Obligatorisk uppgift 2

Chattjänst

Introduktion

I den här obligatoriska uppgiften, som ska lösas 2 och 2, ska delar av en chattjänst implementeras.

En chatt består i regel av en server och flera klienter. Servern tar emot ett meddelande från en klient och vidarebefordrar det till alla klienter som är anslutna till servern. Klienten skickar textmeddelanden, inskrivna av användaren, och presenterar den fortgående konversationen för användaren. Ibland används även en namnserver, där chattservrar registrerar sig, för att hjälpa klienter att hitta chattservrar att ansluta sig till. Vilka tjänster som chatten tillhandahåller samt hur dessa tjänster realiseras beskrivs genom ett applikationsprotokoll.

När ett system blir komplext inför man olika abstraktioner. Ett abstrakt objekt fångar vissa intressanta aspekter av systemet, har ett eller flera gränssnitt som kan användas av andra objekt, samt gömmer detaljer om hur objektet är implementerat. Det finns två stora fördelar med att låta ett system bestå av flera abstrakta objekt. För det första delar det upp problemet med att bygga ett system i mindre och mer hanterbara komponenter. För det andra ger det en mer modulär design, dvs. det blir mycket lättare att underhålla objekt samt att byta ut och lägga till nya objekt.

En typ av abstrakta objekt i ett nätverkssystem kallas kommunikationsprotokoll, ibland kallas de nätverksprotokoll men oftast kallas de bara protokoll. Ett protokoll i applikationslagret kallas oftast ett applikationsprotokoll. Ett protokoll specificerar hur två eller flera parter ska kommunicera. Varje protokoll definierar två olika gränssnitt. Ett servicegränssnitt som används av andra objekt i samma dator som vill använda protokollets service. Samt ett gränssnitt som används för kommunikation mellan parter som använder samma protokoll (peer-to-peer gränssnitt). Förutom på den fysiska nivån sker kommunikationen med motsvarande protokoll på en annan maskin indirekt, kommunikationen sker via de underliggande nätverksprotokollen.

Inom objekt måste det också finnas en bra modularitet så att objekt blir lätta att underhålla och vidareutveckla. Moduler i objekt måste ha tydliga gränssnitt som kanaliserar all åtkomst till objektets tjänster och varje modul måste ha en tydlig och avgränsad uppgift.

Syfte

Syftet med den här uppgiften är i första hand att ge övning i skapandet av klient-serversystem. Lärandedelen för uppgiften syftar till att ge möjlighet till att träna på delar av följande förväntade studieresultat:

- Redogöra för Internets arkitektur, protokoll- och lagerkonceptet, grundläggande säkerhetsaspekter samt några viktiga applikationsprotokoll. (FSR 1)
- Redogöra för hur information överförs från punkt A till punkt B, inbegripet viktiga transportprotokoll och i det sammanhanget relevanta metoder och algoritmer. (FSR 2)

- Förklara hur en distribuerad applikation löser sina uppgifter och varför dess protokoll är utformat som det är. (FSR 5)
- Identifiera relevanta frågeställningar relaterade till dataöverföring. (FSR 6)
- Tillämpa socket- och trådprogrammering i en nätverksapplikation. (FSR 9)

Det tillhörande obligatoriska redovisningstillfället syftar till att examinera delar av följande förväntade studieresultat:

- Tillämpa socket- och trådprogrammering i en nätverksapplikation. (FSR 9)

Övergripande beskrivning

I det följande beskrivs chattjänsten i två perspektiv. Dels ges en kort beskrivning av det ingående protokollet (chattapplikationsprotokoll) och dels beskrivs de ingående delarna (chattklient, chattserver och namnserver).

Ingående protokoll

Chattjänsten realiseras med ett chattapplikationsprotokoll. Chattapplikationsprotokollet tillhandahåller en tjänst där meddelanden kan skickas och tas emot från andra chattklienter. Protokollet har två gränssnitt: ett ”peer-to-peer”-gränssnitt och ett servicegränssnitt. ”Peer-to-peer”-gränssnittet definierar hur två parter inom protokollet kan kommunicera med varandra via paket. Dessa paket kallas Protocol Data Unit (PDU). Servicegränssnittet definierar vilka tjänster som protokollet erbjuder till ovanliggande användande enhet.

