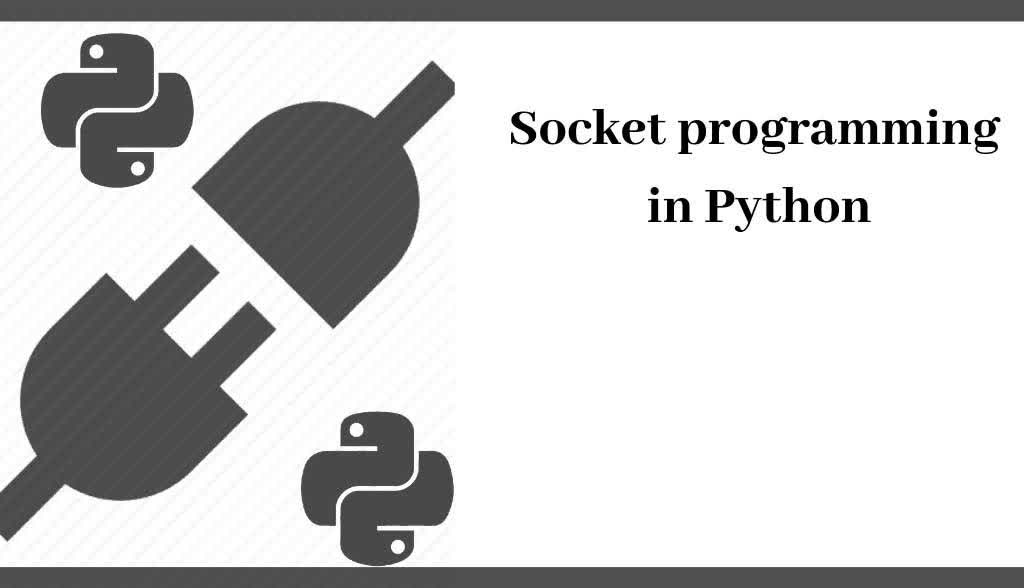
**بسم الله الرحمان الرحیم**

درس:توسعه نرم افزار

موضوع: socket programing using phyton

استاد:سرکار خانم ثریا عناینی

دانش پژوهان: فاطمه ایرجی ، مریم مزینانی ، آرزو قنبری



سوکت پروگرمینگ و برنامه‌نویسی سوکت در شبکه سوکت پروگرمینگ، یا همان Socket Programming، به فرایند تبادل اطلاعات بین دو برنامه در شبکه می‌پردازد. این فرایند بر اساس استفاده از سوکت‌ها، که به صورت نقطه به نقطه بین دو میزبان شبکه ایجاد می‌شوند، انجام می‌شود. برنامه‌نویسی سوکت، روشی قدرتمند برای ارتباط و تبادل داده بین سرور و کلاینت است. در این روش، یک سوکت بر روی هر دو سمت ارتباط (سرور و کلاینت) ایجاد می‌شود. سرور برای گوش دادن به درخواست‌ها و ارسال پاسخ‌ها، و کلاینت برای ارسال درخواست و دریافت پاسخ استفاده می‌شود. برای برنامه‌نویسی سوکت، معمولاً از زبان‌های برنامه‌نویسی متعددی استفاده می‌شود، از جمله:

**1** **زبان C/C++:** با استفاده از کتابخانه‌های مانند Socket و Winsock در سیستم‌عامل Windows و POSIX API در سیستم‌عامل‌های مبتنی بر یونیکس.

**2 زبان Java:** با استفاده از کلاس‌ها و رابط‌هایی که در بستر JDK برای برنامه‌نویسی سوکت ارائه شده است، مانند Socket و ServerSocket.

**زبان Python:** با استفاده از کتابخانه‌های مانند socket و asyncio که قابلیت برنامه‌نویسی سوکت را در پایتون فراهم می‌کنند. برنامه‌نویسی سوکت در شبکه به برنامه‌نویس اجازه می‌دهد از جریان داده از طریق شبکه استفاده کند و برنامه‌هایی بسازد که بتوانند به صورت دوطرفه با سرورها و دیگر کلاینت‌ها ارتباط برقرار کنند. به این ترتیب، برنامه‌ها می‌توانند اطلاعات را به صورت زنده (real-time) ارسال و دریافت کنند و با یکدیگر تعامل داشته باشند. استفاده از برنامه‌نویسی سوکت در شبکه باعث می‌شود تا برنامه‌ها قابلیت‌های پیچیده‌تری نظیر ارسال و دریافت داده‌های چندرسانه‌ای، استفاده همزمان از چندین ارتباط شبکه و ایجاد ارتباطات امن‌تر را داشته باشند.

این تکنیک پیشرفته برنامه‌نویسی در شبکه به برنامه‌های شبکه اجازه می‌دهد تا به صورت کامل از قابلیت‌های شبکه استفاده کنند و بر اساس نیاز‌های خود، ارتباطات سفارشی و قدرتمند ایجاد کنند.

می شود. همانطور که می دانیم، Python دارای نحو آسان و محیط کاربر پسند است که توسعه یا تجزیه و تحلیل داده ها را ساده می کند.

پایتون برای برنامه نویسی شبکه نیز سازگار است و از کتابخانه های زیادی تشکیل شده است که دسترسی سطح بالاتری به پروتکل های شبکه در سطح برنامه خاص مانند HTTP، FTP و غیره را ارائه می دهند. ابتدا باید اصطلاحات اساسی شبکه را درک کنیم.

بهبود در برنامه نویسی سوکت با استفاده از پایتون در سال های اخیر بر افزایش عملکرد، امنیت و مقیاس پذیری متمرکز شده است. ترکیب چارچوب‌ها و کتابخانه‌های ناهمزمان، مانند asyncio و aiohttp، امکان برنامه‌نویسی همزمان و مؤثر شبکه‌ها را فراهم کرده است.

برنامه نویسان پایتون به طور فزاینده ای از برنامه نویسی ناهمزمان استفاده می کنند زیرا آنها را قادر می سازد کدهای همزمان و غیر مسدود کننده ایجاد کنند که به طور مؤثر اتصالات متعددی را مدیریت می کند. پایه ای برای برنامه نویسی ناهمزمان توسط بسته asyncio در پایتون ارائه شده است که امکان توسعه حلقه های رویداد و کوروتین ها را فراهم می کند. Asyncio امکان پردازش اتصالات متعدد را بدون نیاز به رشته یا فرآیند در برنامه نویسی سوکت، بهبود عملکرد و استفاده از منابع فراهم می کند.

امنیت علاوه بر بهبود سرعت، به یکی از اجزای ضروری برنامه نویسی سوکت تبدیل شده است. پایتون چندین ماژول و پروتکل را برای اتصال شبکه ایمن ارائه می دهد، از جمله TLS (Transport Layer Security) و SSL (Secure Socket Layer). این فناوری‌های رمزگذاری، داده‌های انتقال‌یافته از طریق شبکه را از دسترسی و دستکاری غیرمجاز محافظت می‌کنند. حفاظت از اطلاعات حساس و حفظ یکپارچگی سیستم های شبکه با گنجاندن پروتکل های ارتباطی امن در برنامه نویسی سوکت با استفاده از پایتون آسان تر می شود.

علاوه بر این، مقیاس پذیری یک عامل کلیدی در برنامه نویسی سوکت معاصر بوده است. چارچوب‌های توسعه وب رایج پایتون مانند Flask و Django عملکرد داخلی را برای مدیریت درخواست‌ها و پاسخ‌های شبکه ارائه می‌کنند. این چارچوب‌ها پیچیدگی برنامه‌نویسی سوکت را کنترل می‌کنند و به برنامه‌نویسان اجازه می‌دهند تا روی ایجاد برنامه‌های وب مقیاس‌پذیر با ویژگی‌هایی مانند تعادل بار، ذخیره‌سازی حافظه پنهان و محاسبات توزیع‌شده تمرکز کنند.

