گزارش پروژه اول سیستم‌های توزیع شده

علیرضا زارع‌نژاد 810196474

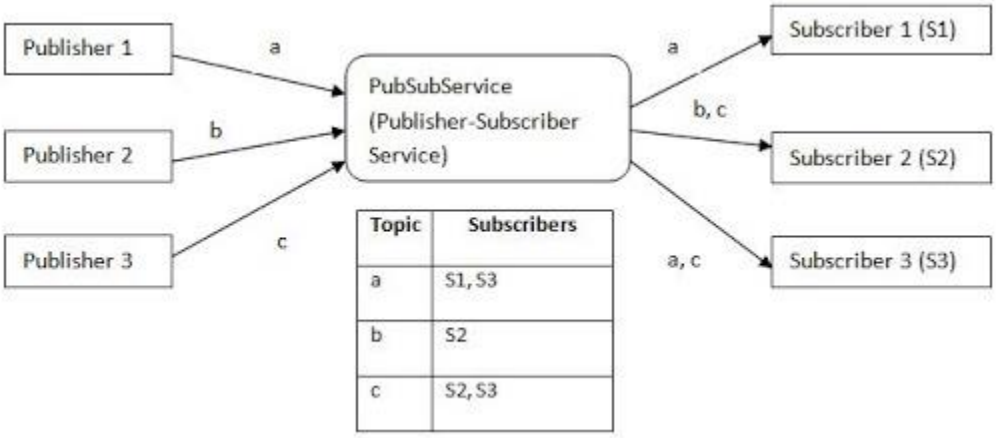
امیر حسین محمودی 810195578

جدول عناصر

مقدمه

در این پروژه به پیاده‌سازی یک سیستم توزیع پیام توسط یک broker پرداخته شده است. این سیستم از الگوری طراحی publisher/subscriber بهره می‌برد. به این منظور broker یک مجموعه صفوفی دارد که توسط master ها یا همان publisher ها ایجاد می‌شوند. سپس پیام‌هایی توسط این master ها روی صفوف قرار می‌گیرند. از آن طرف subscriber ها یا همان client ها این پیام‌ها را از روی صفوف برداشته و در ترمینال استاندارد سیستم نمایش می‌دهند. برای آشنایی بیشتر با مفهوم broker و این الگوری طراحی می‌توانید به اینجا مراجعه کنید.

Publisher/Subscriber pattern



Master / Publisher

پیاده سازی این عنصر بدین صورت است که ابتدا توسط TCP به broker متصل می‌شود. سپس فرامینی را توسط کاربر از ورودی دریافت می‌کند. این فرامین به ترتیب createQueue و eQueue هستند که به ترتیب یک صف ایجاد می‌کنند و یک پیام را روی یک صف خاص می‌نویسند. Publisher توسط یک ثابت به اسم synchronous کنترل می‌شود. با تغییر این متغیر از false به true می‌توان این عنصر را از حالت asynchronous به synchronous تبدیل کرد. منظور از حالت asynchronous این است که در زمانی که publisher یک فرمانی را به سمت broker ارسال می‌کند، منتظر جواب آن نمی‌ماند. در حالت synchronous، publisher منتظر جواب از سمت broker می‌ماند. مثلا در حالتی که سر broker شلوغ باشد و در حال پاسخ‌دهی به فرامین سایر کاربران باشد، نمی‌تواند در لحظه به این publisher خاص جواب دهد. در این سناریو، asynch بودن یا synch بودن فرق ایجاد می‌کند. مثلا اگر buffer overflow رخ دهد، در حالت asynch هیچ واکنشی به آن نشان داده نخواهد شد. در حالی که در حالت synch، publisher هر ۲ ثانیه یک‌بار پیام را بازنشر می‌کند.

Broker

در طراحی این عنصر، یک لیست از صفوف وجود دارد. این لیست دارای محدودیتی برابر brokerSize است که به صورت پیش‌فرض برابر ۱۰ قرار داده شده است. هنگامی که یک سرور درخواست ایجاد یک صف جدید می‌کند، یک صف جدید ایجاد شده و به این لیست اضافه می‌شود. اگر حد brokerSize اشباع شود، broker طی پیغامی مناسب به سرور مورد نظر این موضوع را اعلام می‌کند.

