Katalogtjeneste

Introduksjon

Katalogtjenester finnes hyppig i hverdagen. Alt du kanskje vil slå opp på Internettet er sannsynligvis implementert som en katalogtjeneste av noe slag. De skiller seg fra et nettsøk fordi de alltid returnerer den samme informasjonen, og vanligvis i samme rekkefølge hvis det samme søket brukes igjen.

Som den du skal jobbe med i denne oppgaven, er mange av dem hierarkiske. For eksempel gir opplisting av alle nettsidene som tilhører et institutt ved UiO én trestruktur for IFI, en annen for FYS og enda en annen for MAT. Microsofts Active Directory er oppkalt etter prinsippet, og kan svare på spørsmål med mange attributter, som identiteten og autentiseringen til personen som ber om informasjon.

Denne oppgaven er inspirert av enkle og raske katalogtjenester som Network File System som IFIs Linux-maskiner bruker for å montere hjemmekatalogene dine. Denne katalogtjenesten kalles forresten Yellow Pages og den fungerer over UDP. Den er enkel, rask og skalerbar (men fungerer ikke med personlige og mobile enheter). Den er veldig skalerbar og motstandsdyktig mot problemer på grunn av bruken av UDP, men mangelen på tilkoblinger skaper noen utfordringer. En forespørsel fra en klient besvares av serveren ved å sende en eller flere pakker, og transaksjonen blir deretter umiddelbart glemt av serveren. Det er helt "tilstandsløst" og er derfor i stand til å betjene tusenvis eller klienter, noe forbindelssorienterte servere ikke ville kunne gjøre.

Hjemmeeksamenen handler om implementeringen av to deler av en så enkel, rask katalogtjeneste.

1. Den UDP-baserte kommunikasjonen mellom klient og katalogserver,
2. Og klientens mottak av informasjon fra serveren som har formen av et tre (som filene og katalogene på en disk).

Oppgaven

I denne oppgaven skal du implementere klientbiblioteket til en katalogtjeneste.

Scenarioet er strippet ned til de helt grunnleggende funksjonene, og disse er delt i to lag.

Bunnlaget muliggjør sending og mottak av pakker over UDP, hvor hver datapakke bekreftes av den motsatte siden ved hjelp av separate ACK-pakker.

Det øvre laget benytter seg av dette bunnlaget. Her sender klientsiden en enkelt forespørsel (en enkel pakke) til serveren. Forespørselen kan være veldig kompleks, men det spiller ingen rolle for prinsippet. Vi sender derfor ganske enkelt et heltall >1000. Serveren (som er gitt i binær form) slår denne forespørselen opp i databasen og trekker ut potensielt hundrevis av verdier som er organisert i en trestruktur. Serveren organiserer dette treet ved å tilordne IDer til nodene i treet; disse tilordnes av dybde-første-søk-vandringen gjennom treet, og starter med 0 ved roten av treet for enkelhets skyld. Den sender deretter pakker til klienten for å svare på forespørselen. Den sender først en pakke som inneholder antallet noder it treet som klienten bør forvente, og deretter flere ekstra pakker, som hver inneholder noen få trenoder.

Din første oppgave er å implementere og teste bunnlaget. Dette inkluderer funksjoner for å oppdage serveradressen, opprette og slette en struktur for å holde styr på kontakten din og adressen og porten til serveren. Den inkluderer da funksjoner for å sende og motta både datapakker og bekreftelsespakker.

Din andre oppgave er å implementere og teste det øvre laget. Også her trenger du funksjoner for å opprette og slette en struktur for å administrere assosiasjonen til serveren (det kan bare referere til det nedre lagets struktur), deretter funksjoner for å sende en forespørsel og funksjoner for å samle inn svarene.

For det tredje må disse svarene forstås som noder i et tre som inneholder IDer, verdier og informasjon om underordnede noder. Du skal skrive funksjoner som samler treinformasjonen, og til slutt skrive den ut på skjermen. Minnet må selvfølgelig administreres også.

Prekoden

Denne oppgaven kommer med en betydelig mengde prekode inkludert en Makefile. Ved å kalle make oppretter du:

- Biblioteket **libhe.a** som hovedsakelig inneholder koden skrevet av deg i kompilert form
- Den eneste testklienten i det nedre laget **d1_test_client** som kobler libhe.a
- Det øvre lagets testklient **d2_test_client**, som også kobler libhe.a

Biblioteket og programmene kompilerer, men de fungerer ikke. Alle er avhengige av minst én fil som er ufullstendig, da det er overlatt til deg å skrive funksjonene, **d1_udp.c**. Programmet d2_test_client er også avhengig av filen **d2_lookup.c** hvis funksjoner du må implementere. Funksjonene som ikke er implementert er dokumentert i henholdsvis headerfilene **d1_udp.h** og **d2_lookup.h**. Det er i tillegg header-filene **d1_udp_mod.h** og **d2_lookup.h**, som inneholder datastrukturer som du kan endre for å gjøre dem mer nyttige for koden din.