**شبکه کامپیوتری چیست؟**

شبکه های کامپیوتری امکان اشتراک منابع و ارتباط بین چندین سیستم یا دستگاه کامپیوتری را فراهم می کنند. زیرساختی که این دستگاه ها را به هم متصل می کند و امکان تبادل اطلاعات بین آنها را فراهم می کند شبکه کامپیوتری نامیده می شود. برای تسهیل ارتباط و همکاری، باید یک اتصال شبکه برقرار شود.

بیایید از برقراری تماس تلفنی به عنوان مثال برای درک بهتر شبکه های کامپیوتری استفاده کنیم. هنگامی که ما تماس تلفنی برقرار می کنیم، یک ارائه دهنده خدمات شبکه باید بین دستگاه ما و دستگاه گیرنده پیوند ایجاد کند. الزامات مشابهی برای شبکه های رایانه ای اعمال می شود که در آن دستگاه ها باید به یکدیگر متصل شوند تا داده ها را مبادله کنند و به منابع مشترک دسترسی داشته باشند.

شبکه های کامپیوتری را می توان بر اساس اندازه و منطقه تحت پوشش خود به چند دسته تقسیم کرد. انواع شبکه های معمولی شامل موارد زیر است:

**شبکه محلی (LAN): LAN** شبکه ای است که یک منطقه محدود از زمین، مانند محوطه دانشگاه، ساختمان اداری یا سایر سازه های مشابه را پوشش می دهد. این دستگاه ها را در یک فضای محدود به هم متصل می کند و امکان ارتباط سریع و موثر را فراهم می کند.

**شبکه گسترده منطقه (WAN):** یک WAN چندین شبکه محلی را به هم متصل می کند و اغلب منطقه جغرافیایی بیشتری را در بر می گیرد. این امکان را برای گجت ها فراهم می کند تا در شهرها، کشورها و حتی قاره ها با یکدیگر ارتباط برقرار کنند. یک WAN گسترده و گسترده در سراسر جهان با اینترنت نمونه است.

شبکه منطقه شهری (MAN): با توجه به پوشش جغرافیایی، یک MAN بین یک LAN و یک WAN قرار دارد. این دستگاه ها را در سراسر یک شهر یا منطقه شهری به هم متصل می کند و امکان اشتراک گذاری و ارتباطات گسترده تری از منابع را فراهم می کند.

PAN**: PAN** (شبکه منطقه ای شخصی) شبکه ای است که دستگاه های موجود در فضای شخصی افراد مانند تلفن هوشمند، ساعت هوشمند و رایانه را به هم متصل می کند.

شبکه‌های بی‌سیم: شبکه‌های بی‌سیم بدون سیم فیزیکی با استفاده از فناوری‌های ارتباطی بی‌سیم مانند Wi-Fi، بلوتوث یا شبکه‌های سلولی، اتصالات ایجاد می‌کنند. آنها دستگاه ها را قادر می سازند تا با انعطاف پذیری و تحرک ارتباط برقرار کنند.

شبکه های کامپیوتری برای کارآمدی به اجزای سخت افزاری و نرم افزاری مختلفی نیاز دارند. این موارد شامل موارد زیر است:

**NIC** :کارت های رابط شبکه (NIC) اشیایی فیزیکی هستند که برای اتصال دستگاه ها به شبکه استفاده میشوند. آنها به دستگاه های متصل به شبکه یک رابط فیزیکی برای ارسال و دریافت داده می دهند.

روترها و سوئیچ ها: روترها و سوئیچ ها ابزارهای شبکه ای هستند که به مسیریابی مجدد و هدایت ترافیک شبکه کمک می کنند. سوئیچ‌ها با مسیریابی داده‌ها به گیرنده صحیح، ارتباط را در داخل شبکه فعال می‌کنند، در حالی که روترها چندین شبکه را به هم متصل می‌کنند و اجازه انتقال داده‌ها را بین آنها می‌دهند.

پروتکل ها: قوانین و هنجارهای ارتباط بین دستگاه ها در یک شبکه توسط پروتکل ها یا استانداردهای شبکه تعریف می شوند. پروتکل های رایج عبارتند از HTTP (پروتکل انتقال ابرمتن)، که برای ارتباطات آنلاین استفاده می شود، و TCP/IP (پروتکل کنترل انتقال/پروتکل اینترنت)، پروتکل پایه گذار اینترنت.

امنیت شبکه: برای محافظت از داده های حساس و جلوگیری از دسترسی غیرمجاز، اطمینان از امنیت شبکه های کامپیوتری ضروری است. شبکه‌های خصوصی مجازی (VPN)، فایروال‌ها، روش‌های رمزگذاری و احراز هویت نمونه‌هایی از روش‌های امنیت شبکه هستند.

شبکه‌های رایانه‌ای مانند فناوری به پیشرفت خود ادامه می‌دهند. فناوری‌های نوظهور مانند اینترنت اشیا (IoT) و 5G با اتصال تعداد فزاینده‌ای از دستگاه‌ها و امکان ارتباطات سریع‌تر و مطمئن‌تر، شبکه‌ها را تغییر می‌دهند.

**شبکه های بی سیم**

شبکه‌های بی‌سیم به عنوان شبکه‌های بی‌سیم شناخته می‌شوند که داده‌ها را انتقال می‌دهند و با استفاده از فناوری‌های ارتباطی بی‌سیم بین دستگاه‌ها ارتباط ایجاد می‌کنند، زیرا نیازی به کابل فیزیکی یا اتصالات سیمی نیست. این شبکه‌ها به دستگاه‌ها اجازه می‌دهند تا داده‌ها را به‌صورت بی‌سیم متصل کرده و تبادل کنند و انعطاف‌پذیری، قابلیت حمل و راحتی را در سناریوهای مختلف به ارمغان بیاورند.

انتقال داده بین دستگاه ها از طریق شبکه های بی سیم با استفاده از امواج رادیویی، سیگنال های مادون قرمز و سایر روش های ارتباطی بی سیم انجام می شود. وای فای محبوب ترین شکل فناوری شبکه های بی سیم است. Wi-Fi دستگاه‌ها را قادر می‌سازد به یک شبکه بی‌سیم محلی، که اغلب توسط یک روتر بی‌سیم عرضه می‌شود، بپیوندند و برای دسترسی به اینترنت یا برقراری ارتباط با سایر دستگاه‌های موجود در شبکه به آن متصل شوند. شبکه‌های Wi-Fi اکنون در خانه‌ها، مشاغل، مناطق عمومی و سایر تنظیمات وجود دارند و دسترسی بی‌سیم را برای دستگاه‌های مختلف از جمله تلفن‌های هوشمند، لپ‌تاپ، تبلت و دستگاه‌های IoT ارائه می‌کنند.

**بلوتوث** یکی دیگر از فناوری های بی سیم پرکاربرد است که به ویژه برای ارتباط نزدیک بین دستگاه ها ساخته شده است. بلوتوث اتصال بی سیم بین دستگاه های مجاور را تسهیل می کند و به شما امکان می دهد تلفن هوشمند خود را به هدفون های بی سیم متصل کنید یا فایل ها را بین آنها جابجا کنید.

یکی دیگر از شبکه‌های بی‌سیم که دسترسی به اینترنت و ارتباطات تلفن همراه را به دستگاه‌ها در یک منطقه جغرافیایی بزرگ ارائه می‌دهد، شبکه سلولی است که شامل شبکه‌های 3G، 4G و 5G است. شبکه های سلولی به شبکه ای از ایستگاه های پایه یا برج های سلولی برای انتقال و دریافت سیگنال بین دستگاه ها و زیرساخت شبکه نیاز دارند. این به کاربران امکان می دهد در حین حرکت تماس تلفنی برقرار کنند، پیامک ارسال کنند و به اینترنت دسترسی داشته باشند.