در هر صف ۱۰ پیام قرار می‌گیرد. هر پیام حداکثر ۱۰۲۴ بایت طول دارد. فلذا اگر پیامی بیشتر از این مقدار باشد، قیچی شده و ۱۰۲۴ بایت اول آن نگه‌داری می‌شود.

به ازای هر اتصالی که با broker انجام می‌شود، یک goroutine اجرا می‌شود. بدین ترتیب پاسخ‌دهی به درخواست‌های master ها و client ها به صورت concurrent اجرا خواهد شد.

در broker یک صف به اسم userQueue وجود دارد. هنگامی که یک client متصل به broker درخواست deQueue می‌دهد، ابتدا broker آن کاربر را به این صف اضافه می‌کند. اگر این صف پر باشد، goroutine متناظر با آن کاربر به حالت sleep می‌رود. در غیر اینصورت آن goroutine به ابتدای صف نگاه می‌کند. اگر اسم خودش در ابتدای صف باشد، پس نوبت خودش است که از صفی که درخواست خوانش داده است، بخواند. در غیر این صورت باز به حالت sleep می‌رود تا نوبتش برسد. بدین ترتیب یک round robin scheduling ایجاد می‌شود و کسانی که بیشتر از سایرین در صف منتظر دریافت پیام بوده‌اند زودتر پاسخ داده خواهند شد.

Slave / Subscriber

همانند master، این عنصر نیز با یک TCP به broker متصل می‌شود. سپس فرمان deQueue را از کاربر گرفته و به صف مورد نظر مراجعه می‌کند. اگر صف خالی باشد یا صفی با آن نام وجود نداشته باشد، این موضوع را با پیغامی مناسب به کاربر گزارش می‌دهد. در غیر این صورت اگر نوبتش باشد، پیام سر صف را خوانده و بر روی ترمینال استاندارد سیستم نمایش می‌دهد. این عنصر همواره به صورت synch با broker کار می‌کند (چون عملا خوانش از TCP یک عملیات synch است).

Server-Client

در طراحی اولیه publisher/subscriber فرض ما بر این بود که سیستم توزیع شده، یک سیستم متشکل از یک سری کارفرما (master) و یک سری کارگر (slave) است. فلذا کارفرماها همواره صفوف را ایجاد و پیامی بر روی آنها انباشت می‌کردند. در حالی که کارگرها تنها توان خواندن از صفوف را داشتند. به عبارت دیگر، ارتباط موجود یک

ارتباط یک طرفه بود. پس از اینکه در تالار درس اظهار شد که نیاز است ارتباط دو طرفه باشد، این عنصر جدید را به پروژه اضافه کردیم. این عنصر در اصل ترکیب دو عنصر کارفرما و کارگر است فلذا به توضیح اضافه آن نمی‌پردازیم.

نکات تکمیلی

برای اجرای هر سه برنامه نیاز است که حتما از ورودی، آدرس port مقصد را هنگام اجرای برنامه توسط متغیر os.Args وارد کنید. آدرس هر سه برنامه به صورت پیش‌فرض localhost است. فرم زیر، نحوه اجرای عناصر را نشان می‌دهد:

```
go run broker.go <port number>

go run master.go <broker port number>

go run client.go <broker port number>
```

نکته مهم: کد queue.go یک package تعریف شده است. از آن در broker.go استفاده شده است. برای اجرای broker.go نیاز است که queue.go را از قبل در مسیر $GOPATH/go/src/queue/ اضافه کنید وگرنه با مشکل روبه‌رو می‌شوید.

