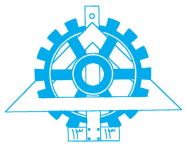
مبانی رایانش توزیع‌شده

Spring 2025



پروژه صفر

مهلت: ۲۰ اسفند ۱۴۰۳

عنوان : مقدمه ای بر مفهوم همروندی (Concurrency) و تست نویسی در زبان برنامه نویسی Go

۱. مقدمه :

هدف این پروژه آشنایی شما با زبان برنامه نویسی Go میباشد. این پروژه از دوبخش تشکیل شده که به صورت کلی از یکدیگر مستقل هستند اما توصیه میشود که تمام توضیحات را به صورت کامل مطالعه کرده و کدهای ارائه شده را قبل از شروع پروژه به صورت کامل بررسی نمایید.

۲. شرح پروژه :

بخش اول : **Implementing a key-value messaging system**

در این بخش شما باید یک سرور پایگاه داده کلید-مقدار (key-value database server) را در زبان Go پیاده سازی نمایید، با این تفاوت که، value ها لیستی از پیام ها هستند که میتوان با یک درخواست Get() مقدار آنها را گرفت و با یک درخواست Put() مقدار جدید به آن اضافه کرد. دراین پایگاه داده کلید ها میتوانند به صورت کامل با درخواست Delete() حذف شوند. همچنین، یک value در لیست را میتوان با یک درخواست Update() تغییر داد. در واقع میتوانید این سیستم را به عنوان یک پایه برای یک سیستم پیام رسان آنلاین بسیار ساده در نظر بگیرید که کاربران میتوانند پیام های سایر کاربران را بخوانند یا تغییر دهند.

ویژگی های سرور پایگاه داده :

در این پروژه یک کد اولیه برای سرور مورد نظر فراهم شده است که شامل عملیات های پایه ای زیر است :

۱. Put(K string, V [ ] byte) : این تابع یک value را در انتهای لیست پیام های مربوط به کلید K وارد میکند (در واقع یک Slice است). توجه داشته باشید که این Slice میتواند در هرلحظه شامل هر تعداد value باشد.
۲. Get(K string) [ ]([ ]byte) : این تابع تمام مقادیر مرتبط با کلید k را به عنوان یک Slice بازمیگرداند.
۳. Delete(K string) : این تابع تمام پیام های مرتبط با کلید k را حذف میکند.
۴. Update(K string, V1 [ ]byte, V2 [ ]byte) : این تابع مقدار V1 را در لیست پیام های مربوط

به کلید K با مقدار V2 جایگزین میکند. اگر V1 در لیست پیام ها وجود نداشته باشد، مقدار V2 به انتهای لیست اضافه خواهد شد.
نکته:فرض کنید که تمام کلیدها و مقادیر از فرمت [a-z][a-z0-9]* پیروی می‌کنند (یعنی شامل حروف کوچک، اعداد و فاصله هستند).

الزامات مورد انتظار در پیاده سازی پروژه :
در نحوه پیاده سازی این بخش لزوما یک روش خاص وجود ندارد و شما آزادید راه حل خودتان را انتخاب و پیاده سازی کنید، فقط توجه داشته باشید که موارد زیر حتما باید درنظر گرفته شود :