**اصطلاحات شبکه های کامپیوتری**

این اصطلاحات برای برنامه نویسی شبکه بسیار ضروری هستند. اگر با آنها آشنایی ندارید، برای آموزش بیشتر آنها را درک کنید.

**پروتکل اینترنت**

انتقال و دریافت داده ها از طریق اینترنت توسط مجموعه ای از قوانین و رویه ها به نام پروتکل اینترنت (IP) کنترل می شود. این سنگ بنای خانواده پروتکل اینترنت و اساس ارتباطات آنلاین است.

IP به هر دستگاه متصل یک آدرس IP متفاوت اختصاص می دهد تا سیستمی برای شناسایی منحصر به فرد ارائه دهد. آدرس های IP اعدادی هستند که شناسایی و مکان دستگاه ها را در شبکه امکان پذیر می کنند.

مسیریابی بسته های داده در سراسر شبکه ها توسط IP انجام می شود. هنگام ارسال داده از طریق اینترنت، به بسته های کوچکتر با اطلاعات آدرس دهی ضروری و بخشی از پیام اصلی تقسیم می شود. سپس این بسته ها به صورت جداگانه ارسال می شوند و مسیرهایی که برای رسیدن به آنجا طی می کنند ممکن است متفاوت باشد. IP از آدرس های IP برای مسیریابی بسته ها در سراسر شبکه ها استفاده می کند و تکه تکه شدن و مونتاژ مجدد بسته ها را مدیریت می کند تا تضمین کند که هر بسته به مقصد مناسب تحویل داده می شود.

UDP (پروتکل داده‌های کاربر)

TCP (پروتکل کنترل انتقال)

پروتوکل دیتاگرام کاربر

**UDP** همچنین ویژگی نامرتب بودن را دارد. دیتاگرام‌ها ممکن است فقط گاهی توسط گیرندگانشان به همان ترتیبی که ارسال شده‌اند دریافت کنند. UDP با هر دیتاگرام به عنوان یک موجودیت مجزا برخورد می کند و هیچ ترتیب یا ترتیب بسته خاصی را تحمیل نمی کند. این کیفیت ممکن است در شرایطی که انتقال به موقع داده مهمتر از سفارش دقیق است مفید باشد.

از آنجایی که نیازی به برقراری ارتباط قبل از تحویل داده نیست، UDP سبک وزن دیده می شود. UDP بر اساس بهترین تلاش و بدون هزینه مدیریت اتصال اجرا می شود، برخلاف TCP، که برای ایجاد یک اتصال قابل اعتماد نیاز به یک فرآیند دست دادن دارد. UDP به دلیل ماهیت سبک آن در استفاده از منابع و سربار پردازش موثرتر است. مسیریابی بسته های داده در سراسر شبکه ها توسط IP انجام می شود. هنگام ارسال داده از طریق اینترنت، به بسته های کوچکتر با اطلاعات آدرس دهی ضروری و بخشی از پیام اصلی تقسیم می شود. سپس این بسته ها به صورت جداگانه ارسال می شوند و مسیرهایی که برای رسیدن به آنجا طی می کنند ممکن است متفاوت باشد. IP از آدرس های IP برای مسیریابی بسته ها در سراسر شبکه ها استفاده می کند و تکه تکه شدن و مونتاژ مجدد بسته ها را مدیریت می کند تا تضمین کند که هر بسته به مقصد مناسب تحویل داده می شود.

بسته های داده ای که UDP استفاده می کند دیتاگرام نامیده می شود. اطلاعات مورد نیاز برای انتقال و تایید یکپارچگی به عنوان یک موجودیت مجزا در هر دیتاگرام استفاده می شود. این دیتاگرام ها یک به یک منتقل می شوند و ممکن است چندین مسیر را به مقصد خود طی کنند. UDP تکنیک های ارسال مجدد یا تصحیح خطا را ارائه نمی دهد. بنابراین لایه برنامه باید از یکپارچگی و قابلیت اطمینان داده ها در صورت نیاز اطمینان حاصل کند.

**پروتکل کنترل انتقال**

یک پروتکل اتصال گرا به نام TCP (پروتکل کنترل تحویل) تحویل داده های قابل اعتماد و سازماندهی شده را از طریق اینترنت امکان پذیر می کند. قبل از شروع انتقال داده بر اساس ایده دست دادن، با ایجاد ارتباط با میزبان دیگری کار می کند.

**آدرس های IP و پورت ها**

یک شبکه، از جمله اینترنت، دستگاه‌های مرتبط با آدرس‌های IP خود را که هویت‌های عددی متمایز هستند، اختصاص می‌دهد. آنها به عنوان مکان و اطلاعات شناسایی ابزارهایی مانند رایانه ها، سرورها و دستگاه های اینترنت اشیا عمل می کنند. برای ایجاد ارتباط بین دستگاه ها به صورت آنلاین، آدرس های IP ضروری است.

آدرس های IP و پورت ها با هم کار می کنند تا ارتباط دقیق بین دستگاه ها و برنامه های شبکه را فراهم کنند. داده ها به یک آدرس IP خاص تحویل داده می شوند و هنگام انتقال از طریق اینترنت به یک پورت خاص در دستگاه هدف متصل به یک برنامه یا سرویس خاص ارسال می شوند. ممکن است برنامه یا سرویس مورد نظر به درستی داده ها را دریافت و مدیریت کند.

**شماره پورت**

22 پوسته امن

23 سرویس ورود از راه دور Telnet

25 پروتکل انتقال ایمیل ساده (SMTP)

53 سرویس سیستم نام دامنه (DNS).

80 پروتکل انتقال متن بیش از حد (HTTP) که در WWW استفاده می شود.

**دیواره آتش**

فایروال یک مانع بین شبکه داخلی و منابع خارجی مانند اینترنت است و برای امنیت شبکه ضروری است. وظایف اصلی آن شامل نظارت بر ترافیک شبکه ورودی و خروجی و اجرای مجموعه ای از قوانین امنیتی از پیش تعیین شده است. فایروال بسته های داده را ارزیابی می کند و تصمیم می گیرد که آیا ترافیک را طبق قوانین تعریف شده مجاز یا متوقف کند.

**چرا پایتون برای برنامه نویسی شبکه ای؟**

پایتون، یک زبان برنامه نویسی قدرتمند و سازگار، به طور باورنکردنی برای برنامه نویسی شبکه محبوب شده است. چند دلیل وجود دارد که چرا پایتون یک تصمیم مطلوب برای برنامه نویسی شبکه است، و در این توضیح، بخشی از متغیرهای کلیدی را بررسی خواهیم کرد.

اول اینکه پایتون به دلیل خوانایی و سادگی یک زبان عالی برای برنامه نویسی شبکه است. نقطه گذاری کامل و طبیعی آن به مهندسان اجازه می دهد تا کدی را بنویسند که ساده باشد و با آن هماهنگ باشد. به دلیل سادگی، کد پایتون کمتر باگ دارد زیرا احتمال کمتری دارد که حاوی خطا باشد. علاوه بر این، کتابخانه استاندارد گسترده پایتون مجموعه ای جامع از ماژول ها و توابع خاص شبکه را ارائه می دهد که اجرای پروتکل های شبکه و ارتباطات را ساده می کند.

اکوسیستم گسترده پایتون و پشتیبانی جامعه مزیت مهم دیگری است. کتابخانه ها و فریمورک های شخص ثالث زیادی برای پایتون وجود دارد که می توانند واقعاً به وظایف برنامه نویسی شبکه کمک کنند. انتزاعات سطح بالا برای ارتباطات شبکه توسط کتابخانه هایی مانند "socket" و "asyncio" ارائه شده است، که مدیریت سوکت ها و پیاده سازی پروتکل های شبکه را ساده تر می کند. سیستم‌هایی مانند «Curved» و «Django» ابزارها و قطعات گسترده‌ای را برای ساخت برنامه‌های پیچیده سازمان، از جمله سرورهای وب و APIها ارائه می‌دهند. توسعه دهندگان می توانند از راه حل های از قبل موجود برای تسریع توسعه و کاهش تلاش کلی به لطف در دسترس بودن این منابع استفاده کنند.