Chattapplikationsprotokollet beskrivs i ett separat dokument.

Ingående delar

Chattjänsten består av tre huvuddelar: en chatklient, en chattserver och en namnserver.

Chattklient
Användaren av chattjänsten använder chattklienten för att ansluta sig till en chattsession, skriva och läsa chattmeddelanden, samt avsluta chattsessionen. Chattklienten ska använda sig av ett kommandoradsbaserat gränssnitt. Vid start av chattklienten anges antingen:

- Namnserverns IP-adress eller namn samt portnumret som namnservern lyssnar på, eller
- Chattservers IP-adress eller namn samt portnumret som chattservern lyssnar på.

Chattklienten följer chattapplikationsprotokollet samt nyttjar underliggande transporttjänster.

Chattserver
Chattservern ansvarar för att hålla ordning på vilka klienter som ingår i en session, samt ta emot paket från en klient (som ingår i sessionen) och skicka vidare till övriga klienter i sessionen. Servern har två huvuduppgifter:

- Hålla koll på vilka klienter som är uppkopplade mot servern inklusive deras identiteter.
- Hålla koll på vilka meddelanden som postas för att behandla dem och distribuera dem vidare till samtliga anslutna klienter.

Chattervern följer chattapplikationsprotokollet samt nyttjar underliggande transporttjänster.

Namnserver
Namnserverns funktion är att underlätta för en chattklient att hitta en chattserver att ansluta sig till för att gå med i en chattsession. Chattservern registrerar sig hos en namnserver samt skickar regelbundet ett paket till namnservern för att berätta att den fortfarande är igång. När en chattklient vill ansluta sig till en chattsession börjar den med att fråga en namnserver om vilka sessioner som finns. Därefter väljer chattklienten en chattserver som den ansluter till. Namnservern följer chattapplikationsprotokollet samt nyttjar underliggande transporttjänster.

Uppgift
Ni som går kursen DoD (5DV197) ska implementera en chattserver och en chattklient utifrån det givna applikationsprotokollet. Namnservern är given. Chattservern och chattklienten ska implementeras i C.

Ni som går kursen DoI (5DV198) ska implementera en chattklient utifrån det givna applikationsprotokollet. Chatserver och namnserver är givna. Chattklienten ska implementeras i Java.

För båda kurserna gäller att implementationen av chattjänsten ska vara väl strukturerad och följa det givna applikationsprotokollet. Målsättningen är att alla gruppers implementationer ska kunna kommunicera med andra gruppers implementationer samt den chattserver och namnserver som tillhandahålls. Chattklienten ska ha ett kommandoradsbaserat användargränssnitt.

Uppgiften är uppdelad i tre faser.

Fas 1 – PDU-hantering
I den första fasen ska en modul för PDU-hantering implementeras. I modulen ska PDU:er kunna skapas och läsas. I detta ingår att implementera tester för att försäkra att modulen fungerar korrekt.

Fas 2 – Socketabstraktion
I den andra fasen ska en modul för socketabstraktion implementeras. När en PDU ska skickas ska den läggas i en utkö, sedan skickas den i köordning. När en PDU anländer ska den läggas i en inkö, därefter kan den hämtas i köordning.

Fas 3 – Chattlogiken/-reglerna samt användargränssnitt

I den tredje fasen ska en modul för logiken/reglerna implementeras. Modulen ska implementera det som ska ske vid olika händelser (användaren gör något eller en PDU anländer). Vidare ska ett kommandoradsbaserat användargränssnitt implementeras.

