**به نام خدا**

حسین محصل ارجمندی گزارش پروژه شبکه

هر client، یک برنامه متشکل از چند کلاس است. کلاس اصلی، کلاس Node است که شامل اطلاعات اصلی از جمله اسم گره، پورت udp است که داخل constructor کلاس قرار می گیرد.

یک سری متغیر استاتیک هم داریم که مشخصه هر گره است اما داخل constructor نیست. برا سادگی بیشتر برنامه و پیچیده نشدن پیاده سازی، تصمیم گرفتم که خارج از سازنده پیاده شود و به صورت دستی مقدار دهی شود. عبارت است از requestWaitPeriod که مدت زمانی است که یک درخواست کننده منتظر پاسخ از بقیه گره ها می ماند که آیا فایل درخواستی موجود است یا خیر؟

مورد بعدی discoveryIntervalMillisec است که بازه های ارسال پیام اکتشاف است. مورد بعدی nestPath است که مسیر دایرکتوری ای است که برنامه قرار است روی آن عمل کند. مورد بعدی، servingLimit است که تعداد گره هایی را مشخص می کند که هر نود حداکثر به صورت همزان به آن ها سرویس می دهد.

مورد نهایی هم trickTimeMillisec است که می گوید برای piggybacking چند میلی ثانیه صبر کنیم؟

داخل سازنده کلاس Node، از طریق تابع createRandomTcpPort یک پورت نامعلوم و آزاد برای ارتباط TCP انتخاب می شود. هر گروه، کار اصلی خود را در تابع loop انجام می دهد. در تابع loop، هر گره، اولا یک discoverysender میسازد و یک udpBroadcast میسازد و هردو را استارد می کند (بعدا دقیق تر توضیح خواهم داد) و سپس به صورت نامتناهی درخواست کاربر را به صورت string ورودی دریافت می کند. این درخواست ها یا به صورت list هستند یا به صورت GETFILENAME. توجه شود که حروف GET باید بزرگ باشند و اسم فایل هم نباید از GET فاصله داشته باشد. این دو درخواست طبق صورت پروژه عمل می کنند.

هر گره برنامه یک DiscoverySender دارد که در بازه های مشخص شده، پیام اکتشاف را برای اعضای نود خود می فرستد. از آنجا که ممکن است همه نود ها روی یک کامپیوتر اجرا شوند، موقع ارسال دیسکاوری، پورت های udp هم تبادل می شوند. توجه شود که تابع sendUDPSignal یک تابع استایک در کلاس Node است که یک رشته را به صورت UDP، به یک host با آدرس ip مشخص و پورت udp مشخص می فرستد.

هر گره یک UDPBroadcast هم دارد که توسط آن بسته های دریافت می شود و بسته ها بررسی می شود. هر بسته دریافتی UDP، سه حالت دارد: یا پیام اکتشاف است که اولش DIS دارد، یا درخواست فایل است که اولش GET دارد و یا پاسخ به موجود بودن فایل و در نتیجه پورت tcp فرستنده است، که اول این درخواست عبارت TCP وجود دارد. با توجه به این سرآیند ها، رشته دریافتی را با تابع multiplexUDPPacket پردازش می کنیم و اقدام لازم را انجام میدهیم:

* اگر پیام DIS بود، مجموعه cluster که مجل نگهداری اعضای خوشه از دید خودمان هست را آپدیت می کنیم و اگر نود جدید قبلا در لیست ما موجود نبود، اضافه می کنیم.
* اگر پیام GET بود، بررسی می کنیم که آیا فایل درخواستی را در دایرکتوری برنامه خود داریم یا نه؟ و اگر داریم، آیا الان دریافت فایل کامل شده؟ (یا خودمان فعلا در حال دریافت فایل با آن اسم هستیم؟ ) اگر جواب هر دو سوال بله بود، باید جواب بدهیم. اما طبق فرض سوال، یک چیزی چک می کنیم. آیا درخواست دهنده، قبلا به ما فایل فرستاده؟ یعنی اسمش در وکتور nodesALreadyGotFileFrom هست یا نه؟ اگر بود که بلافاصله جواب میدهیم، اما اگر نبود، تابع trick را اجرا می کنیم که در اینجا trickTimeMillisec میلی ثانیه دیرتر جواب می دهد.
* اگر پیام دریافتی سرآیند TCP داشت، اگر از آخرین بار ارسال درخواست GET، هنوز مدت زمان requestWaitPeriod از زمان ارسال درخواست نگذشته بود و اگر فایلی که فرستنده پیام TCP میخواد به ما بفرستد، اسمش اسم همان آخرین فایلی بود که ما می خواستیم، آن فرستنده را در responders ثبت می کنیم. این کار را برای این کردیم که مثلا ممکن است یک پاسخ دهنده خیلی دیر پاسخ دهد، وقتی که زمان requestWaitPeriod تمام شده و ما درخواست جدید فرستادیم، ممکن است پاسخ دهنده ای تازه نسبت به درخواست قبلی ما پاسخ دهد که این ممکن است باعث اشتباه شود. در ضمن، در این برنامه فرض کردیم که ما وقتی یک درخواست GET می دهیم، یا مدت زمان requestWaitPeriod حق نداریم درخواست GET جدید بفرستیم.