علاوه بر این، سازگاری پایتون آن را قادر می سازد تا به طور یکپارچه با فناوری ها و زبان های مختلف ادغام شود. به لطف پشتیبانی از انواع مکانیسم های ارتباطی بین زبانی مانند API ها، سوکت ها و صف های پیام، به برنامه های شبکه اجازه می دهد تا با اجزای نوشته شده به زبان های مختلف تعامل داشته باشند. زمانی که زیرساخت یا ابزارهای شبکه موجود به زبان هایی غیر از پایتون نوشته شده باشند، این قابلیت به ویژه مفید است. توانایی پایتون برای کار با سایر فناوری‌ها، ادغام و کار یکپارچه با یکدیگر را آسان‌تر می‌کند، که باعث می‌شود وظایف برنامه‌نویسی شبکه در کل مؤثرتر باشد.

در نهایت، محبوبیت پایتون برای برنامه نویسی شبکه با مستندات گسترده و جامعه بزرگ آن تقویت شده است. منابع آنلاین، آموزش‌ها و انجمن‌های متعددی برای توسعه‌دهندگان برای جستجوی راهنمایی و به اشتراک گذاشتن دانش در دسترس جامعه فعال و پر جنب و جوش پایتون است. توسعه دهندگان به لطف مستندات گسترده می توانند به راحتی به اطلاعات دسترسی پیدا کنند و راه حل هایی برای مشکلات رایج شبکه بیابند.

**پایه برنامه نویسی سوکت**

ما مفاهیم اولیه شبکه را آموخته ایم و اصطلاحات اولیه شبکه را درک می کنیم. قبل از شروع برنامه نویسی شبکه پایتون، باید مقدمه سوکت را مرور کنیم.

برنامه نویسی شبکه و سوکت موضوعات بزرگی هستند. شبکه های کامپیوتری موضوعات زیادی برای بررسی دارد، اما در اینجا به برنامه نویسی اصلی شبکه با استفاده از پایتون می پردازیم.

بیایید سوکت را درک کنیم و ببینیم آنها چه هستند و چرا از آنها استفاده می کنیم.

**سوکت چیست؟**

ابتدا باید اتصال به اینترنت را یاد بگیریم. اتصال به اینترنت اساساً برای اتصال دو نقطه پایانی در سراسر اینترنت برای به اشتراک گذاری داده و موارد دیگر استفاده می شود.

سوکت یک رابط نرم افزاری است که دو کامپیوتر را قادر می سازد تا در یک شبکه مانند اینترنت با یکدیگر ارتباط برقرار کنند. این یک نقطه پایانی برای انتقال و دریافت داده بین برنامه ها در دستگاه های مختلف است. اتصال به اینترنت این امکان را فراهم می کند که یک فرآیند از کامپیوتر C1 می تواند با یک فرآیند از یک کامپیوتر C2 ارتباط برقرار کند. از ویژگی های زیر تشکیل شده است.

**قابل اعتماد:** یک روش قابل اعتماد برای انتقال داده از طریق اتصال به اینترنت توسط سوکت ها ارائه می شود. آنها ضمن ایجاد یک مسیر ارتباطی امن بین مشتری و سرور، انتقال داده های دقیق و بدون خطا را ارائه می دهند.

**نقطه به نقطه:** سوکت ها دو نقطه انتهایی خاص را به صورت نقطه به نقطه به هم متصل می کنند. داده ها را می توان در محل مورد نظر منتقل و دریافت کرد زیرا آدرس IP و شماره پورت آن می تواند هر نقطه پایانی را تشخیص دهد.

**Full Duplex:** ارتباط فول دوبلکس یکی از ویژگی های اصلی سوکت ها است. این بدان معناست که داده ها می توانند به طور همزمان بین مشتری و سرور به هر دو روش منتقل شوند. داده ها را می توان هم توسط مشتری و هم توسط سرور ارسال و دریافت کرد و ارتباط تعاملی و دو طرفه را امکان پذیر می کند.

نقاط انتهایی کانال های ارتباطی دو طرفه و نقطه به نقطه توسط سوکت ها ارائه می شوند که برای ارتباطات شبکه ضروری هستند. آنها اغلب برای انتقال پیام از طریق شبکه استفاده می شوند. هنگامی که یک کلاینت، مانند یک مرورگر وب، به سروری مانند www.javatpoint.com متصل می شود، دو نقطه پایانی درگیر می شوند: سوکت مشتری و سوکت سرور.

ایده سوکت ها برای اولین بار در سال 1971 ظاهر شد و بعداً به رابط برنامه نویسی کاربردی سوکت برکلی (API) تبدیل شد که در سال 1983 در سیستم عامل توزیع نرم افزار برکلی (BSD) گنجانده شد.

برنامه های کاربردی سرویس گیرنده-سرور در سال 1990 با راه اندازی اینترنت و شبکه جهانی وب، که استفاده از سوکت را افزایش داد، به طور گسترده مورد استفاده قرار گرفت. یک طرف به عنوان سرور در برنامه های سرویس گیرنده-سرور، منتظر اتصالات مشتری است. این رویکرد به خدمات شبکه ای متعددی از جمله وب سرورها، سرورهای ایمیل و سرورهای پایگاه داده اجازه می دهد و به طور موثر بسیاری از مشتریان را مدیریت می کند.

سوکت ها با استفاده از چندین پروتکل، از جمله پروتکل Datagram کاربر (UDP) و پروتکل کنترل انتقال (TCP) بین کلاینت ها و سرورها ارتباط برقرار می کنند. به لطف پروتکل TCP قابل اعتماد و اتصال گرا، بسته های داده همیشه به ترتیب مناسب ارسال می شوند. این یک پیوند بین مشتری و سرور ایجاد می کند و دقت و قابلیت اطمینان انتقال داده را تضمین می کند. برعکس، UDP یک پروتکل سبک وزن و بدون اتصال است که سرعت را بالاتر از قابلیت اطمینان قرار می دهد. تأخیر کم برای برنامه های بلادرنگ، جایی که اغلب از آن استفاده می شود، بسیار مهم است.

هیچ زبان برنامه نویسی خاصی برای برنامه نویسی سوکت مورد نیاز نیست. این برنامه با چندین زبان از جمله پایتون، جاوا، سی پلاس پلاس و سی شارپ سازگار است و آن را برای توسعه دهندگانی که بر روی چندین پلتفرم کار می کنند در دسترس قرار می دهد.

برای فعال کردن فناوری های جدید، برنامه نویسی سوکت اخیراً پیشرفت کرده است. به عنوان مثال، استفاده از IPv6 اتصال و آدرس دهی را برای تطبیق تعداد فزاینده دستگاه های متصل به اینترنت بهبود بخشیده است.

**توابع سوکت API**

برنامه نویسان برنامه ممکن است سوکت ها را در برنامه های شبکه خود با استفاده از Socket API (Application Programming Interface) بسازند، پیکربندی و مدیریت کنند که مجموعه ای از توابع را ارائه می دهد. این فعالیت‌ها شامل ایجاد سوکت، اتصال به آدرس‌های خاص، ایجاد اتصالات، تبادل داده‌ها و بستن سوکت‌ها و غیره است. در زیر چند توابع پرکاربرد Socket API آورده شده است:

**socket():** یک سوکت جدید با استفاده از متد socket() ایجاد می شود که یک توصیفگر سوکت را ارائه می دهد که ممکن است برای اقدامات بعدی مورد استفاده قرار گیرد. آرگومان هایی را می پذیرد که نوع سوکت را نشان می دهد (مانند SOCK\_STREAM برای TCP یا SOCK\_DGRAM برای UDP) و خانواده آدرس (مانند AF\_INET برای IPv4 یا AF\_INET6 برای IPv6).