Krav på lösningar

Följande är ett antal krav som ska vara uppfyllda för godkänt resultat på uppgiften (se även kvalitetsskriterier i slutet av detta dokumentet):

- Chattjänsten ska implementeras enligt det givna chattapplikationsprotokollet
- Programmen ska vara kompiler- och exekverbara på institutionens Linux-datorer
    - C-enheterna ska kompileras med `gcc` och flaggorna `-Wall -Werror` Tillsammans med källkoden ska en `Makefile` lämnas in, och programmen ska kompileras med `make chatserver` och `make chatclient`
    - Java-enheterna ska kompileras under Java 8 eller lägre. Programmen ska kompileras med `javac ChatServer.java` och `javac ChatClient.java`.
- Chattservern tar 4 argument i följande ordning:
    - port
    - chattservernamn
    - namnserver-host
    - namnserver-port
- Chattklienten tar 4 argument i följande ordning:
    - identitet (t.ex. ett nickname)
    - ns (för namnserver) eller cs (för chattserver)
    - namnserver-host eller chattserver-host
    - namnsserver-port eller chattserver-port
- Chattjänsten ska ha ett kommandoradsbaserat gränssnitt
- Eftersom UDP inte är tillförlitligt måste det hanteras om ett svar uteblir på en skickad PDU (där svar förväntas) (detta hanteras genom att skicka om paket om svar inte anländer inom 5 sekunder).

Resurser

Till ert förfogande för att lösa uppgiften finns ett antal labbsalar och ett antal handledningstillfällen (enligt handledningsschema på kursens Cambro-sida). Det är tillåtet att implementera på annan plats, till exempel hemma, men tänk på att programmen ska kompileras och vara körbara på institutionens Linux-datorer enligt ovanstående krav.

Till kurs(erna) ges en del övningsmaterial samt övningsuppgifter, dessa finnes under kursens Cambro-sida och fliken *Övningsuppgifter*. Dessa uppgifter ger en bra grund för båda obligatoriska uppgifterna på kursen.

Senare under uppgiften kommer det att på itchy.cs.umu.se:1337 finnas en av kursen tillhandahållen namnserver. Där finns information om en referenschattserver (som tillhandahålls av kursen) som kan användas. Det kommer även att finnas ett http-gränssnitt mot den tillhandahållna namnservern på itchy.cs.umu.se:1338.

Redovisning

Redovisning av uppgiften sker vid tre tillfällen. Respektive tillfälle är kopplade till varsin fas.

Redovisning av fas 1 – PDU-hantering

Redovisning av fas 1 är frivillig. Dock rekommenderas ni starkt att delta, godkänd redovisning ger även examinerande bonuspoäng för kursen, se kursens Cambro-sida, flik *Examination*. Fas 1 redovisas genom deltagande vid ett kodgranskningstillfälle. För datum och tid för kodgranskningstillfället se respektive kurs schema. Vid tillfället ska följande saker tas med:

- Er kod utskriven i tre exemplar.
- Ett ifyllt testprotokoll för de tester ni utfört.

Redovisning av fas 2 – Socketabstraktion

Redovisning av fas 2 är frivillig. Dock rekommenderas ni starkt att delta, redovisning ger även examinerande bonuspoäng för kursen, se kursens Cambro-sida, flik *Examination*. Fas 2 redovisas genom deltagande vid ett kodgranskningstillfälle. För datum och tid för kodgranskningstillfället se respektive kurs schema. Vid kodgranskningstillfället ska följande saker tas med:

- Er kod utskriven i tre exemplar.
- Ett ifyllt testprotokoll för de tester ni utfört.

Redovisning av fas 3 – Chattlogiken/-reglerna samt användargränssnitt

Redovisning av fas 3 är obligatorisk. Fas 3 redovisas genom en demonstration för handledare. Inlämning av kod (och `Makefile` för 5DV197) sker via ett repository på institutionen GitLab. För båda kurserna gäller att testprotokollet är inlämnat i labres som även inkluderar en länk till ert repository senast 48 timmar innan bokad tid för demonstration.[1]

Syftet med demonstrationen är att visa att er implementation är korrekt. Ett givet testprotokoll ska följas för att visa att ni uppfyller alla kvalitetskriterier samt uppfyller testprotokollet som finns i slutet på detta dokument. Tänk på att köra igenom testprotokollet ett antal gånger innan er demonstration för handledare. Om redovisning vid ordinarie redovisningstillfälle resulterar i godkänt betyg för uppgiften ges även examinerande bonuspoäng för kursen, se kursens Cambro-sida, flik *Examination*.

