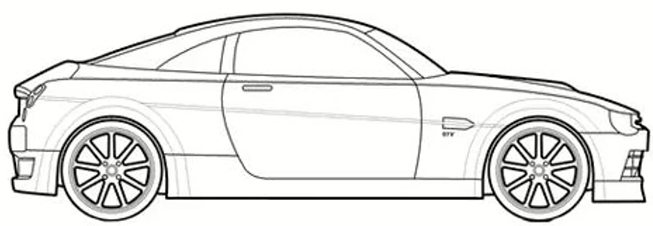
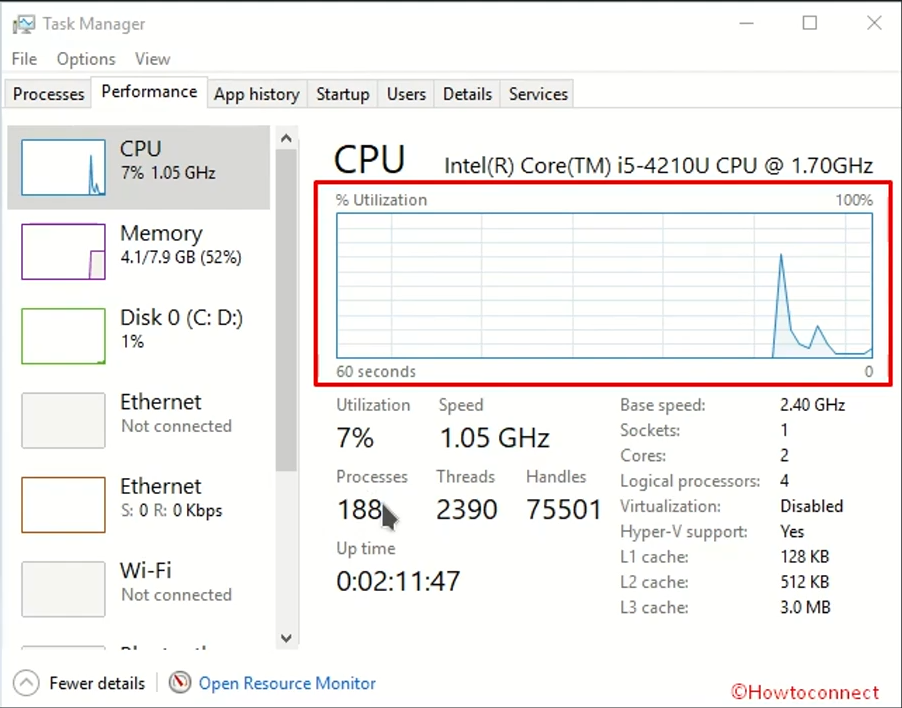
**Intro :**



در ابتدا لازم است با مفهوم process و thread آشنا شویم .شما یک ماشین را درنظر بگیرید که یک مجموعه واحدی میباشد داخل این ماشین ما ابزارها و قطعات مختلف را داریم (مثلاً لاستیک و چراغ ها) به ماشین process و به هرکدام از قطعات یک thread گفته می‌شود

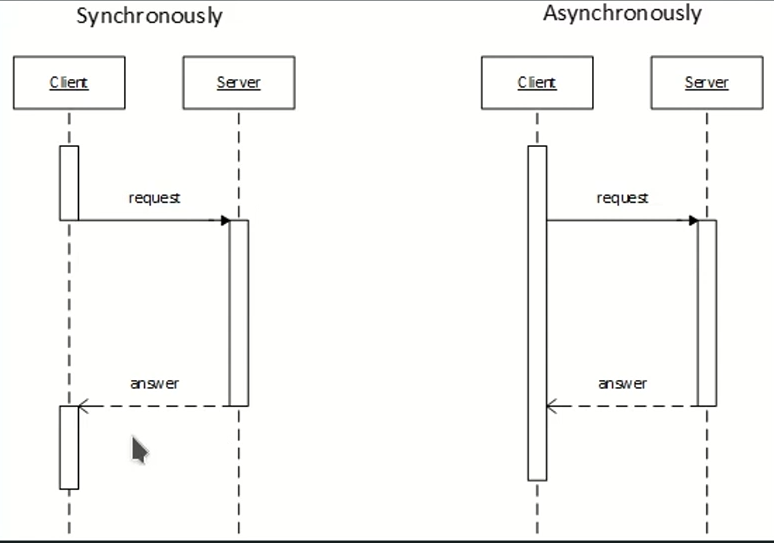
یا مثال دیگر مرورگر firefox یک process میباشد و هرtab این مرورگر که باز میباشد و کار خاصی را انجام میدهد thread گفته می‌شود



