پروسس (Process) : به هر برنامه که شما روی سیستم عاملتون ران میکنید یک پروسس میگویند، مثلا فایرفاکس یا همین ورد که من دارم توش مینویسم.**

**ترد (Tread) : به کوچک ترین قسمت هر پروسس یک ترد میگویند، مثلا توی فایرفاکس تو میتونی دانلود کنی که این خودش یه ترد هست، میتونی تب های مختلف باز کنی ، میتونی فیلم ببینی و.....**

**هر پراسس حداقل یدونه ترد داره که خوده اون برنامه هستش.**

مثلا ماشین یک پروسس هستش و فرمون و دنده و آینه و .... هر کدوم یک ترد هستند.

نکته : برای دیدن پروسس ها و ترد های ران شده روی سیستمت ، برو توی قسمت تسک منیجر و بعدش برو توی سی پی یو اونجا نوشته :



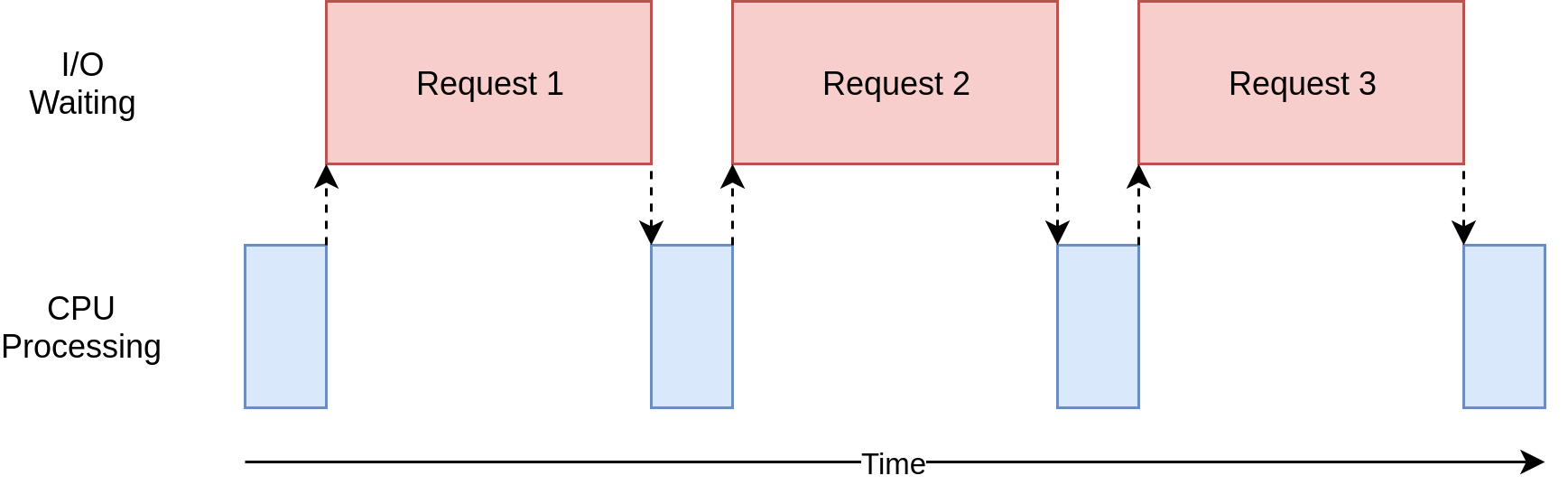
در پایتون نمیتونیم به اون صورت واقعی مالتی تردینگ رو پیاده سازی کنیم و پایتون توانایی اجرای دو ترد را در یک زمان را ندارد، بلکه مشخص میکنیم کدوم ترد اول شروع بشه ، بعدش صبر کنه تا یکی دیگه ران بشه و......