Demonstrationen är upp till 15 minuter lång. Det är upp till er att planera demonstrationen, inklusive att bestämma en lämplig balans mellan heltäckande tester och mer detaljerade tester, t.ex. felhantering. Tänk på att vi vill se en tydlig uppdelning i moduler (motsvarande faserna).

För datum och tid för demonstrationer se respektive kurs schema. Bokning av tid för demonstration sker via Cambro-sidan (information om detta kommer som meddelande). Eventuella ofullständiga lösningar ska kompletteras vid kompletteringstillfälle och bedöms genom en ny demonstration för handledare. För datum och tid för komplettteringstillfälle se respektive kurs schema. Det tredje tillfället för redovisningen, tillika resttillfället, meddelas senare i kursen.

[1] Handledarna måste vara tillagda till ert repository.

Tips

Skriv tester som både kontrollerar att de PDU:er som skapas har rätt format och padding, och tester som kontrollerar att om en PDU är felaktig (har felaktig padding, checksumma etc.). Bygg in validering i varje enskild PDU så att det är enkelt att kontrollera ifall en PDU är korrekt uppbyggd.

Ni rekommenderas att använda någon form av versionshantering. Detta är bra både för att dela kod med sin labbpartner, men också för att kunna ångra ändringar i koden ifall något slutar fungera. Institutionen tillhandahåller GitLab på adressen https://git.cs.umu.se.

Tänk på att en kommunikationspart kan vara långsam och inte skriva all data i en PDU på en gång, samt att en kommunikationspart kan skriva flera PDU:er i samma sändning.

Tänk på att en read läser och en write skriver upp till ett givet antal bytes. Kontrollera returvärdet för att veta hur många bytes det blev. Skriv gärna en rutin read_exactly som anropar read tills alla bytes lästs in.

DoD
Försök skriva automatiska integrationstester. Det innebär att man bygger en automatisk testsvit som t.ex. startar en server och en klient och ser till att klienten kan ansluta till servern. Fördelen med sådana tester är att det går mycket snabbare att kontrollera att allt fungerar än om det måste göras manuellt efter varje liten ändring. Det gör också att man inte riskerar att glömma att köra något test. Eftersom chattklienten är kommandoradsbaserad så kan det vara enklare att skriva automatiska tester i något skriptspråk.

För att ta reda på längd på en UTF-8-sträng räcker det med att använda `strnlen()` eftersom den bara räknar antalet bytes till `'\0'`.

DoI
Alla Javas heltalstyper (byte, short, int, long) är `signed`, d.v.s. de kan ha negativa värden. Det kan leda till buggar när de castas till datatyper som representeras med fler bitar. Kolla upp hur heltalen representeras (two's complement), prioritet för casting jämfört med andra operatorer, samt hur `sign extensions` fungerar. Se sedan till att skriva tester som kontrollerar att denna hantering fungerar (t.ex. användandet av nicknames med större längd än 255 bytes).

Vid läsning av data använd `"sträng".getBytes("UTF-8").length` istället för bara `"sträng".length()`.