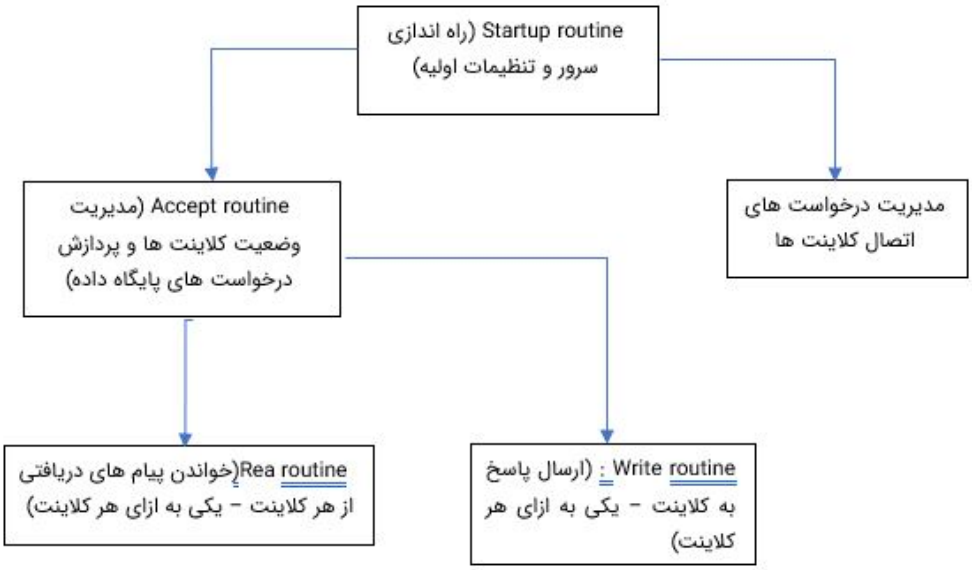
۱.سرور باید بتواند به صورت همزمان با استفاده از Goroutines و Channels کلاینت های خود را مدیریت کند و با آنها تعامل داشته باشد، به این صورت که چندین کلاینت باید بتوانند به صورت همزمان به سرور connect/disconnect شوند.
۲.فرمت درخواست هایی که کلاینت ها به سرور ارسال میکنند باید به صورت زیر باشد :
Put:key:value : هنگامی که کلاینت میخواهد یک value را در سرور قرار دهد.
Get:key : هنگامی که کلاینت میخواهد یک value را از سرور دریافت کند.
Delete:key : هنگامی که کلاینت میخواهد یک key را از سرور حذف کند.
Update:key:oldValue:newValue : هنگامی که یک کلاینت میخواهد برای یک کلید مشخص یک مقدار oldValue را با یک مقدار newValue آپدیت کند.
این چهار نوع پیام، تنها پیام های مجازی هستند که کلاینت میتواند ارسال کند. شما مسئول پردازش این پیام های درخواستی و انتخاب عملیات مناسب جهت اجرای آنها هستید.
۳. زمانی که کلاینت دستور Get را به سرور ارسال میکند، سرور پس از دریافت دستور باید پاسخی به فرمت key:value[newline] به همان کلاینت ارسال نماید، در واقع سرور برای هر مقدر مرتبط با کلید مشخص شده یک بار این پیام را ارسال نماید. شایان ذکر است برای درخواست های Update، Put و Delete هیچ پاسخی نباید توسط سرور به کلاینت ارسال شود.
۴. تمام پیام هایی که ارسال یا دریافت میشوند، باید با کارکتر (/n) newline پایان یابند.
۵.توابع API key-value که در قسمت ویژگی های سرور به آنها شاره شده است (put، Get و ...) به طور پیش فرض "Thread-Safe" نیستند، شما مسئول اطمینان از عدم ایجاد Race condition هنگام دسترسی به پایگاه داده هستید.
۶. شما باید در سرور تابع CountActive() را پیاده سازی کنید، این تابع وظیفه دارد تعداد کلاینت هایی که به سرور متصل هستند را برگرداند.
۷.شما باید تابع CountDropped() را پیاده سازی کنید، این تابع تعداد کلاینت هایی که disconnect

شده اند را برمیگرداند.

۸. سرور باید نسبت به کلاینت هایی که پیام ها را با تاخیر دریافت میکنند (Slow-Reading Clients) پاسخگو و منعطف باشد. برای درک این موضع سناریویی را در نظر بگیرید که یک کلاینت برای مدت طولانی Read را فراخواندی نکند (یعنی پیام های دریافتی خود را نخواند). در این مدت، اگر سرور همچنان به ارسال پیام به این کلاینت ادامه دهد، در نهایت بافر خروجی (Outer Buffer) اتصال TCP پر خواهد شد. هنگامی که بافر خروجی TCP به حداکثر ظرفیت خود برسد، هر فراخوانی جدید Write توسط سرور مسدود (Blocked) میشود و سرور نمیتواند هیچ پیام جدیدی را به کلاینت ارسال کند.
برای حل این مشکل، سرور باید یک صف (Queue) با حداکثر ظرفیت ۵۰۰ پیام را برای هر کلاینت نگه دارد. اگر این صف پر شود، پیام های جدیدی که قرار است به کلاینت ارسال شوند، باید حذف (Dropped) شوند و سرور در این حالت دیگر منتظر نمیماند. اگر کلاینت مجددا شروع به خواندن داده ها کند، سرور باید تمام پیام های باقی مانده در صف آن کلاینت را برایش ارسال کند
(نکته مهم : برای پیاده سازی این مکانیزم باید از یک Buffered channel استفاده کنید تا پیام های ارسالی به هر کلاینت را به طور موقت ذخیره و مدیریت کنید).

راهنمایی :
سرور شما میتواند این فرض را داشته باشد که کلاینت ها هرگز در خواست Delete()، Get() یا Update() را برای کلید هایی که هنوز در سرور ذخیره نشده اند، ارسال نخواهد کرد.