**bind():** با استفاده از متد ()bind، یک سوکت ممکن است به آدرس IP یا پورت خاصی در شبکه متصل شود. IP که سرور از طریق آن به اتصالات ورودی گوش می دهد معمولاً در سمت سرور مشخص می شود.

**listen():** یک سوکت می تواند اتصالات ورودی را با علامت گذاری به عنوان سوکت غیرفعال با استفاده از متد listen() بپذیرد. حداکثر ظرفیت سوکت برای اتصالات معلق مشخص شده است.

**accept():** با استفاده از متد ()accept، یک سوکت شنود می‌تواند درخواست‌های اتصال ورودی را بپذیرد. برای اتصال پذیرفته شده، یک سوکت تازه تولید می کند و یک توصیفگر سوکت جدید را برمی گرداند. در سمت سرور، این تابع معمولاً استفاده می شود.

**connect():** این روش یک اتصال سوکت کلاینت را به یک سرور دور راه اندازی می کند. با استفاده از آدرس سرور دور یک اتصال ایجاد می کند.

**send() و recv():** این دو تابع-send() و recv()- برای ارسال و دریافت داده ها در یک سوکت استفاده می شوند. آنها به مشتری و سرور ابزار ارسال و دریافت پیام را می دهند. این عملیات انتقال داده به موقع و قابل اعتماد را برای سوکت های اتصال گرا مانند TCP فراهم می کند.

**sendto() و recvfrom():** برخلاف send() و recv()، sendto() و recvfrom() با سوکت های بدون بدون اتصال مانند UDP استفاده می شوند. آنها انتقال و دریافت داده ها را بدون برقراری ارتباط فعال می کنند.

**close():** با استفاده از متد close() یک سوکت بسته می شود و تمام منابع مرتبط آزاد می شوند. برای سوکت های متصل، اتصال را قطع می کند و هرگونه ارتباط بیشتر را متوقف می کند.

**getaddrinfo():** بر اساس نام میزبان، پورت و عوامل دیگر، این تابع برای جمع‌آوری لیستی از آدرس‌های شبکه مناسب استفاده می‌شود. این قابلیت انعطاف پذیری وضوح آدرس شبکه IPv4 و IPv6 را ارائه می دهد.

**setsockopt():** این روش اصلاح گزینه سوکت را فعال می کند. این امکان مدیریت دقیق چندین ویژگی سوکت، از جمله تنظیم سوکت برای استفاده مجدد از آدرس ها، تغییر اندازه بافر سوکت و فعال کردن گزینه های چندپخشی را فراهم می کند.

**getsockopt():** مقدار فعلی یک گزینه سوکت را با استفاده از متد getsockopt() دریافت می کند. این امکان درخواست اطلاعات در مورد پیکربندی‌ها یا وضعیت‌های فعلی گزینه‌های سوکت، مانند یافتن مقدار وقفه یا مقدار یک پرچم سوکت خاص را فراهم می‌کند.

**shutdown():** با کمک متد shutdown()، اتصال سوکت ممکن است به خوبی قطع شود. می تواند پایان ارسال یا دریافت اتصال یا هر دو را خاتمه دهد. قبل از بستن سوکت، بررسی می‌کند که تمام داده‌های باقی مانده منتقل شده باشند.

**select():** این تابع قبل از نظارت بر چندین سوکت برای فعالیت، انتظار می‌دهد تا حداقل یک سوکت برای خواندن، نوشتن یا شرایط غیرعادی آماده شود. این کمک می کند تا چندین سوکت را به طور موثر و همزمان مدیریت کنید.

**fcntl():** این تابع انواع عملیات را برای کار با توصیفگرهای فایل ارائه می دهد، مانند تنظیم پرچم آنها یا تغییر حالت غیر مسدود کردن آنها.

ایجاد سوکت

وقتی روی پیوندی که ما را به صفحه خاصی می آورد کلیک کردیم. مرورگر وب به روش های زیر عمل می کند.

**Client.py**

# یک شی سوکت ایجاد می کنیم تا اتصال برقرار شود.

Client\_socket = socket.socket(socket.AF\_INET، socket.SOCK\_STREAM)

# اکنون به وب سرور در پورت 80 متصل شوید

# - پورت http معمولی

Client\_socket.connect(( "www.javatpoint.com" , 80 ))

کد بالا در سمت مشتری کار می کند. هنگامی که مشتری سعی می کند با سرور ارتباط برقرار کند، یک پورت Ephermal توسط سیستم عامل برای اتصال اختصاص داده می شود. پورت Epherimal چیزی نیست جز یک پورت تصادفی که توسط سیستم عامل اختصاص داده می شود. سوکت مشتری به محض تکمیل تبادل داده بسته می شود. سوکت مشتری فقط برای یک بار استفاده می شود.

سرور یک سوکت سرور ایجاد می کند که کمی پیچیده از سوکت مشتری است. بیایید ببینیم در سمت سرور چه اتفاقی می افتد.

**Server.py**

Server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#سوکت را به هاست عمومی و پورت معروف متصل می کنیم

Server\_socket.bind((socket.gethostname()، 80 ))

# تبدیل به سوکت سرور می شود و برای اتصالات گوش می دهد

Server\_socket.listen( 5 )

**تفاوت بین کلاینت و سوکت سرور:**

**سوکت کلاینت:**

برای اتصال به سرور، برنامه های سرویس گیرنده از سوکت های مشتری استفاده می کنند.

آنها یک مکالمه را با ایجاد پرس و جو از سرور شروع می کنند. سپس آنها پاسخ می گیرند.

ارتباطات مشتری موقتی است و برای تعاملات یا فعالیت های خاص ایجاد شده است.

پورت های زودگذر، که سیستم عامل به صورت پویا اختصاص می دهد، اغلب توسط آنها برای ارتباط استفاده می شود.

داده های ارسال شده به سرور و دریافت شده باید توسط سوکت های مشتری پردازش شوند.

**سوکت سرور:**

برنامه های در حال اجرا بر روی سرورها از سوکت های سرور برای بررسی اتصالات مشتری جدید استفاده می کنند.

آنها منتظر اتصالات مشتریان می مانند و چندین اتصال مشتری را به طور همزمان مدیریت می کنند.

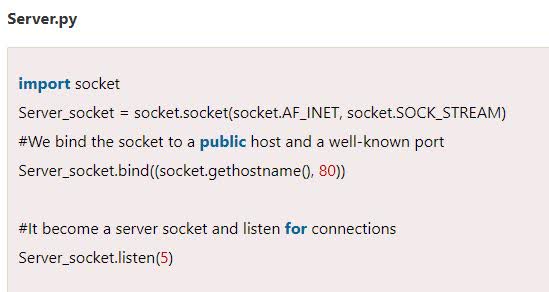
سوکت های سرور طول عمر بالایی دارند و در نظر گرفته شده که همیشه در دسترس باشند.

آنها با گوش دادن به پورت های مشخص شده مرتبط با سرویس ها یا پروتکل های خاص عمل می کنند.

سوکت های سرور وظیفه دریافت اتصالات مشتری ورودی و پاسخگویی با خدمات یا داده ها را بر عهده دارند.

آنها اغلب می توانند چندین اتصال کلاینت را به طور همزمان با تنظیم رشته ها یا فرآیندهای مختلف برای هر اتصال مدیریت کنند.





**ماژول سوکت پایتون:**

ماژو ل سوکت پایتون روش های مورد نیاز برای تولید و استفاده از سوکت ها را ارائه می دهد. Socket.socket() تکنیک اصلی برای ایجاد سوکت است. سینتکس تابع socket() به صورت زیر است:

Create\_Socket = socket.socket(socket\_family, socket\_type, protocol=0)

آرگومان های متد socket() به شرح زیر است:

**socket\_family:** خانواده آدرس برای سوکت توسط socket\_family مشخص می شود. می تواند AF\_UNIX (برای سوکت های دامنه یونیکس) یا AF\_INET (در دامنه اینترنت) باشد.