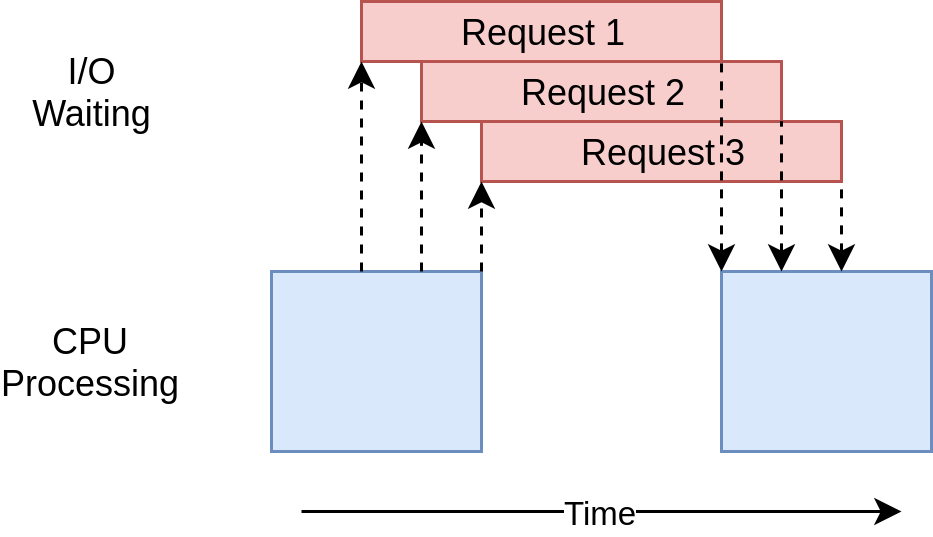
 برای multi-threading در پایتون از ماژول threading  استفاده میشود و ما در پایتون میتوانیم به صورت I/O bound ترد هارا مدیریت کنیم، در صورتی که برنامه شما به شکل I/O bound باشد زمانی را که صرف منتظر ماندن برای رسیدن پاسخ میکند را میتواند به کار دیگری مشغول شود.

اگر برنامه شما CPU bound باشد و همواره CPU شما مشغول کار باشد استفاده کردن از multi-threading نه تنها باعث افزایش سرعت نخواهد شد بلکه باعث افزایش فشار بروی CPU میشود و در نتیجه با کندی سرعت مواجه خواهید شد.

: I/O boundتوجه داشته باشید که فقط وقتی کاره I/O boundداشتید نیاز به multi-treading دارید.

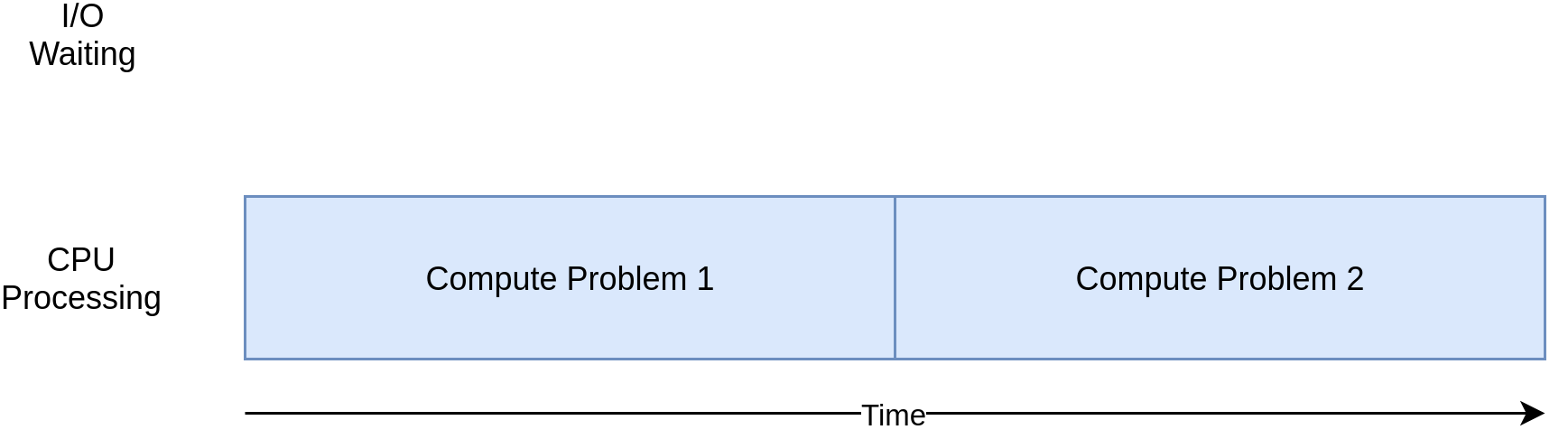
مثلا وقتی که یه درخواستی رو ارسال کردید و منتظر پاسخ اون هستش سی پی یو : (برای مثال کار کردن روی سیستم یکی دیگه با استفاده از سیستم خودت)





ما اینجا زمان هایی که سی پی یو منتظر یک عملی بوده رو استفاده کردیم و یک درخواست دیگه را ارسال کردیم.

: CPU boundدر اینجا کلا سی پی یو ما در حال استفاده و کار کردن هست و منتظر نمیماند که اینجا اگه multi-treadingبزنی بدتر کند میشی، پس در این حالت ما ازmulti-processing استفاده میکنیم که در این آموزش ما نیست.



**پیاده سازی با استفاده از صورت میگیره که دو تا روش داره :**

1. **با استفاده از ارسال یک شئ قابل فراخوانی به تابع سازنده (--init--).**

*passing a callable object to the constructor.*



بعد از استفاده از مالتی تریدینگ :



که در این حالت 3 ثانیه صرفه جویی شد.

