Proxy Tjeneste

Introduksjon

Proxy-servere har eksistert på Internett i svært lang tid. Slike proxy-servere oppfyller et enormt utvalg av jobber, fra linklagsoperasjon som gjør det mulig for ARP å oppløse navn i flere sammenkoblede nettverk til applikasjonslagsoperasjoner som konverterer videoformater mellom hverandre.

I hovedsak er "proxy" et ord som kan gis til hvem eller hva som helst som gjør noe på vegne av noe annet.

Denne oppgaven er inspirert av IoT-systemer for hjemmeautomatisering. Disse automasjonssystemene omfatter et stort antall enheter, hvor noen kan være små noder med liten regnkraft, mens andre har mye regnekraft. Selv skyservere er en del av mange IoT-systemer.

For å gjøre utvikling og feilsøking enkelt, implementeres kommunikasjon mellom slike IoT noder ofte ved å sende meldinger i XML-format. Noen IoT-noder har imidlertid så lite datakraft og så dårlige nettverksforbindelser at den enorme overheaden til XML ikke kan støttes. Disse IoT-nodene blir deretter inkludert i nettverket ved å legge til en proxy veldig nær den lille IoT-noden, men med litt mer regnekraft og en bedre nettverkstilkobling. Denne proxyen konverterer et svært effektivt, men vanskelig å forstå binært format til XML på vegne av IoT-noden.

Oppgaven

I denne oppgaven skal du implementere en proxy og en rekke hjelpefunksjoner til en proxy som konverterer studentoppføringer frem og tilbake mellom et XML-format og et binært format.

Scenarioet er strippet ned til de helt grunnleggende funksjonene. Det er et eksempel avsender av studentoppføring i XML-format kalt xmlSender, en annen som sender poster i binært format kalt binSender, og en mottaker som enten kan motta XML eller binært format kalt anyReceiver. Senderne leser fra disk, mottakeren skriver til disk.

Alle disse kobles til proxyen, som fungerer som en TCP-server. Når en avsender leser en post fra disk, sender den den til proxyen. Proxyen konverterer denne oppføringen til et internt format og kontrollerer destinasjons-ID-en i den posten. Hvis det er en node med denne IDen koblet til proxyen, konverterer proxyen oppføringen til formatet (XML eller binært) som er passende for denne noden og sender den.

Proxyen må støtte alle kombinasjoner: XML sender - XML receiver, XML sender - binary receiver, binary sender - XML receiver, binary sender - binary receiver.

Studentoppføringer

Oppgaven tar for seg 3 ulike representasjoner av en studentoppføring. Det er en intern representasjon som bruker en datastruktur kalt struct Record som vi anbefaler for implementering av proxyen, og to representasjoner lagret på fil og sendt over nettverket, ved bruk av XML og binært format.

XML format

En eksempelpost i XML-format kan se slik ut:

```
<record>
  <source="A" />
  <dest="D" />
  <username="griff" />
  <id="1003" />
  <group="200" />
  <semester="27" />
  <grade="PhD" />
  <courses>
    <course="IN1020" />
    <course="IN1060" />
  </courses>
</record>
```

Dette er en XML-melding som sendes fra node A til node D. Det gjelder bruker griff som har bruker-ID 1003 og gruppe-ID 200. Han studerer i 27. semester og den høyeste graden han har fått så langt er PhD. Fra alle 1.årskursene ved IFI har han kun tatt IN1020 og IN1060.

Hver linje i dette formatet er valgfri. Dersom en bruker aldri har tatt noen 1. års kurs ved IFI, vil hele blokken fra <courses> til </courses> være fraværende.