**socket\_type:** نوع سوکت توسط socket\_type مشخص می شود. برای سوکت های TCP، می تواند SOCK\_STREAM و برای سوکت های UDP، SOCK\_DGRAM باشد.

پروتکل: پروتکل خاص مورد استفاده را مشخص می کند. معمولاً روی 0 تنظیم می شود و سیستم عامل را قادر می سازد تا پروتکل مناسب را به دنبال socket\_type ارائه شده انتخاب کند.

در بیشتر موارد، ما با سوکت‌های AF\_INET (برای دامنه اینترنت) و سوکت‌های SOCK\_STREAM (برای TCP) کار خواهیم کرد. پس از ایجاد سوکت، می‌توانیم از روش‌های مختلفی که ماژول سوکت برای تعامل با سوکت ارائه می‌کند، مانند bind()، connect()، listen()، send()، recv() و غیره استفاده کنیم.

این روش‌ها به ما اجازه می‌دهند سوکت را به یک آدرس و پورت خاص متصل کنیم، یک اتصال به یک سرور راه دور برقرار کنیم، به اتصالات ورودی گوش دهیم، داده‌ها را ارسال کنیم، داده‌ها را دریافت کنیم و سایر عملیات مربوط به سوکت را انجام دهیم.

در مدل سوکت سرویس گیرنده، سوکت مشتری به سرور متصل می شود در حالی که سوکت سرور منتظر اتصالات ورودی است. در حالی که سوکت مشتری از متد connect() برای ایجاد یک اتصال استفاده می کند، سوکت سرور سوکت را با استفاده از متدهای bind() و listen() تنظیم می کند.

هنگام ایجاد برنامه های شبکه ای که از ارتباطات سوکت استفاده می کنند، درک و استفاده از عملکردهای ماژول سوکت پایتون ضروری است.

**روش های سوکت های کلاینت**

اتصال ()

این عملکرد برای راه اندازی اتصال به یک سوکت راه دور در یک آدرس استفاده می شود. یک قالب آدرس شامل جفت میزبان و پورت است که برای خانواده آدرس AF\_INET استفاده می شود.

**روش های سوکت سرور**

بستن()

این روش برای اتصال سوکت به یک آدرس استفاده می شود. قالب آدرس بستگی به خانواده سوکت ذکر شده در بالا (AF\_INET) دارد.

قبول می کند()

متد accepts() یک اتصال را می پذیرد. سوکت باید به یک آدرس متصل شده و آماده گوش دادن به اتصالات باشد. جفت (conn, address) را برمی گرداند که در آن con یک شی سوکت جدید است که می تواند برای ارسال و دریافت داده در اتصال استفاده شود و آدرس آدرسی است که به سوکت در انتهای دیگر اتصالات پیوند داده شده است.

**چند روش رایج سوکت**

چند توابع رایج مورد استفاده برای شی سرور در زیر آورده شده است.

Server\_Object = socket.socket(socket\_family، socket\_type، protocol = 0 )

روش های سوکت TCP روش‌های سوکت UDP

Server\_Object.recv() - پیام های TCP را دریافت می کند Server\_Object.recvfrom() - پیام های UDP را دریافت کنید.

Server\_Object.send( ) - پیام های TCP را انتقال دهید Server\_Object.sendto() - پیام های UDP را ارسال می کند.

انواع سوکت

در شبکه، دو نوع مختلف سوکت وجود دارد: SOCK\_STREAM و SOCK\_DGRAM. بیایید هر نوع را با جزئیات بیشتری بررسی کنیم:

**SOCK\_STREAM**:

ارتباطات مبتنی بر TCP (پروتکل کنترل انتقال) از سوکت های SOCK\_STREAM استفاده می کند.

آنها جریان قابل اعتمادی از داده را ارائه می دهند که بر اتصال متمرکز است.

داده ها از طریق SOCK\_STREAM در یک جریان پیوسته منتقل می شود و تضمین می کند که به همان ترتیبی که ارسال شده است می رسد.

این سوکت ها تحویل داده ها را تضمین می کنند و به طور خودکار هرگونه ارسال مجدد یا بسته های از دست رفته را مدیریت می کنند.

برنامه هایی که به تحویل داده های قابل اعتماد و سازماندهی شده نیاز دارند، مانند وب گردی، انتقال فایل، ایمیل و پروتکل های ارتباطی بلادرنگ مانند HTTP، FTP و SSH، اغلب از سوکت های SOCK\_STREAM استفاده می کنند.

داده ها ممکن است در هر دو جهت تحویل و دریافت شوند زیرا ارتباط دو طرفه است.

**SOCK\_DGRAM**:

برای ارتباطات مبتنی بر UDP (پروتکل داده‌های کاربر)، سوکت‌های SOCK\_DGRAM استفاده می‌شوند.

آنها یک سرویس دیتاگرام ناپایدار و بدون اتصال را ارائه می دهند.

داده ها از طریق SOCK\_DGRAM در بسته های جداگانه ای به نام دیتاگرام منتقل می شوند که هر کدام یک واحد ارتباطی مجزا هستند.

دیتاگرام ها ممکن است در حین انتقال از بین بروند یا تکرار شوند و هیچ اطمینانی وجود ندارد که به ترتیب ارسال شده دریافت شوند.

برنامه‌هایی مانند پخش همزمان چند رسانه‌ای، بازی‌های آنلاین، DNS و SNMP که می‌توانند از دست دادن بسته‌های عجیب و غریب یا تحویل خارج از سفارش را تحمل کنند، اغلب از سوکت‌های SOCK\_DGRAM استفاده می‌کنند.

داده ها ممکن است فقط در یک جهت در طول ارتباط یک طرفه ارسال شوند.

**سوکت های TCP**

در این بخش، شی سوکت را با استفاده از تابع ()socket.socket ایجاد می کنیم و نوع سوکت را به عنوان socket.SOCK\_STREAM تعیین می کنیم . همانطور که می دانیم، یک پروتکل برای ارسال داده ها از یک طرف به سر دیگر ضروری است. در اینجا TCP (پروتکل کنترل انتقال) از پروتکل پیش فرض استفاده می شود. بسیار کاربردی و قابل اعتماد است. ما می توانیم این پروتکل را در نظر بگیریم زیرا -

قابل اعتماد است - اگر بسته در بین انتقال به انتهای دیگر از دست رفته باشد، توسط فرستنده دوباره ارسال می شود.

در تحویل سفارش - پیام به همان ترتیبی که توسط فرستنده نوشته شده ارسال می شود.

بیایید مثال زیر را از برنامه ساده Client-Server درک کنیم.

برنامه سرور ساده

مثال -

#!/usr/bin/python

#این اسکریپت tcp\_server.py است

# ماژول سوکت را وارد کنید

سوکت واردات

# یک شی سوکت ایجاد کنید

s = socket.socket()

# نام دستگاه فعلی را دریافت کنید

host = socket.gethostname()

# شماره پورت را برای اتصال دریافت کنید

پورت = 9999

# پیوند با آدرس

s.bind((میزبان، پورت))

# برای اتصالات گوش دهید

چاپ ( "در انتظار اتصال..." )

s.listen( 5 )

# اتصال و پذیرش از مشتری

در حالی که درست است:

conn,addr = s.accept()

چاپ ( "اتصال از" , adr)

conn.send ( "پاسخ از سرور" )

# اتصال را ببندید

اکنون یک برنامه client.py برای ارتباط با فایل server.py ایجاد می کنیم .

برنامه مشتری ساده

برنامه client.py زیر را در نظر بگیرید . کلاینت سعی می کند اتصال را به پورت سرور تنظیم کند. ما پورت 9999 را به خوبی تعریف می کنیم.