1. **بازنویسی متودrun() توی ساب کلاسی که میسازی.**

*. overriding the run () method in a subclass*

**

بدون multi-treading برنامه ما باید 8 ثانیه طول میکشید ولی الان به 5 ثانیه کاهش یافته است.

**Daemon (دی مون):** در دنیای کامپیوتر به برنامه هایی که توی پس زمینه (background) اجرا میشوند دی مون میگویند.

اما در پایتون یعنی ترد هایی که برنامه حق داره اون هارو نادیده بگیره و ازشون بگذره و از برنامه خارج بشه.برای استفاده ازش هم باید قبل از استارت بنویسیمیش وگرنه ران تایم ارور میگیریم.(حالت دیفالتش هم فالسه)

متودjoin اولویت داره نسبت به متود daemon.

میتونیم از isDaemon()و setDaemon()هم استفاده کنیم ولی همون به شکل key argsبفرستیم خیلی بهتره.

**: Current-thread** اون تردی که فعال هست رو به ما نشون میده.

عبارتی که برمیگردونه یک اسم داره و یک عدد، که پیش فرض خودش بهش اسم داده ولی خودتم میتونی بهش اسمشو بدی.(ولی معمولا چنین کاری نمیکنند)

عددی که میده هم هر سری فرق میکنه و کاره خاصی انجام نمیده، مثلا برای اینکه بخوای توی یک دیکشنری ازشون استفاده بکنی میتونه بدردت بخوره.



**: enumerate** همه ترد های فعال و زنده را به صورت لیست برمیگردونه.



که توی این برنامه ما با اینکه دوتا ترد تعریف کردیم ولی به صورت کلی 3 تا ترد داریم:

1. ترد t1
2. ترد t2
3. ترد اصلی (main)

**ThreadPoolExecuter:**

مشکلی که کلاس Theardبرای ایجاد ترد ها دارد این است که نمیتواند تعداد بالایی ترد را داشته باشد و اگرم داشته باشد مجبورید کد بیشتری بنویسید، برای همین از کلاس ThreadPoolExecuter استفاده میکنیم. با استفاده از ماژول concurrent.futures ایمپورت میکنیم و به صورت یک context manager ازش باید استفاده کنیم.(with…. )

نکته : تابع مپ ، آرگومان های دومش رو به آرگومان اولیش که یک تابع هست پاس میده.



**Lock کردن در threading :**

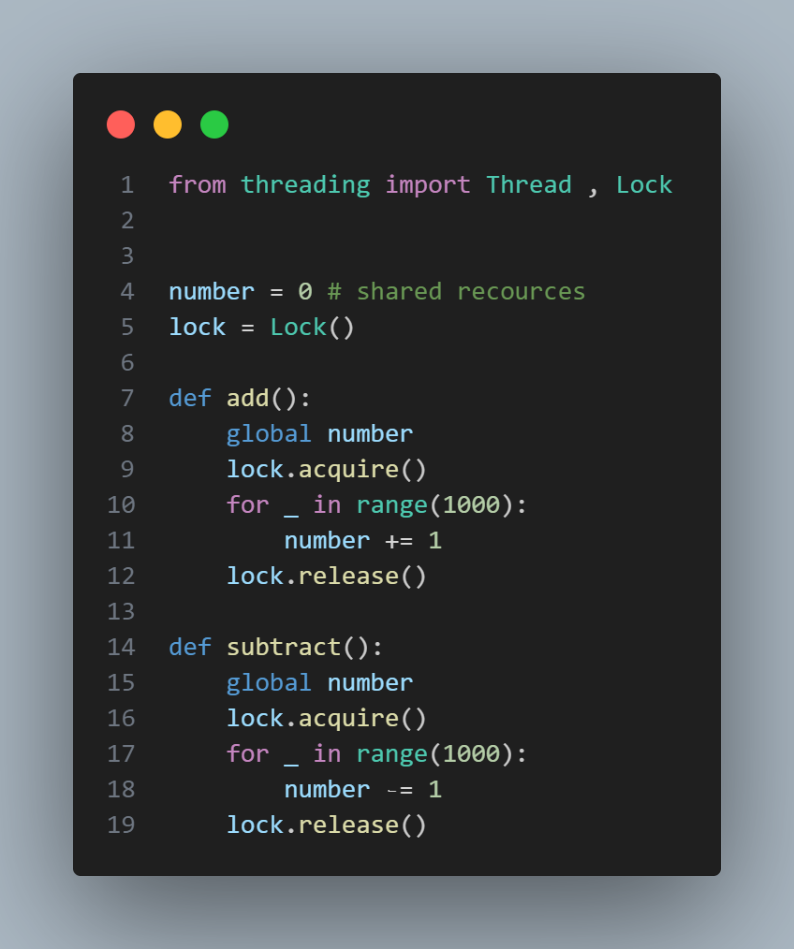
مشکل race conditionوقتی اتفاق میوفته که دو یا چند ترد به صورت همزمان به یک shared source دسترسی پیدا کنند. که باعث میشه اطلاعات غلطی رو برگردونند، برای حل این مشکل از کلاس Lock استفاده میکنیم. به صورتی که با استفاده از متودacquire برنامه را قفل کرده و بعد از اتمام کار با release برنامه را آزاد کند. در این حالت برنامه thread safe خواهد بود.