Binary format

Hex-representasjonen av den binære versjonen av den samme posten er

```
ff 42 44 00 00 00 05 67 72 69 66 66 00 00 03 eb 00 00 00 c8 1b 03 00 24
```

- `ff` - den første byten inneholder 8 flagg som indikerer hvilke av oppføringene som finnes. I dette tilfellet eksisterer alle oppføringer.
- `42` - kilden er B, 42 er ASCII for B
- `44` - destinasjonen er D, 44 er ASCII for D
- `00 00 00 05` - strenglengden til brukernavnet i nettverksbyte-rekkefølge. Siden den er i nettverksbyte-rekkefølge, kan vi enkelt lese at lengden er 5.
- `67 72 69 66 66` - ASCII-representasjonen for brukernavnet griff. Merk at det ikke er noen avsluttende null.
- `00 00 03 eb` - 0x3EB er heksadesimal for 1003
- `00 00 00 c8` - 0xC8 er heksadesimal for 200
- `1b` - 0x1b er heksadesimal for 27

- `03` - dette er enum-verdien for "PhD" etter vår definisjon. Dette er dokumentert i prekodefilen record.h
- `00 24` - hex-representasjonen av big endian int16_t-verdien som inneholder flagg for tatt kurs. IN1020 har verdi (1<<2) = 4, IN1060 har verdi (1<<5) = 32. Hex for 36 er 0x24.

Internt format

Det interne formatet vi foreslår bruker strukturen definert i record.h. Det følgende viser hvordan en struct-post kan fylles manuelt (helst vil du bruke hjelpefunksjonene). Merk at r.username er tildelt en kopi av strengen griff med sitt eget heap-minne.

```
struct Record r;
r.has_source = 1;       r.source = 'B';
r.has_dest = 1;         r.dest = 'D';
r.has_username = 1;     r.username = strdup( "griff" );
r.has_id = 1;           r.id = 1003;
r.has_group = 1;        r.group = 200;
r.has_semester = 1;     r.semester = 27;
r.has_grade = 1;        r.grade = Grade_PhD;
r.has_courses = 1;      r.courses = Course_IN1020 | Course_IN1060;
```

Du trenger faktisk ikke å bruke `struct Record`, men du kan dra nytte av flere hjelpefunksjoner som allerede er gitt i forhåndskoden.

Precoden

Denne oppgaven kommer med en betydelig mengde precoden inkludert en Makefile. Ved å kalle make vil du lage 4 programmer:

- binSender
- xmlSender
- anyReceiver
- proxy

Disse programmene kompilerer, men de fungerer ikke. Alle er avhengige av minst én fil som er ufullstendig og som du må skrive, connection.c. Programmet proxy er avhengig av proxy.c, som stort sett er tom bortsett fra kommandolinjeparsing og noen tipps. Her er en oversikt over filer som er inkludert i precoden, men som er ufullstendige:

- Connection.c: Denne filen inneholder en skisse av funksjonene som implementerer TCP-funksjonalitet. header-filen connection.h inkluderer deres erklæringer. Du må implementere disse funksjonene. Alle 4 programmene er avhengige av dem.
- Proxy.c: denne filen inneholder main()-funksjonen til proxy-programmet. Du må utvide denne filen for å behandle en hendelsesløkke som kan lytte etter nylig tilkoblede sendere og mottakere, frakoblede mottakere og nylig ankomne meldinger fra alle de tilkoblede mottakerne. Når en melding kommer og en passende mottaker er koblet til, må du kanskje konvertere det ankommende XML- eller binære formatet til det formatet mottakeren forventer. Å skrive proxy er den største innsatsen for hjemmeeksamenen.
- recordFromFormat.c: dette er to hjelpefunksjoner som lager interne datastrukturer på heap kalt Record fra inngangsbuffere som inneholder oppføringer i XML eller i binært format. Du trenger

ikke å implementere disse funksjonene, men vi anbefaler på det sterkeste at du bruker dem i stedet for å gjøre alt i proxy.c. Headerfilen recordFromFormat.h inkluderer deres erklæringer.

Flere andre filer er inkludert i forhåndskoden:

- Makefile: vår makefile, utvidet som du vil
- anyReceiver.c: inneholder main() for programmet anyReceiver
- binSender.c: inneholder main() for programmet binSender
- xmlSender.c: inneholder main() for programmet xmlSender
- record.c / record.h: erklæring av den interne strukturen Record og en rekke hjelpefunksjoner for å opprette og slette Record-objekter på heapen og sette verdier i dem.
- recordToFormat.c / recordToFormat.h: definisjon og erklæring for hjelpefunksjoner som konverterer en intern struct Record til buffere som inneholder oppføringen i XML eller binært format.
- binfile.c / binfile.h: hjelpefunksjoner brukt av binSender for å lese binært format fra fil.
- xmlfile.c / xmlfile.h: hjelpefunksjoner brukt av xmlSender for å lese XML-format fra fil.