مثال -

#!/usr/bin/python

#این اسکریپت tcp\_client.py است

سوکت واردات

#سوکت بسازید

s = socket.socket()

# نام دستگاه فعلی را دریافت کنید

host = socket.gethostname()

# مشتری می خواهد به سرور متصل شود

شماره پورت 9999

پورت = 9999

شماره 1024 اندازه بافر یا حداکثر مقدار است

# داده به یکباره دریافت می شود

s.connect((میزبان، پورت))

چاپ (s.recv( 1024 ))

برای به دست آوردن نتیجه، ابتدا فایل server.py را اجرا کنید و سپس فایل client.py را اجرا کنید . اگر خطایی وجود نداشته باشد، خروجی زیر را می دهد.

توجه - در اینجا، فایل های سرویس گیرنده و سرور بر روی یک دستگاه در حال اجرا هستند، اما در زندگی واقعی سرور در مکان متفاوتی قرار دارد. نکته این است که توجه داشته باشید که در اینجا client.py خاتمه یافته است اما server.py هنوز در حال اجرا است. این اتفاق در سناریوی واقعی نیز رخ می دهد.

به عنوان مثال - هنگامی که شما درخواستی را به javatpoint.com می دهید، آن درخواست را انجام می دهد و سرور آن به طور مداوم در حال اجرا (24\*7) در پس زمینه است.

**ماژول اینترنت پایتون**

لیست زیر ماژول پایتون مربوط به برنامه نویسی شبکه است.

ن را ارسال می کند و بازتاب می دهد.

Echo Client

ما برنامه مشتری echo را توصیف می کنیم و آن را ذخیره می کنیم echo-client.py:

برنامه -

سوکت واردات

# این نشان دهنده نام میزبان یا آدرس IP سرور است

HOST = '127.0.0.1'

# این شماره پورتی است که توسط سرور استفاده می شود

PORT = 65432

با socket.socket(socket.AF\_INET، socket.SOCK\_STREAM) به عنوان s:

s.connect((HOST، PORT))

s.sendall (b "سلام، دنیا" )

داده = s.recv( 1024 )

چاپ ( "دریافت" ، repr(داده))

**سوکت های مسدود کننده و غیر مسدود کننده**

در بخش های قبلی، یاد گرفتیم که مشتری درخواست را به سرور ارسال می کند و سرور درخواست را پردازش می کند. با استفاده از سوکت ها (TCP/UDP) یک پاسخ ارسال کنید.

**مسدود کردن سوکت I/O**

سوکت های TCP اغلب به گونه ای پیکربندی می شوند که هنگام برخورد با آنها به طور پیش فرض در حالت مسدود کردن اجرا شوند. در حالت مسدود کردن، اجرای برنامه تا پایان یک عملیات معین به حالت تعلیق در می آید. به عنوان مثال، هنگام اجرای تابع connect() برای اتصال به سرور، برنامه متوقف می شود و منتظر می ماند تا اتصال ایجاد شود.

در برخی شرایط، ممکن است سیستمی را اضافه کنیم تا اتصالات فعال را در حین استفاده از آنها مدیریت کند یا قطع کند. حالت غیر مسدود در این شرایط مفید است. می‌توانیم با تنظیم یک سوکت در حالت غیر مسدود کردن، اجازه دهیم برنامه در حالی که منتظر پایان فعالیت‌های خاص است، به اجرا ادامه دهد. این به عملیات سوکت عرض جغرافیایی و کنترل بیشتری می دهد.

**سوکت I/O غیر مسدود کننده**

برای جابه‌جایی بین حالت‌های مسدود و غیر مسدود برای سوکت در پایتون، از تابع setblocking() استفاده کنید. ممکن است منطق اضافی برای رسیدگی به اختلالات یا انجام سایر فعالیت‌ها در حالی که منتظر تکمیل عملیات سوکت هستیم با تغییر سوکت به حالت غیر مسدود کننده اضافه کنیم.

بیایید مثال زیر را در مورد Blocking Socket درک کنیم .

فایل blockclient.py را ذخیره کنید

برنامه

سوکت واردات

# ایجاد یک شی سوکت

socket\_obj = socket.socket()

میزبان = socket\_obj.gethostname()

socket\_obj.connect((میزبان، 12345 ))

socket\_obj.setblocking( 1 )

# Or simply omit this line as by default TCP sockets

# در حالت مسدود کردن هستند

# حجم عظیمی از داده ها برای ارسال

data = "Hello Python\n" \* 10 \*1024\*1024

# ارسال داده تا درست است

عنوان socket\_obj.send(داده)

اکنون، blockserver.py زیر را درک کنید:

برنامه

سوکت واردات

# ایجاد یک شی سوکت

socket\_obj = socket.socket()

# تعریف نام میزبان

host = socket.gethostname()

# تعریف شماره پورت

پورت = 12345

socket\_obj.bind((میزبان، پورت))

socket\_obj.listen( 5 )

در حالی که درست است:

# اتصال را بپذیرید

conn، addr = socket\_obj.accept()

داده = conn.recv( 1024 )

# تا زمانی که داده در راه است

در حالی که داده ها:

چاپ (داده)

داده = conn.recv( 1024 )

# زمانی که همه داده ها دریافت شوند اجرا می شود

چاپ ( "همه داده های دریافت شده" )

conn.close()

**کار با سوکت های UDP**

ما آموخته ایم که اگر خانواده سوکت و نوع سوکت را ذکر نکنیم، به طور پیش فرض TCP است . برای ایجاد سوکت UDP، باید خانواده سوکت و نوع سوکت را به صراحت مشخص کنیم. بیایید نحو زیر را درک کنیم.

s = socket.socket(socket.AF\_INET، socket.SOCK\_DGRAM)

بیایید برنامه سرور UDP زیر را درک کنیم.

برنامه سرور UDP

اسکریپت زیر را بنویسید و با نام server.py ذخیره کنید

مثال -

سوکت واردات

# برای پروتکل UDP استفاده می شود

udp\_socket = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)

# IP میزبان

udp\_host = socket.gethostname()

# ما در حال تعیین پورت برای اتصال هستیم

udp\_port = 12345

#print type(udp\_sock) 'type' می تواند برای دیدن نوع استفاده شود

# از هر متغیر ( "جوراب" در اینجا)

udp\_socket.bind((udp\_host,udp\_port))

در حالی که درست است:

چاپ ( "در انتظار مشتری..." )

data,addr = sock.recvfrom( 1024 ) #receive data from client

print( "پیام های دریافتی:" ,داده ها, " از" ,addr)

خروجی:

در انتظار مشتری

برنامه مشتری UDP

این فایل را با نام udpclient.py ذخیره کنید

**کلاینت و سرور اکو**

یک کلاینت می تواند با استفاده از یک سرور اکو، یک برنامه کاربردی شبکه ساده، پیام یا درخواستی را به سرور ارسال کند و سرور با ارسال پیام یا درخواست مشابه به مشتری پاسخ خواهد داد. یک پارادایم اساسی ارتباط مشتری-سرور نشان داده شده است.

سرور اکو به روش های زیر کار می کند:

سرور یک پورت شبکه خاص را برای اتصالات مشتری ورودی نظارت می کند.

هنگامی که یک کلاینت به سرور متصل می شود یک سوکت برای ارتباط بین مشتری و سرور تشکیل می شود.

مشتری از طریق سوکت با سرور ارتباط برقرار می کند یا چیزی از آن درخواست می کند.

پیام از مشتری به سرور تحویل داده می شود.

سرور پیام بافر دوباره خالی شود. هنگامی که بافر خالی شد، هسته فرآیند خود را دوباره شروع می کند تا تکه بعدی داده ای که قرار است منتقل شود را دریافت کند.

اکنون یک سوکت غیر مسدود کننده را در نظر بگیرید.