در ویندوز میتوانید در task manager تعداد processها و threadها را مشاهده کنید

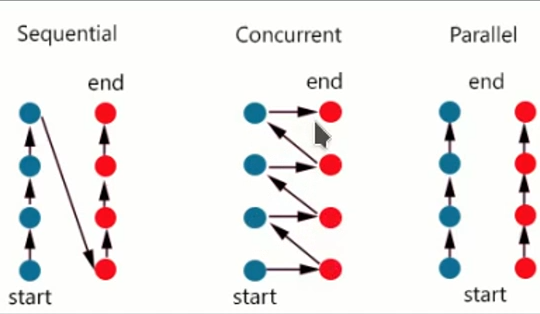
هر process حداقل یک thread را باید داشته باشد

در لینوکس هم میتوانیم با استفاده از دستور htop تعداد processها و threadها را ببینیم (البته process را با Tasks و thread را با thr نمایش میدهد)



در برنامه نویسی sync هنگامیکه client درخواستی را به server ارسال میکند تا زمانی که پاسخ از سرور بیاید باید منتظر بماند و هیچ کار دیگری انجام ندهد

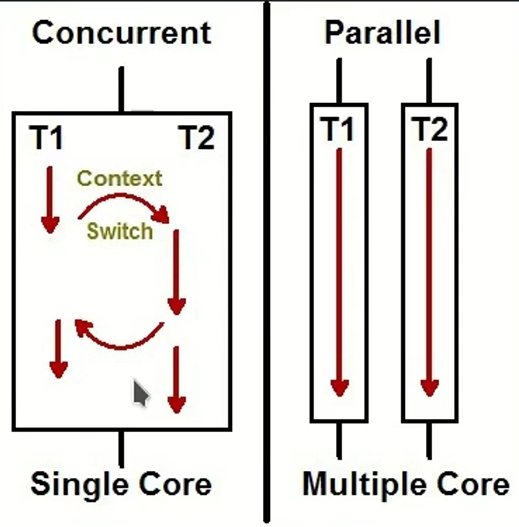
در برنامه نویسی async هنگامیکه client درخواست خود را به server ارسال میکند دیگر لازم نیست منتظر پاسخ server بماند و متواند کارهای دیگر خود را انجام دهد ( مثالا زمان ارسال پیامک احراز هویت)



sequential : مربوط به برنامه نویسی عادی و معمولی میباشد

concurrent : مربوط به برنامه نویسی async میباشد که در پایتون با استفاده از asyncio میتوانیم این مدل برنامه نویسی هایی را انجام دهیم

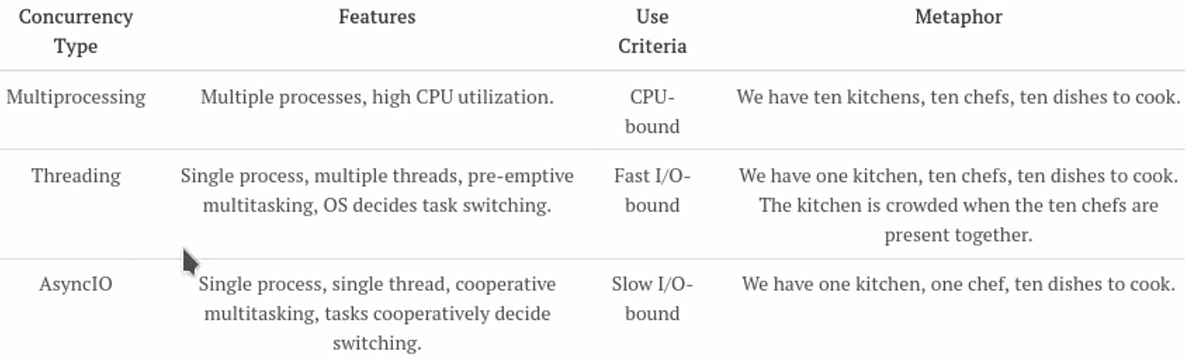
parallel : این مدل برنامه نویسی زمانی است که میخواهیم چندین thread یا process به طور همزمان اجرا شوند multitreading و multiprocessing در پایتون برای این مورد میباشد



زمانی از برنامه نویسی async استفاده می‌شود که بخواهیم از منابع i/o سیستم بهترین استفاده را بکنیم و منتظر نمانیم ( مثلاً میخواهیم درخواستی را به یک دیتابیس ارسال کنیم خب در این حالت i/o سیستم درگیر میباشد و میتوانیم با استفاده از برنامه نویسی async از زمانی که باید منتظر پاسخ دیتابیس باشیم استفاده کنیم) ( IO bound )