مشکل dead lock زمانی اتفاق میفتد که به صورت اشتباه برنامه ای که با استفاده از متد acquire قفل شده بود را دوباره با متد acquire قفل کنیم. در این حالت برنامه block شده و نه پاسخی برگشت داده میشود و نه برنامه تمام میشود. برای حل این مشکل پیشنهاد میشود که lock را به صورت یک context manager با متد with استفاده کنید.

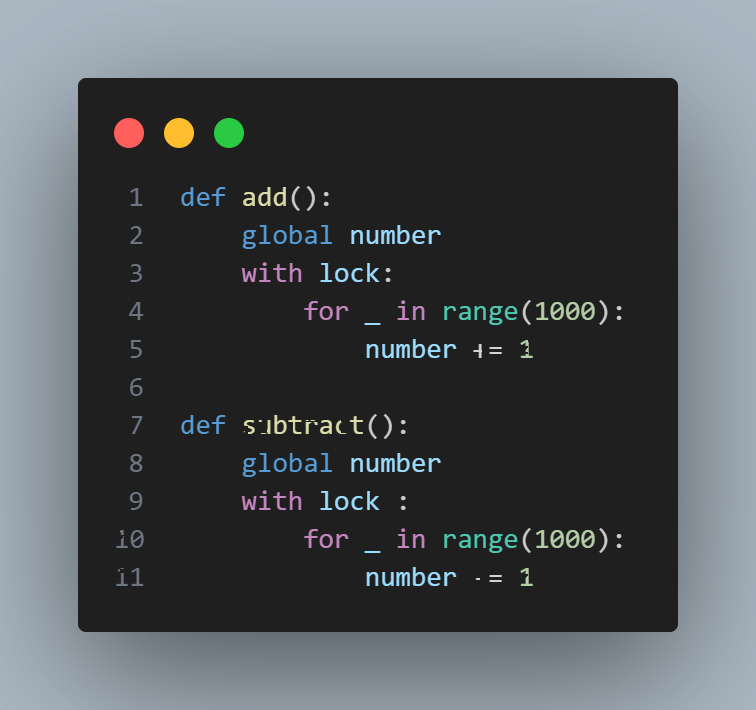


در این قسمت از کد چون دوتا تابع ما از یک ریسورس استفاده میکنند ممکنه که جواب غلط بدهند، مثلا 78 جای 0.

برای همین از استفادهlock میکنیم و قفل میکنیم و بعد کارش تموم شدش دوباره آزادش میکنیم، thread safe بودن در برنامه ها بسیار مورد افتخار هست و میتونی توی گیت هاب سرچ کنی راجبش.



برای اینکه راحت تر باشه کارمون و هی نخوایم این دوتارو صدا بزنیم و از Dead lock هم در امان باشیم به صورت context mangerاستفاده میکنیم :



**Semaphore:**

سمافور مثل لاک ها هستند، با این تفاوت که شما میتوانید روی ترد هایی متصل به shared recourses کنترل داشته باشید. در این کلاس یک شمارنده وجود دارد (پیش فرضش صفره)که تعداد ترد های متصل را کنترل میکند و هر بار که یک thread از متد acquire استفاده میکند یک عدد از این counter کم شده و هربار که از متد release استفاده میشود به این عدد اضافه میشود. زمانی که این عدد به صفر برسد هیچ thread دیگری پذیرفته نیست و باید منتظر بمانند. با context manager هم میشه ازش استفاده بکنیم یا مثلا اگه تعداد counter از acquire بیشتر باشه ، شمارنده منفی میشود و و باعث میشود در مرحله بعد تعداد threadهای بیشتری کار کنند.



برای حل این مشکل از کلاس BoundedSemaphore استفاده میکنیم که اگر شمارنده منفی شد پیغام خطای ValueError خواهد داد.

این سمافور ها وقتی بخوای از دیتابیس ها یا شبکه ها اطلاعات بخونی برامون مفید میشن.