Program

# non\_blocking\_client.py

سوکت واردات

socket\_object = socket.socket()

host = socket.gethostname()

socket\_object.connect((میزبان، 12345 ))

# اکنون روی حالت غیر مسدود کننده تنظیم کنید

socket\_object.setblocking( 0 )

# حجم عظیمی از داده ها برای ارسال

داده = "Hello Python\n" \* 10 \* 1024 \* 1024

# ارسال داده تا درست است

assert socket\_object.send(داده)

وقتی non\_block\_client.py را اجرا می کنیم ، برنامه برای مدت کمی اجرا می شود، "همه داده های دریافت شده" را چاپ می کند و به سرعت تمام می شود.

اگر سوکت غیرمسدود کننده را با setblocking(0) ایجاد کنیم، هرگز منتظر تکمیل عملیات نخواهد بود. بنابراین وقتی تابع send() را فراخوانی می کنیم، سعی می کند حداکثر مقدار داده را در بافر قرار دهد.

**بستن یک اتصال**

استفاده از تابع ()close در سوکت معمولاً هنگام پایان دادن به اتصال توصیه می شود. استفاده از ()close معمولاً کافی است و بستن آن نیازی به فراخوانی صریح تابع shutdown() ندارد .

با این حال، در برخی شرایط - به ویژه زمانی که با یک تبادل HTTP مانند سروکار داریم، استفاده از روش shutdown() ممکن است مفید باشد. به عنوان مثال ، اگر مشتری از shutdown (1) استفاده کند، می تواند به دریافت داده ادامه دهد ، که به این معنی است که تحویل داده را کامل کرده است. وضعیت "EOF" (پایان فایل) زمانی که سرور 0 بایت دریافت می کند، می تواند توسط سرور شناسایی شود، به این معنی که مشتری درخواست خود را به پایان رسانده است. سپس، در صورت نیاز، سرور می تواند پاسخی را ارائه دهد. در صورت موفقیت عملیات انتقال، توانایی مستمر مشتری برای دریافت تأیید می شود.

فتی را پردازش می کند (که در مورد سرور اکو ممکن است اصلاً نیاز به پردازش نداشته باشد).

از طریق سوکت، سرور همان پیام یا پاسخ را به مشتری ارسال می کند.

مشتری پیام یا پاسخ سرور را دریافت می کند.

سپس مشتری می تواند پیام های بیشتری ارسال کند یا ارتباط خود را با سرور قطع کند.

از آنجایی که به مشتری امکان می دهد تأیید کند که سرور با موفقیت پیام ها را دریافت و پردازش می کند، سرور اکو اغلب برای آزمایش و اشکال زدایی استفاده می شود.

**کار با دستگاه های ورودی و خروجی**

جریان های ورودی و خروجی برای مدیریت جریان داده بین سرور اکو و مشتری برای برقراری ارتباط مورد نیاز است. این جریان‌ها ارتباط بین مشتری و سرور را امکان‌پذیر می‌کنند و کلاینت پژواک پیام را از سرور دریافت می‌کند.

جریان ورودی نشان دهنده جریان داده از مشتری به سرور است. سرور از آن برای دریافت پیامی که مشتری ارسال کرده است استفاده می کند. سرور داده ها را پس از خواندن از جریان ورودی پردازش می کند.

جریان خروجی نشان دهنده جریان داده از سرور به مشتری است. سرور پس از دریافت پیام از مشتری، همان پیام یا پاسخی به جریان خروجی منتشر می کند. سپس مشتری می تواند پیام بازتاب داده شده را بخواند یا از جریان خروجی پاسخ دهد.

این جریان‌های ورودی و خروجی به مشتری و سرور اجازه می‌دهند در هر دو جهت ارتباط برقرار کرده و داده‌ها را مبادله کنند. سرور ممکن است چندین پیام مشتری را دریافت کند، که ممکن است هر کدام را جداگانه تکرار کند.

بیایید مثال زیر را درک کنیم:

سرور اکو

سوکت واردات

# آدرس رابط Loopback استاندارد (localhost)

host\_name = '127.0.0.1'

# پورت مشخص شده برای گوش دادن (پورت های غیرمجاز > 1023 هستند )

port\_name = 65432

# سوکت TCP

با socket.socket(socket.AF\_INET، socket.SOCK\_STREAM) به عنوان s:

s.bind((نام\_میزبان، نام\_پورت))

s.listen()

conn, addr = s.accept()

با اتصال:

print( 'Connected by' , addr)

در حالی که درست است:

داده = conn.recv( 1024 )

تابع socket.socket() یک شی سوکت ایجاد می کند که از نوع مدیریت متن پشتیبانی می کند، همچنین می توانیم از آن در دستور with استفاده کنیم و نیازی به فراخوانی تابع ()close نیست .

اکنون یک شی سوکت از تابع ()accept داریم. یادآوری این نکته مهم است که سوکتی که برای برقراری ارتباط از آن استفاده خواهیم کرد باید از سوکت شنیداری که سرور برای پذیرش اتصالات جدید از آن استفاده می کند متمایز باشد.

با اتصال:

print( 'Connected by' , addr)

در حالی که درست است:

داده = conn.recv( 1024 )

اگر داده نیست:

زنگ تفريح

conn.sendall (داده)

هنگامی که فرآیند دریافت شیء conn را از تابع ()ccept انجام دادیم . ما از حلقه while بی‌نهایت روی مسدود کردن تماس‌های conn.recv() استفاده کردیم . کلاینت با استفاده از () conn.sendall آ با این وجود، استفاده از خاموش شدن و بسته شدن خودکار مورد تردید است. اگر سوکتی پیش از موعد بسته شود، می‌تواند منجر به مشکلاتی مانند ارسال درخواست‌ها توسط سوکت دیگر شود. بنابراین، بستن سوکت زمانی که فعالیت های مورد نظر به صراحت تمام شد، بسیار مهم است.

شما می‌توانید با قطع صریح سوکت با استفاده از () close، تضمین کنید که تمام فرآیندهای پاکسازی و خاتمه ضروری انجام می‌شوند، و از هرگونه مشکل احتمالی اتصال اجتناب می‌کنید.

**WebSockets** یک فناوری محبوب برای ارتباط بلادرنگ بین مشتریان و سرورها. Tornado به دلیل ماهیت ناهمزمانش برای حفظ اتصالات WebSocket دائمی کاملاً مناسب است و برنامه‌های بلادرنگ کارآمد و سریع را ممکن می‌سازد.

کارایی بالا: تورنادو به لطف طراحی و معماری آن یک چارچوب با کارایی بالا است. تورنادو می‌تواند زمان پاسخگویی سریع‌تر و عملکرد تراکنش‌های بالاتر را با کاهش سربار مربوط به برنامه‌نویسی سوکت استاندارد فعال کند.

**برنامه نویسی سوکت با استفاده از پایتون**

در کد بالا، ما کلاس ApplicationHandler را تعریف کرده ایم که به عنوان کنترل کننده درخواست استفاده می کند و با استفاده از write() پاسخی را برمی گرداند.

متد main() ورودی برنامه است.

کلاس web.Application پایه برنامه وب را ایجاد می کند و مجموعه ای از کنترل کننده ها را می پذیرد.

برنامه به پورت 5000 گوش می دهد و مشتری می تواند با استفاده از همان پورت با این برنامه ارتباط برقرار کند.

تابع ioloop.IOLoop.instance().start() برای ایجاد یک رشته غیر مسدود کننده برای یک برنامه استفاده می شود.

در این مقاله به مفاهیم اولیه برنامه نویسی شبکه با استفاده از پایتون پرداخته ایم. ما اصطلاحات اصلی شبکه را تعریف کرده ایم و سرور و کلاینت ساده را ایجاد می کنیم. برنامه نویسی شبکه یک زمینه گسترده است و پوشش آن در یک آموزش سخت است، اما سعی شده است تمام مفاهیم مهم در مورد برنامه نویسی شبکه با استفاده از پایتون پوشش داده شود.