ساختار منطقی پروژه :
یافتن ساختار منطقی مناسب برای کد شما یکی از جنبه های مهم برنامه نویسی همزمان (Concurrent programming) است. اگر چه شما آزادی دارید که از هر ساختاری استفاده کنید، اما ساختار زیر را میتوان حالت ساده و کار آمدی برای این پروژه در نظر گرفت :



بخش دوم : تست نویسی برنامه در Go

در این بخش از شما خواسته شده است که حداقل یک تست کوتاه برای برنامه داده شده (Squarer) بنویسید. شی squarer با استفاده از کانال، اعداد صحیح را دریافت میکند سپس مربع آنها را به کانلی دیگر ارسال میکند. فایل squarer-api.go که به شما داده شده است یک interface است. شما نباید تغییری در این فایل بدهید بلکه فقط باید از این فایل برای نوشتن تست های squarerImpl در فایل squarer-test استفاده نمایید. (بنابراین شما باید تست هایی بنویسید که بررسی کند آیاsquarerImpl به درستی اعداد را مربع میکند یا خیر.)

توجه :

۱. باید حداقل یک تست ساده در squarer-test.go بنویسید. میتوانید بیش از یک تست بنویسید، اما امتیاز اضافی نخواهید گرفت.

۲. پیاده سازی صحیح squarer باید تمام تست های شما را با موفقیت پشت سر بگذارد.

۳. پیاده سازی نادرست squarer نباید بتواند حداقل یکی از تست های شما را پشت سر بگذارند.

۴. تمام تست ها باید یکی از ویژگی های squarer را که در sqarer-api.go تعریف شده است را بررسی کند، نمیتوانید ویژگی های پیاده سازی خاصی را تست کنید یا به صورت تصادفی نتیجه تست را تعیین کنید. دستیاران آموزشی (TAs) تست های شما را بررسی خواهند کرد و اگر راه حل شما مطابق موارد گفته شده نباشد، نمره ای دریافت نمیکنید.

۳. فایل های پروژه :

پوشه p0partA شامل کد مربوط به بخش اول میباشد :

۱. Server-impl.go : این فایل تنها فایلی است که اجازه تغییر آن را دارید، در واقع شما باید قسمت اول پروژه را در این فایل پیاده سازی نمایید.
۲. Kv-impl.go : شامل API مربوط به key-value است که برای انجام عملیات روی پایگاه داده استفاده میشود. پنج تابع درون آن را باید مستقیما در server-impl.go استفاده کنید.
۳. Server-api.go : شامل interface و مستندات مربوط به key-value server است که شما باید پیاده سازی کنید (این فایل را تغییر ندهید).
۴. Server-test.go : شامل تست هایی است که ما برای ارزیابی و نمره دهی به پیاده سازی شما اجرا خواهیم کرد (تست عملکرد برنامه شما).

پوشه p0partB شامل کدهای مربوط به بخش دوم میباشد :

۱. Squarer-test.go : تنها فایلی است که باید برای بخش دوم تغییر دهید، در این فایل تست های مربوط به squarerImpl را بنویسید.
۲. Squarer-api.go : شامل interface و مستندات مربوط به sqarer server است که شما قرار است آن را تست کنید.(نباید این فایل را تغییر دهید).
۳. Sqarer-impl.go : این فایل پیاده سازی صحیح squarer است که از آن برای نوشتن تست های خود استفاده میکنید(نباید این فایل را تغییر دهید).

۴. نکات مهم پروژه :

۱. پیاده سازی های این پروژه باید به صورت فردی انجام شود. همچنین شما به جز کدهایی که در اختیارتان گذاشته شده است مجاز به استفاده از کدهای دیگران یا استفاده از AI نیستید. درصورتی که استفاده از AI یا مشابهت کدها با دیگر پروژه ها کشف شود، نمره صفر به شما تعلق خواهد گرفت.
۲. استفاده از Locks و Mutexes ممنوع است. تمامی همگام سازی ها باید از طریق goroutine ها، channel ها و دستور select مبتنی بر channel در Go انجام شود.
۳. شما فقط مجاز به استفاده از پکیج های زیر در کد خودتان میباشید :
bufio، io، bytes، fmt، net و strconv.

۴. کد شما باید با استفاده از go fmt استاندارد شود و از استاندارد های نام گذاری Go پیروی کند. برای جزئیات بیشتر به لینک های زیر مراجعه نمایید :

https://golang.org/doc/effective_go.html#formatting

https://golang.org/doc/effective_go.html#names