Også servere for testing av kun det nedre laget og for testing av både det nedre og øvre laget kommer forhåndsbygd for flere plattformer.

De følgende filene er inkludert i prekoden.

- To filer som allerede er nevnt som du må endre
  - d1_udp.c
  - d2_lookup.c
- To filer som du har lov å endre:
  - d1_udp_mod.h : inneholder en struktur for å holde informasjon om serveren
  - d2_lookup_mod.h : inneholder en struktur for å beholde serverinformasjon og en for å holde informasjon om treet sendt av serveren
- Flere filer som du ikke burde endre:
  - Makefile : en veldig enkel makefil som kompilerer med debug-informasjon
  - d1_udp.h : headerfil med dokumentasjon for forventet oppførsel av funksjonene i d1_udp.c
  - d2_lookup.h : tilsvarende for d2_lookup.c
  - d1_test_client.c : en testklient som kobles til en d1 testserver og validerer at pakker kan sendes og mottas som forventet
  - d2_test_client.c : en testklient som kobles til en d2 testserver og validerer henting av en trestruktur

Dokumentation

Vi forventer at du oppgir en README.txt-fil med koden din som forklarer hvordan implementasjonen din fungerer, og hvilke elementer du har eller ikke har implementert.

README.txt skal være i samme katalog som kodefilene.

Anbefalinger

Utivklingstrinn

En mulig måte å løse denne oppgaven på er i følgende trinn:

1. Vi anbefaler på det sterkeste at du implementerer de nedre lagfunksjonene i d1_udp.c først. Gå gjennom filen d1_test_client.c og implementer de nødvendige funksjonen ei d1_udp.c i den rekkefølgen du finner dem. Testserveren er pratsom. Sørg for at koden din har mye feilsøkingsutgang.
2. Når funksjonaliteten fungerer, sørg for at du ikke har minnelekkasjer. Bruk valgrind.
3. For det andre, gå gjennom funksjonene i d2_test_client.c og implementer de relevante funksjonene for kommunikasjon i d2_lookup.c. Ignorer funksjonene for å opprette, lagre, skrive ut og slette treet i denne runden.
4. Når kommunikasjonen fungerer, implementer trefunksjonene. Sørg for at utskriften fra d2_print_tree er nøyaktig som beskrevet i headerfilen.
5. Forbedre dokumentasjonen av koden din.
6. Fjern de gjenværende advarslene som vises på grunn av -Wall og -Wextra.
7. Fullfør README.txt

Bruk valgrind

For å sjekke programmet for minnelekkasjer anbefaler vi at du kjører Valgrind med følgende flag:

```
valgrind \
    --leak-check=full --track-origins=yes \
    DITT_PROGRAM
```

Submission

Du må sende inn all koden din i ett enkelt TAR-, TGZ- eller ZIP-arkiv. Ingen andre komprimeringsformater støttes.

Inkluder Makefilen din og inkluder all prekoden. Sørg for at du har kalt "make clean" før du oppretter arkivet. Vi kompilerer leveransen din.

Hvis filen din heter `<candidatenumber>.tar` eller `<candidatenumber>.tgz`, vil vi bruke kommandoen tar på `login.ifi.uio.no` for å pakke den ut. Hvis filen din heter `<candidatenumber>.zip`, bruker vi kommandoen unzip på login.ifi.uio.no for å pakke den ut. Sørg for at dette fungerer før du laster opp filen. Det er også fornuftig å laste ned og teste koden etter å ha levert den.

Arkivet ditt må inneholde Make-filen.

Om evaluasjonen

Hjemmeeksamen vil bli evaluert på datamaskinene til login.ifi.uio.no-poolen. Programmene må kompilere og kjøre på disse datamaskinene. Hvis det er uklarheter i oppgaveteksten bør du påpeke dem i kommentarfeltet i koden og i din README.txt. Skriv om dine valg og forutsetninger i din README.txt som leveres sammen med koden.

Det er mulig å bestå eksamen uten å løse oppgaven helt. Imidlertid forventer vi som minimum at d1_test_client kjører fullstendig og uten minnelekkasjer, og at d2_test_client sender pakker til serveren og mottar pakker fra den.