پروژه پیام رسان تحت لینوکس

1. ساختار کلی پروژه

در این پروژه به طور کلی باید یک بستر پیام رسان پیاده شود، در نتیجه نیاز به یک برنامه اجرایی برای سرور و یک برنامه اجرایی دیگر برای کلاینت ها داریم. از لحاظ فایل های منبع کد، یک فایل به نام "server.c" شامل تابع اصلی برنامه (main) و تمامی منابع و توابع مستقیم خود است. برای برنامه کلاینت هم به همین شکل فایل منبع "client.c" تعریف شده و دارای تمامی منابع و توابع مستقیم خود است. علاوه بر این دو فایل، یک فایل دیگر به نام "messageProtocol.h" داریم که شامل تعدادی از مقادیر ثابت است و این مقادیر در هر دو برنامه سرور و کلاینت استفاده می شوند. در نتیجه این فایل باید در هر دو منبع کد ذکر شده در قبل مشمول شود.

2. نحوه اجرای برنامه

همانطور که گفتیم در این پروژه دو برنامه اجرایی داریم. همواره برای راه اندازی چنین سیستم هایی، ابتدا باید سرور راه اندازی شود. در نتیجه فایل اجرایی سرور را باز میکنیم، اینجا از ما پورت مورد نظر برای گوش دادن و متصل شدن به کلاینت ها خواسته می شود. اگر مقدار وارد شده نامعتبر باشد و یا سیستم قادر به در اختیار گرفتن پورت نباشد، برنامه بلافاصله متوقف می شود و باید دوباره اجرا شود.

بعد از راه اندازی سرور، میتوانیم برنامه اجرایی کلاینت را به تعداد دلخواه اجرا کنیم. این برنامه در ابتدا از ما آدرس آی پی سرور و پورت مورد نظر را درخواست میکند. در صورت معتبر نبودن مقادیر وارد شده و یا عدم توانایی ارتباط شبکه با آدرس وارد شده، برنامه متوقف می شود و باید دوباره اجرا شود.

بعد از راه اندازی هر کلاینت، کنترل برنامه آماده دریافت فرمان از کاربر می شود و طبق دستورات از قبل تعیین شده عمل می کند.

دستوراتی که در ترمینال برای اجرای برنامه وارد می شوند به این ترتیب خواهند بود:

- cd to directory
- make
- ./server
- ./client

3. نحوه پیاده سازی کد

ابتدا به پیاده سازی کد سرور میپردازیم و بعد از آن پیاده سازی کد کلاینت را بررسی می کنیم.

3.1. برنامه سرور

در برنامه سرور ابتدا تعامل با کاربر برای گرفتن پورت انجام میشود، سپس با کمک توابع سیستمی لینوکس در زبان c، یک سوکت پذیرنده راه اندازی میکنیم. لازم به ذکر است که این سوکت میزبان فقط برای پذیرش اتصالات به کار میرود، و با برقراری هر اتصال، پارامتر دیگری در اختیار ما قرار میگیرد که معرف همان اتصال خواهد بود. به عبارت دیگر برای هر اتصال برقرار شده هم یک سوکت مجزا در اختیار خواهیم داشت و به کمک آن میتوانیم عمل خواندن و نوشتن روی آن کلاینت خاص انجام دهیم.

برای برقراری امنیت و مقیاس پذیری برنامه، صرفا تعداد محدودی جایگاه برای دریافت اتصال از کلاینت در نظر گرفته ایم. یعنی مثلا اگر 10 کلاینت به سرور متصل شوند، برنامه سرور دیگر اتصال جدیدی نمیپذیرد، تا موقعی که یکی از اتصالات فعال قطع شوند و جایگاهی آزاد شود. به کمک متغیرها و آرایه های مختلف، همواره اطلاعات مربوط به وضعیت اتصالات فعال و غیر فعال را در اختیار داریم.

در ادامه یک نخ پردازشی به نام socketListenerThread تعریف می کنیم و با شروع آن، تابع handleConnectRequests به صورت موازی شروع به کار می کند. در این تابع در یک حلقه ی بینهایت ظرفیت برای تخصیص اتصالات بیشتر بررسی می شود و اگر جایگاهی موجود باشد، اقدام به ارتباط با کلاینت جدید میکنیم. در اینجا ممکن است که هیچ درخواستی برای اتصال موجود نباشد، پس باید دقت داشت تا سوکت پذیرنده سرور ما در حالت non-blocking قرار گرفته باشد. در غیر اینصورت برنامه متوقف میشود تا درخواست ارتباط جدیدی بیاید و اگر سوکت های فعال دیگری داشته باشیم، عملا آنها را رها کرده ایم. قرار دادن سوکت به حالت non-blocking از طریق تابع استاندارد fcntl انجام می شود. این تابع، مشخصات فایل مربوط به دستگیره سوکت را تنظیم میکند. در طول برنامه تمامی سوکت ها را در این حالت تنظیم میکنیم، زیرا این امر حتی در هنگام خواندن از سوکت از توقف برنامه جلوگیری می کند.

بعد از تعریف سوکت جدید و انجام مقداردهی های مربوط به آن، یک نخ پردازشی دیگر به نام clientHandlerThread تعریف می کنیم که با شروع آن، تابع handleClient با آرگومان اندیس مربوط به

سوکت مورد نظر ما، به صورت موازی شروع به کار می کند. باید توجه داشت که این نخ پردازشی به صورت کاملا جداگانه و به تعداد کلاینت های متصل شده به سرور ایجاد می شود، به همین دلیل آرگومان اندیس سوکت مربوطه برای آن ارسال می شود.

در تابع handleClient در یک حلقه بینهایت، 3 عملیات همواره انجام می شود.