Dokumentasjon

Skriv et designdokument som forklarer designvalgene dine samt koden.

Hovedtemaene i designdokumentet bør være TCP-koden du skriver i connection.c, utformingen av hendelsesløkken din i proxy.c og konverteringen mellom XML og binære formater.

Forklar hva som fungerer, og hvis deler av oppgaven ikke er løst vellykket, avklar på hvilken måte de mislykkes. Dokumentasjonen din må være en PDF-fil og den bør være mellom 1 og 2 sider lang (10 pkt skrift eller større).

Tipps

Development steps

One possible way to solve this task is in the following steps:

1. Start by writing the TCP-related functions in connection.h that make it possible to connect and send data over TCP.
    - Hint: use the program `nc` (netcat) to receive data from your `xmlSender` and print it to stdout.
2. Extend `connection.c` and implement the `proxy.c` until it can receive connection from one or more `xmlSenders` at the same time, receive records in XML format from them and write it to the screen.
3. Extend `proxy.c` until it can also receive connections from `anyReceiver` in XML mode.
    - Send dummy data to the receiver first.
    - After that, send all records that you receive from an `xmlSender` to this `anyReceiver`.

    - After that, send records only to that `anyReceiver` if the ID is matching. Connect several receivers with different IDs and make sure that records are sent only to the correct one using the `dest` field in the record.
4. After that, get `binSenders` to talk with `anyReceiver` in Binary mode.
5. After that, get `binSenders` to talk to `anyReceivers` in XML mode.
6. Finally, get `xmlSender` to talk to `anyReceivers` in Binary mode.

These recommendations are built around the idea that XML is readable and debugging is therefore easier. You don't have to follow the steps in this order.

Utviklingstrinn

En mulig måte å løse denne oppgaven på er å følgende denne oppskriften:

1. Start med å skrive de TCP-relaterte funksjonene i connection.h som gjør det mulig å koble til og sende data over TCP.
    - Hint: bruk programmet nc (netcat) for å motta data fra xmlSender og skriv det ut til stdout.
2. Utvid connection.c og implementer proxy.c til den kan motta tilkoblinger fra en eller flere xmlSenders samtidig, motta poster i XML-format fra dem og skrive den til skjermen.
3. Utvid proxy.c til den også kan motta tilkoblinger fra mottakere i XML-modus.
    - Send dummy-data til mottakeren først.
    - Deretter sender du alle poster du mottar fra en xmlSender til denne mottakeren.
    - Etter det, send poster bare til den mottakeren hvis ID-en samsvarer. Koble sammen flere mottakere med forskjellige IDer og sørg for at poster kun sendes til den riktige ved å bruke dest-feltet i posten.
4. Etter det, få binSenders til å snakke med hvilken som helst mottaker i binær modus.
5. Etter det, få binSenders til å snakke med alle mottakere i XML-modus.
6. Til slutt, få xmlSender til å snakke med alle mottakere i binær modus.

Disse anbefalingene er bygget rundt ideen om at XML er lesbar og feilsøking er derfor enklere. Du trenger ikke å følge trinnene i denne rekkefølgen.

Bruk valgrind

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

```
valgrind \
    --leak-check=full \
    DITT_PROGRAM
```

Innlevering

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

Inkluder Makefilen din og inkluder all precode.

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 Makefile, som må ha minst disse alternativene:

- `make` - oversetter kildekoden din til følgende kjørbare programemr: `proxy`, `binSender`, `xmlSender` og `anyReceiver`
- `make realclean` - sletter de eksekverbare programmene og midlertidige filer (e.g. `*.o`)

Om evalueringen

Hjemmeeksamen vil bli evaluert på datamaskinene til login.ifi.uio.no bassenget. Programmene må kompilere og kjøre på disse datamaskinene. Dersom det er uklarheter i oppgaveteksten bør du påpeke det i kommentarer i koden og i dokumentasjonen din. Skriv om dine valg og forutsetninger i dokumentasjonen som du leverer den sammen med koden.