TCP Chat

Alexander Milton – b13alemi – 12 februari 2015

Uppgiftsbeskrivning

Uppgiften tillhanda var att skapa en chattapplikation i Java som nyttjar TCP-protokollet (Transmission Control Protocol) samt JSON för att kommunicera mellan klienter med hjälp av en server.

Implementation

Lösningen tillåter för klienter att koppla upp sig mot en server genom att ange serverns adress, port och ett valfritt användarnamn. Förutsatt att användarnamnet är ledigt godkänner servern klientens förfrågan och läggs till i serverns klientsystem.

Meddelanden skapas och utläses med hjälp av protokollet JSON, som packar ihop relevant data (avsändare, kommandon, eventuella parametrar och meddelanden) till ett JSON-paket som konverteras till en sträng, skickas över TCP-protokollet till servern, packas ihop till ett nytt JSON-paket och läser ut relevant data för att utföra tjänster eller skicka meddelanden.

I serverns finns fyra fundamentala trådar som körs under hela exekveringens gång. Vid behov används en semafor för att skydda hanteringen av klienterna mot *race conditions* och dödläge.

- *ClientConnectionThread.* Tar emot förfrågningar om att gå med i servern. När klienter skapas skickas en förfrågan till servern som nekas (om namnet är upptaget) eller godkännes. Kritisk sektion skyddad med semafor.
- *ClientMessageThread.* Tar emot meddelanden från samtliga klienter och vidarebefordrar dem till en meddelandehanterare, där de behandlas utifrån sitt kommando. Kritisk sektion skyddad med semafor.
- *HeartbeatThread.* Pingar klienterna med några sekunders mellanrum för att undersöka om klienten är uppkopplad. Om inget svar fås antas klientens koppling vara bruten och markeras som frånkopplad. Använder en semafor för att med timer invänta respons från klienter.
- *BringOutYerDeadThread.* Med några sekunders mellanrum kontrolleras vilka klienter som är markerade som frånkopplade och tar bort dessa ur serverns klientlista. Kritisk sektion skyddad med semafor.

Exekvering

En klient startas med hjälp av en kommandotolk och kommandot `java –jar Client.jar <serveradress> <serverport> <användarnamn>.`
Servern startas med hjälp av en kommandotolk och kommandot `java –jar Server.jar <port>.`
Portarna måste överensstämma för att uppkopplingen skall ske korrekt.

När klienten startats och är uppkopplad till servern finns en lista med kommandon och förklaringar att tillgå med hjälp av kommandot `/help` eller `/h`:

```
➢ Type a message to broadcast it to all users
➢ Type /help or /h to show this message
➢ Type /list or /l to view a list of all active users
➢ Type /whisper or /w followed by a username and a message (delimited by
  spaces) to send a private message to that user
➢ Type /cat and squint for some awesome ASCII art
➢ Type /disconnect or /dc to leave the chat
```

UDP vs TCP

Den mest fundamental skillnaden mellan protokollen UDP och TCP är hur meddelandepaket transporteras i praktiken och vilka garantier/risker detta medför. UDP skickar *Datagram Packets* med risk för försummelsefel (*omission failures)* (meddelanden kommer fram i fel ordning, för många gånger eller inte alls) men i hög hastighet. TCP garanterar framkomsten av meddelandepaket i rätt ordning och korrekt antal, men riskerar att vara betydligt långsammare på grund av omfattande overhead. En koppling upprättas mellan klient och server och används för att strömma meddelanden mellan dem.

Felmodell

Ett antal fel kan uppstå med TCP-kommunikation, trots att meddelanden inte försummas likt med UDP.

För att inte klienter skall ”låsas ute” från chatten om deras klient skulle krascha stängas av ser servern till att kontrollera vilka klienter som fortfarande är uppkopplade med hjälp av en *heartbeat*-algoritm. Frånkopplade klienter raderas ur systemet och lämnar möjlighet för nya uppkopplingar att återanvända tidigare användarnamn.

För att inte konflikter skall uppstå mellan trådarna i servern används semaforer för att styra programflödet. Specifikt semaforen `criticalSection` är viktig för att undvika parallelliseringsproblem. Den låses (binärt, endast en tråd har tillgång till den kritiska sektionen åt gången) när kritisk data behandlas, framför allt listan med uppkopplade klienter.

Ett problem som inte har fullständig explicit felhantering är felformaterade JSON-paket. Korrekt angivna meddelanden hanteras felfritt genom programmets gång (även opåverkat av blandade gemener/versaler i kommandon), men om ett meddelande saknar förväntade parametrar finns en risk att servern misslyckas med meddelandehanteringen eller att någon komponent havererar.

Det sistnämna problemet hade kunnat lösas med hjälp av någon form av validering av JSON-paketen för att varna om ett paket är malformerat.

Reflektion

Uppgiften var en givande erfarenhet som gav förtydligad förståelse om TC-protokollet och meddelandekommunikation mellan distribuerade system. Dock finns omfattande problematik med uppgiftens upplägg. Instruktionerna är otydliga och diffusa. Det behövs en konkret och uniform beskrivning av vad som krävs av studentens lösning och rapport, vilka kriterier som täcks av uppgiften samt vilka resurser studenten kan/bör tillgå.

Det känns även olämpligt att inte ha någon handledning inom de sista tre dagarna av inlämningsdatumet. Ett slutgiltigt tillfälle dagen innan inlämning skulle underlätta, då många av de mer komplicerade problemen och frågorna inte sällan uppstår mot slutet av utvecklingen.