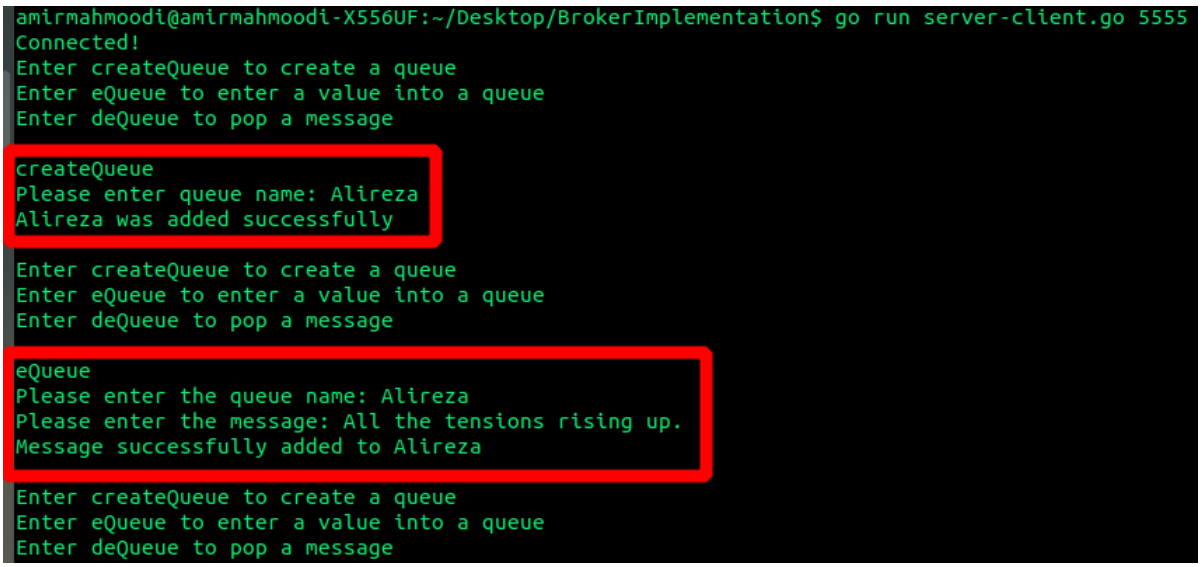
چند سناریو

در این قسمت به بررسی چند سناریو مختلف از اجراهای برنامه می‌پردازیم. سناریوی اول سناریوی عادی اجرای برنامه و دو سناریو دوم سناریو ناخوش‌آیند اجرا هستند. دقت کنید که عکس‌ها با حاشیه قرمز مربوط به دستورات کارفرما و عکس‌ها با حاشیه زرد مربوط به دستورات کارگر هستند. **نکته مهم اینکه تمامی حالات برای وقتیست که ارتباط server-broker در حالت synchronous باشد.** برای حالت asynchronous تنها تفاوت ایجاد شده این است که server منتظر جواب نمی‌ماند فلذا متوجه چیزی نمی‌شود.

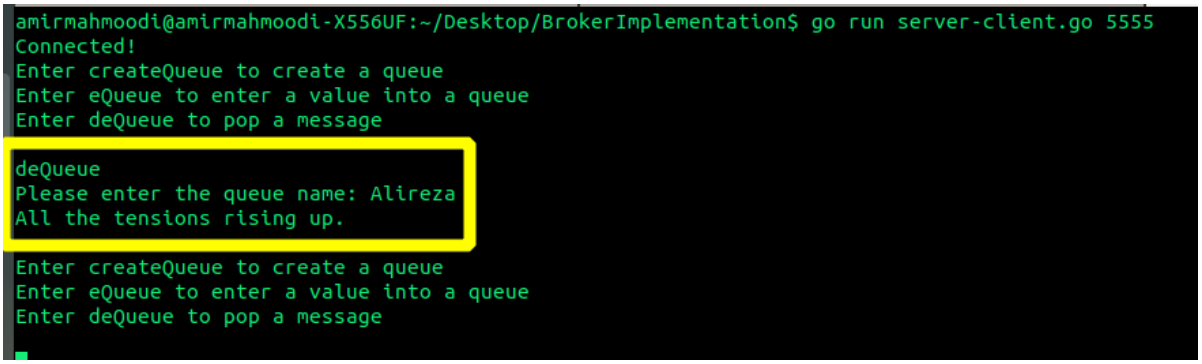
ایجاد یک صف، نوشتن روی آن و خواندن از آن

در این سناریو، یکی از طرفین یک صف ایجاد کرده و بر روی آن یک پیام می‌نویسد. سپس طرف دیگر ارتباط آن پیام را از صف می‌خواند.