هر برنامه، علاوه بر UDPBroadcast، یک TCPBroadcast هم دارد که دائما روی پورت TCP خود گوش می دهد. وظیفه این کلاس این است که وقتی یک کانکشن TCP با آن از طریق یک کلاینت برقرار شد، کلاینت (که قبلا با توجه به تاخیر minimum این سرور را انتخاب کرده) اسم فایلی که کاربر درخواست کرده را به سرور می گوید و سرور فایل را برای او می فرستد. هر نود در اینجا به طور همزمان به حداکثر servingLimit تا کلاینت خدمت می دهد و در این شرایط اگر این حد اشباع شده باشد، سرور درخواست اتصال جدید را قبول نمی کند. یعنی به کلاینت اطلاع می رساند که شلوغ است و بعدا اقدام کن. وقتی که TCPBroadcast یک درخواست دریاف فایل دریافت می کند و می خواهد فایل را بفرست، یک TCPSender ایجاد می کند با سوکت accept شده که وظیفه ارسال فایل درخواستی را بر عهده دارد، تا TCPBroadcast بتواند به بقیه نود ها سرویس بدهد.

کلاس دیگر، TCPReceiver است. وظیفه این کلاس این است که وقتی کاربر درخواست یک فایل می کند، وقتی دوره requestWaitPeriod تمام شد، بین پاسخ دهندگان واجد شرایط، کسی که حداقل تاخیر را داشت انتخاب می کند و یک ارتباط TCP با او برقرار می کند و اسم فایل را به او می گوید و فایل را دریافت می کند. در ضمن، سرور را داخل nodesALreadyGotFileFrom اضافه می کند تا دیگر بعدا نخواهد پاسخ او را دیر تر بدهد.

در این برنامه به دو موضوع توجه شده:

اولا داده ساختار های به کار برده از نوع Vector هستند که Thread safe هستند و موقع تبادلات داده با شبکه، مسئله رقابت پیش نمی آید.

دوما، تمامی برقراری ارتباطات به صورت Thread پیاده شده اند تا برنامه ها بتوانند به صورت همزمان چندین درخواست را مدیریت کنند.

**نهایتا موقع تست برنامه به 6 نکته توجه شود:**

* اولا کلاس نود دوتا constructor دارد. یکی با وکتور cluster اولیه و دومی بدون وکتور کلاستر اولیه. بنابرین در بدو امر، اگر یک نود توی خوشه اش نود دیگری هست، باید در تابع main یک وکتور ساخته بشه و اون گره ها به اون وکتور اضافه بشه، و گره ای که میخوایم بسازیم رو باید با وکتور بسازیم. در مورد گره هایی که در اول خوشه شون خالیه، میشه بدون وکتور هم ساخت و فقط پورت UDP و ip مربوط به این گره رو داد. در ضمن توجه شود که ورودی های وکتور کلاستر اولیه هم یک سری نود هستن.
* نکته دوم این است که در هر برنامه، اون نود اصلی ای که می خوایم بسازیم (نه نود هایی که صرفا میخوایم داخل وکتور بزاریم) بعد از ساخته شدن، باید تابع loop() فراخوانی بشه تا برنامه شروع به اجرا بشه.
* در ثالثا، ممکنه پورت های UDP تخصیص داده شده روی سیستم شما پر باشه و برنامه binding exception بخوره. این ارور با عوض کردن پورت ها چه در برنامه اصلی مربوط به گره و چه در وکتور هایی که این گره در آن ها هست، مشکل رو حل می کنه
* رابعا، موقعی که برنامه رو میخواین تست کنین، طبیعتا چند پنجره مختلف از برنامه من باز می کنین. حواستون باشه که اون nestPath رو اصلاح کنین برای هر پنجره. در واقع اون nestPath همون مسیر دایرکتوری ای است که اون برنامه قراره روش کار کنه.
* خامسا، تابع main هم اصلاح کنین با توجه به نود هایی که میخواین بسازین. یعنی نود هایی که کلاستر اولیه شون خالیه رو با constructor بدون کلاستر بسازین و loop() بزنین روشون، نود هایی که کلاستر اولیه شون خالی نیست رو اول کلاسترشونو توی main بسازین و بعد با اون کلاستر نود رو بسازین وloop() بزنین روش.
* نهایتا، فراموش نکنید که متغییر هایی که میخواین تنظیم کنین رو قسمت ابتدای کلاس Node تنظیم کنین. مثلا requestWaitPeriod یا servingLimit.

آدرس ریپازیتوری پروژه من در گیتهاب:

<https://github.com/hm0ss/CN>