Kvalitetskriterier

Uppgiften kommer att bedömas enligt följande kriterier:

Kriterium	**Godkänd**	**Kommentar**	**Ofullständig**
Kompilering	Utan varning	Mindre allvarlig	Allvarlig
Testkörning	Utan fel Följer CLI	Mindre fel Svarar felaktigt på felaktigt beteende	Felaktig output Räknar fel Följer ej CLI Slutar fungera
Läsbarhet	Lagom kommenterat med informativa kommentarer med konsekvent språk Korrekt indentering Konsekvent och bra namngivning Bra och tydlig struktur på koden	Mindre bra kommenterat Något indenteringsfel Mindre bra och/eller ej konsekvent namngivning Mindre problem med struktur på koden	Missvisande eller saknade nödvändiga kommentarer Många indenteringsfel Missvisande och/eller olämplig namngivning Ostrukturerad kod
Felhantering	Korrekt undantag/hantering av returvärden	Några få missade	Många missade
Resurser	Bra användning av trådar och övriga resurser Trådsäkert	Mindre problem med trådar, tex resurser delade i onödan eller onödiga trådar Mindre problem med trådsäkerhet av ej allvarlig karaktär Mindre minnesproblem	Felanvändning av trådar Allvarliga problem med trådsäkerhet Minnesläckor/Minnesfel
Sockets	Används korrekt	Mindre problem	Felaktig användning
Lösningens kvalitet	En bra implementerad lösning av uppgiften	Mindre problem Mindre mängd onödig kod Onödigt krånglig lösning	Större problem Större mängd onödig kod Mycket omständlig lösning
Modularitet (coupling och cohesion)	Uppdelat i funktioner/klasser/delar med tydliga uppgifter	Mindre problem med uppdelning	Felaktig/otillräcklig uppdelning
Abstraktion	Tydlig abstraktion och ansvarsfördelning Tydlig mainfunktion	Något diffus ansvarsfördelning Saknar goda gränssnitt mot structar/klasser	Undermålig ansvarsfördelning Saknar gränssnitt
Korrekt	Följer givet protokoll Löser uppgiften	Mindre avvikelse från protokoll	Följer ej protokoll Löser ej uppgiften
Testprotokoll	Testar efter uppdelat ansvar DoD: Testar beteende vid högre belastning DoD: Testar beteende vid långsam klient/server	Testar bara väldigt enkel användning	Testar inte all funktionalitet

Testprotokollsunderlag

Detta avsnitt ger en grund och ide till testprotokoll som ni ska lämna in, detta är även ett förslag på strukturen för redovisningen. Eventuella förtydliganden kan tillkomma under kursens gång (Observera, inget som ställer högre krav på lösningen).

DOD	DOI
1. **Kompilera** (Makefil) 2. **CLI x3** ○ Om/registrera (**starta om CS**, *start om NS*) ○ **CK->NS->CS & CK->CS** ○ **Acceptera klienter** ○ **Chatta** ○ **Återansluta** ○ *UTF-8* 3. Kod (**CK** och **CS**) ○ **PDU-API** ○ **Kommentarer** ○ **Struktur/Modulisering** ○ **Abstraktion (PDU/***Sockets/Clean API för trådar***)** 4. Felhantering ○ Både CS och CK ○ **NS skickar skräp** ○ **Fel meddelanden från** CK och från CS (padding, checksumma et c.) ○ **Anslutning går sönder** (ej genom QUIT) (CK) ○ *MESS/NICK-validering klient* 5. Trådar ○ **Visa trådar, delade resurser och synk** ○ Stresstest ○ Vad händer om en klient blockerar? ○ Vad händer om 200 klienter spammar? ○ Hur *städas trådar* och **fildeskriptorer** upp när	1. **Kompilera** 2. *CLI* (**ska gå att ansluta både till NS och CS**) ○ **Hämta serverlista** ○ **Ansluta till ref.** och **chatta** (*UTF-8*) 3. Läsbarhet ○ Visa **kommentarer** och **struktur** 4. Vad händer om ○ *NS skickar skräp* ○ CS skickar skräp (fel *checksumma*, *pad*, *OP*) ○ *Anslutningen stängs* 5. Trådar ○ **Visa trådar** ○ Visa *delade resurser* och *synk* (köer) 6. Visa abstraktion ○ **PDU** ○ *Sockets* ○ *Clean API för trådar* (bra abstraktion) 7. **Korrekt tolkning av protokoll**

CK lämnar? ○ Hur undviks **race condition** vid återanvändning av fildeskriptorer? 6. **Korrekt tolkning av protokoll**	