createQueue – eQueue



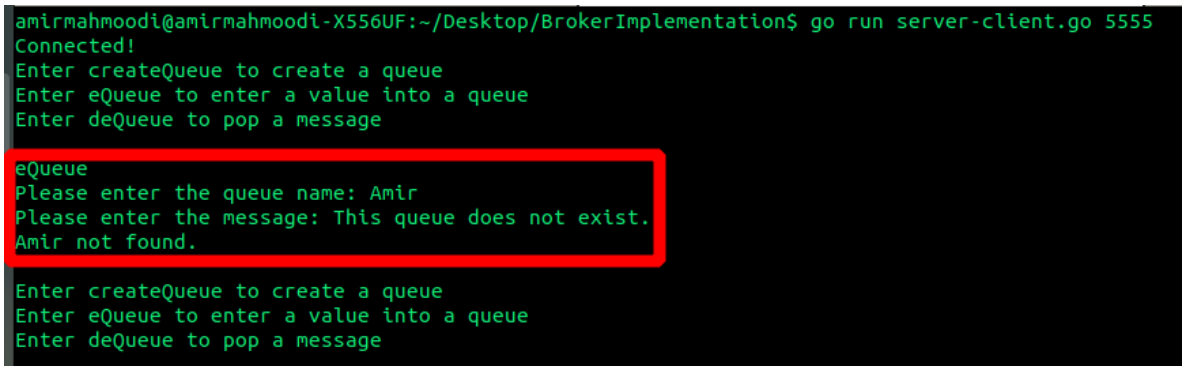
deQueue



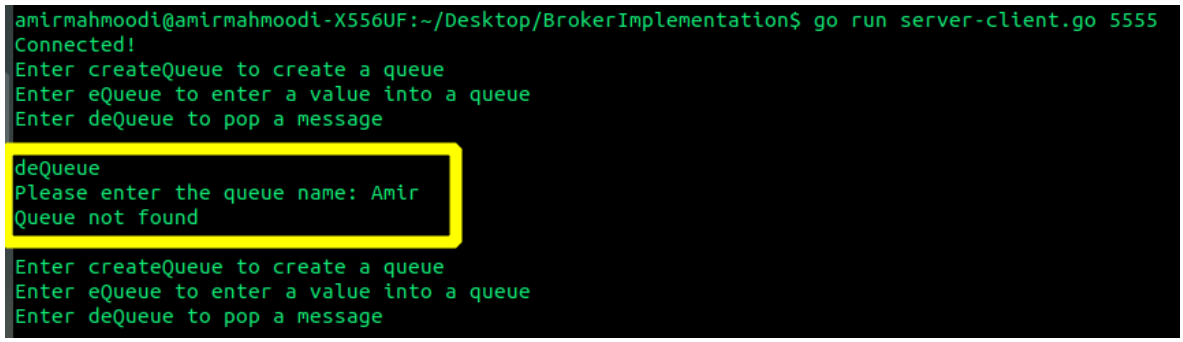
نوشتن و خواندن از صف ناموجود

این سناریو مربوط به حالتیست که صف موجود نباشد.

eQueue



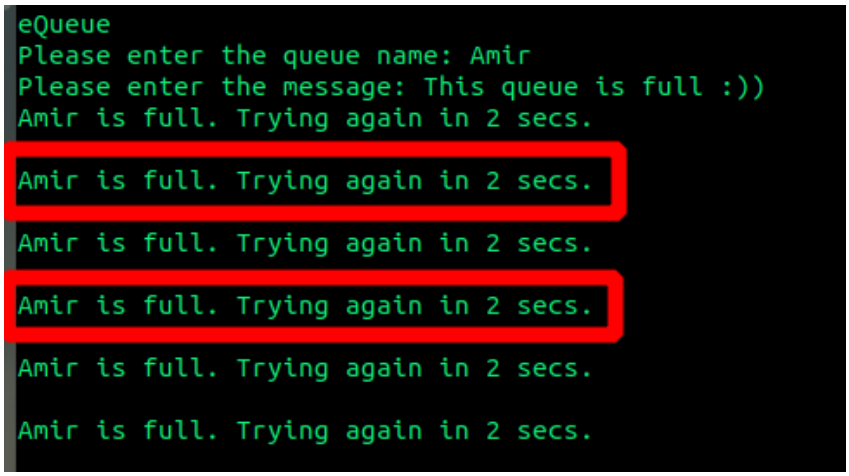
deQueue



نوشتن روی صف پر و خواندن از صف خالی

این سناریو برای حالتیست که صف پر (buffer overflow) یا خالی باشد. همانطور که گفته شد وقتی صف پر باشد، پروتکل بدین صورت عمل می‌کند که هر ۲ ثانیه یک‌بار پیام باز فرستاده می‌شود.

eQueue



deQueue

Broker جایی برای صف اضافی ندارد

این سناریو مربوط به حالتیست که broker دیگر جایی برای ایجاد صف اضافه ندارد.