در مرحله اول، در سوکت مربوط به نخ پردازشی خودمان بررسی میکنیم که آیا پیام جدیدی رسیده است یا خیر. اگر پیامی رسیده باشد تابع handleClientCommand فراخوانی می شود. در این تابع بررسی می شود که آیا پیام رسیده شده ساختار صحیحی دارد یا خیر. در صورت صحیح بودن دستور، با توجه به نوع آن اقدام مربوطه انجام می شود. (مثلا پیام همگانی و یا پرسش کاربران آنلاین).

در مرحله دوم بررسی می شود که آیا سوکت مربوط به نخ پردازشی خودمان، دارای پیام ارسال نشده است یا خیر. این حالت فقط وقتی اتفاق می افتد که کاربر آفلاین بوده و پیامی برایش ارسال شده بوده، و حالا لاگین انجام داده است. در این حالت پیام های در صف ارسال، به ترتیب به آن کاربر ارسال می شوند. لازم به ذکر است که تعداد پیام های ذخیره شده در این حالت محدود هستند.

در مرحله سوم برای سوکت مربوط به نخ پردازشی خودمان ، بررسی میکنیم که آیا پیام keep-alive در موعد مقرر ارسال شده است یا خیر. در صورت وقوع این حالت، یعنی کلاینت مربوطه آفلاین شده است. بنابراین وضعیت آن کاربر را به حالت آفلاین می بریم و سوکت را آزاد می کنیم، و این نخ پردازشی که در مورد آن صحبت می کردیم از حلقه ی بینهایت خارج می کنیم تا به اتمام برسد.

در نهایت در ادامه ی تابع اصلی برنامه، وارد یک حلقه ی بینهایت می شویم که در آن از کاربر ورودی دریافت می کند و پردازش می کند. فعلا فقط یک دستور تعریف شده داریم که به کمک آن می توان کاربر جدید برای سیستم تعریف کرد.

به این ترتیب برنامه ی سرور، در حلقه ی اصلی خود دستورات کاربر را پردازش می کند و به صورت موازی در یک نخ پردازشی، برای دریافت اتصالات کلاینت ها گوش می کند. در این نخ اگر اتصالی رویت شود، برای آن یک نخ پردازشی منحصر به فرد راه اندازی می کند تا عملیات خواندن و نوشتن مربوط به آن کلاینت به شکل موازی انجام پذیرد.

3.2. برنامه کلاینت

در برنامه کلاینت مثل برنامه سرور در ابتدا با کاربر تعاملی انجام می دهیم و مشخصات سرور را دریافت می کنیم. اما این بار صرفا یک سوکت داریم که به همتای خاص خود در سرور متصل می شود و به تنهایی عملیات

خواندن و نوشتن روی شبکه را انجام می دهد. همچنین این سوکت هم به دلایل ذکر شده در بخش قبلی، در حالت non-blocking قرار میدهیم.

در برنامه کلاینت، از دو نخ پردازشی دیگر استفاده میکنیم. در ادامه وظایف محوله به هرکدام را توضیح خواهیم داد.

بعد از ایجاد اتصال به سرور، بلافاصله عملیات لاگین انجام می شود. در این مرحله سوکت در کلاینت و سوکت همتای آن در سرور هر دو مقداردهی شده اند، اما در برنامه ی سرور هنوز کاربر آفلاین محسوب می شود و محدودیت زمانی برای ارسال اطلاعات کاربری برایش اعمال می شود و در برنامه ی کلاینت هم کاملا متوقف می شویم تا کاربر نام کاربری و رمز عبور را وارد کند. نام کاربری و رمز عبور به سرور ارسال می شوند تا عمل اعتبارسنجی انجام گیرد. در صورت صحیح بودن اطلاعات، کاربر در سمت سرور آنلاین شناخته می شود و در سمت کلاینت هم عملا وارد حلقه ی اصلی و بینهایت برنامه می شویم. همچنین نخ های پردازشی کلاینت در این مرحله شروع به کار می کنند. اگر در مهلت زمانی مربوط عمل لاگین انجام نشود، سرور سوکت مربوط به کلاینت را آزاد می کند و در سمت کلاینت هم در اولین تلاش برای ارسال دستور، برنامه متوقف می شود.

در حلقه ی اصلی برنامه ی کلاینت، صرفا یک عملیات انجام می شود. طی آن ورودی از کاربر دریافت می شود و بررسی می شود تا ساختار دستور طبق پروتکل تعریف شده صحیح باشد. اگر دستور صحیح باشد، یک دستور طبق پروتکل بین کلاینت و سرور (که خودمان طراحی کرده ایم) ساخته می شود و از طریق سوکت به سرور ارسال می شود.

در نخ پردازشی اول مربوط به کلاینت، همواره سوکت خوانده می شود و اگر داده ای آمده باشد، بررسی می شود و عملیات متناسب با آن انجام می شود. مثلا این پیام ممکن است شامل یک پیغام unicast به کاربر باشد، و یا پاسخی به دستور whoisonline باشد. واضح است که در مثال دوم، دستور اولیه ی whoisonline توسط خود کاربر داده شده است و پردازش و ارسال آن در حلقه ی اصلی برنامه به صورت موازی و جداگانه انجام گرفته است.

در نخ پردازشی دوم، صرفا در بازه های زمانی مشخص پیام keep-alive از روی سوکت به سرور ارسال می شود تا اتصال کلاینت از طرف سرور قطع نشود.

پس بدین ترتیب وظایف محوله ی کلاینت هم در یک حلقه ی اصلی بینهایت و دو نخ پردازشی شامل حلقه ی بینهایت مربوط به خود، انجام می گیرند.