زمانی از برنامه نویسی parallel استفاده میکنیم که منابع cpu زیادی در اختیار داشته باشیم دراین حالت میتوانیم با استفاده از هر core cpu چندین process یا thread را بصورت همزمان انجام دهیم (cpu bound )

توجه شود که از multi treading میتوانیم در موارد ( IO bound ) نیز استفاده کنیم در این حالت سیستم عامل تصمیم میگیرد که چگونه بین taskها جابجا شود اما در async این برنامه و taskها میباشند که باعث جابجایی میشوند

کتابخانه asyncio از دو بخش high level و low level تقسیم‌بندی می‌شود که high level برای برنامه نویسان عادی میباشد و low level برای برنامه نویسانی میباشد که framework یا library طراحی میکنند

=======================================================================

**coroutines :**

coroutine همان فانکشن python میباشد منتها میتواند توی بدنه خودش stop بشه ما بریم یک کار دیگه ای رو انجام بدیم بعد برگردیم و ادامه اون function رو بریم جلو

برای ایجاد coroutine قبل از کلمه def باید async نوشته شود در ادامه داخل بدنه فانکشن این قابلیت رو باید داشته باشد که در یک مرحله‌ای stop بشه بره یک کار دیگه ای رو انجام بده اون جایی که ممکن است برنامه stop شود را با await مشخص میکنیم

به اون قسمتی که برنامه متوقف می‌شود IO block گفته می‌شود

async و await که درباره آن صحبت شد ربطی به پکیچ asyncio ندارد و در هسته اصلی پایتون میباشد

async def one(name):  
 print(f“Hello {name}”)

one(“ahmad”)

هنگامیکه کد بالا را اجرا کنیم درواقع اخطار RuntimWarning دریافت میکنیم مبنی بر اینکه هیچ awaitی در coroutine one وجود ندارد

coroutine ها را نمی‌توان بصورت مستقیم اجرا کرد و ۳ روش برای اجرای coroutine وجود دارد :

- run : به وسیله دستور run میتوانیم یک coroutine را اجرا کنیم :

asyncio.run(one(“ahmad”))

توجه داشته باشید که استفاده از run به تنهایی دستورات coroutineها را به صورت async اجرا نمیکند

-create\_task‌ : از این دستور استفاده مکنیم و برای هر اجرای coroutine یک task درست میکنیم و به صورت await اجرا میکنیم ( در حالت دومی که نوشته شده است هنگامیکه ahmad قرار است print شود ۲ ثانیه sleep می‌شود به همین دلیل سریعاً به دنبال اجرای sajjad می‌رود و دوباره همینجا ۲ ثانیه sleep می‌شود به همین خاطر تمام این ۲ ثانیه ها روی هم قرار گرفته‌اند و بعد از اتمام ۲ ثانیه هر دو print بصورت همان اجرا می‌شوند و زمان کلی اجرا ۲ ثانیه می‌باشد )  
تسک های create شده در event loop ثبت می‌شوند

=======================================================================

**cancel :**

در فایل cancel مثالی برای cancel کردن یک task نوشته شده است به این صورت که اگر اجرای یک task بیشتر از ۵ ثانیه طول کشید task به وسیله فانکشن مربوطه cancel می‌شود ( فانکشن مربوطه cancel میباشد و خطای CancelledError را برمیگرداند )

فانکشن done خروجی boolean دارد به این صورت که اگر یک task با موفقیت تمام شود مقدار true و اگر هنوز تمام نشده باشد مقدار false دارد

برنامه‌ای که در حالت اول نوشته شده است به صورت دستی بررسی میکند که مدت زمان اجرای یک تسک چقدر طول کشیده است

اما در حالت دوم از wait\_for استفاده کرده ایم ورودی های این فانکشن دو مقدار میباشد یکی coroutine و دومی مقدار زمان مورد نظر که در این حالت اگر خطای TimeoutError را برمیگرداند

فانکشن cancelled خروجی boolean دارد به این صورت که اگر یک task با موفقیت اجرا شود و cancel نشود مقدار false و اگر cancel شود مقدار true را برمیگرداند

در حالت سوم از cancel شدن تسک به وسیله فانکشن shield جلوگیری میکنیم توجه داشته باشید که برای ادامه ی اجرا شدن تسک باید دوباره صدا زده شود (خط ۵۳ )

=======================================================================