IN2140 Hjemmeeksamen vår 2025

IN2140 hjemmeeksamen skal løses i grupper. På grunn av begrensninger i leveringssystemet Inspera, må alle i gruppen levere løsningen. Siden løsningen er ment å være anonym, kan vi ikke bruke Google-dokumentet til å matche innsendingene deres for dere.

Vi ber dere derfor inkludere en GROUP.txt-fil i innsendingen deres. Denne GROUP.txt-filen skal inneholde ett kandidatnummer per linje, og ingenting annet.

Vi vil ikke kunne sette karakter på gruppemedlemmer som ikke har levert sin kopi av gruppens arbeid. De vil stryke.

Det er instruksjonene vi mottok. Vi beklager denne komplikasjonen.

Labyrinten

Innledning

I denne oppgaven implementerer dere et nettverkssystem som emulerer noen få kommunikasjonsoppgaver som de ulike lagene i et nettverkssystem skal løse. Dere skal implementere én side av et emulert datalinklag (L2), én side av et emulert transportlag (L4), og klienten til en distribuert applikasjon (L5). Sistnevnte vil bli brukt til å kontakte en server for å be om en labyrint, som du bør løse, og deretter returnere løsningen til serveren.

På datalinklaget (L2) fokuserer vi på oppgaven med å sende rammer mellom nabodatamaskiner. Dette laget definerer størrelsesgrensen for rammer, og nekter å motta data for overføring som overskrider størrelsen på rammen etter å ha lagt inn en L2-protokollheader. Den legger også til et checksum-felt, som dekker alle bytes i rammen (header og payload) bortsett fra selve kontrollsumfeltet. Når en ramme ankommer den mottakende L2-entiteten og kontrollsumfeltet er feil, blir rammen forkastet. Hvis kontrollsummen er riktig, overføres rammens payload til laget over.

Vi dropper nettverkslaget (L3) i denne hjemmeeksamenen og jobber kun med datamaskiner som er direkte naboer. I praksis emulerer vi naboskap på L2 ved å bruke IPv4-adresser og porter, noe som betyr at hvert par datamaskiner kan betraktes som direkte naboer. Som en konsekvens bruker L4 L2-funksjoner direkte.

På transportlaget (L4) fokuserer vi på oppgaven med å sende data mellom to datamaskiner når det gjelder pakker. Pakker kan ikke være større enn payloaden til L2-rammer, men i stedet for å forkaste data som er for store, avkorter L4-sendefunksjonen dataene den blir bedt om å sende av en applikasjon. En L4-enhet som vi oppretter for denne hjemmeeksamenen er svært begrenset: den kan bare kommunisere mellom to prosesser. Når en annen prosess (IPv4-adresse eller port) forstyrrer, er oppførselen til L4-enheten udefinert. Hovedtrekket til dette

veldig enkle transportlaget er at det implementerer stopp-og-vent-protokollen mellom to likeverdige entiteter.

På toppen av dette implementerer vi på L5 en labyrintgenerator og løsningsapplikasjon. L5-klienten, som du skriver, ber om en labyrint fra den gitte L5-serveren, samt en start- og sluttkoordinat øverst i midten og nederst i midten av labyrinten. Klienten din finner en vei gjennom denne labyrinten og returnerer løsningen til serveren. Etter å ha gjort dette, avslutter klienten kommunikasjonen med serveren og avslutter.

Steg

I denne oppgaven er de høyere lagene avhengige av hverandre. Det betyr at det er viktig å få L2 til å fungere før L4 og L4 før L5. Vi tilbyr separate binære servere for testing av L2 (datalink-text-server), L4 (transport-test-server) og L5 (maze-server), samt kildekode for de aktuelle klientene (datalink-text-client, transport-test-client, maze-client).

Merk deg at du ikke trenger å implementere hvert av lagene perfekt før du jobber med neste lag over det. Ved å starte serverne med parameteren "-p 0.0", forhindrer du at det oppstår feil i kontrollsummene til L2-rammer ved sending eller mottak. Dette gjør det mulig for deg å starte med en veldig enkel implementering (L2-rammeheaderne og L4-pakkeheaderne må være korrekte, men pakker vil ikke bli droppet eller sendt på nytt).

Fornuftige skritt kan være:

(1) Implementer rammesending og mottak med riktige headere og korrekt kontrollsumberegning på L2. Dette betyr å implementere innledende versjoner av l2sap_create, l2sap_destroy, l2sap_sendto og l2sap_recvfrom_timeout. Hvis du velger "-p 0.0" for serveren, går ingen rammer tapt, og du trenger ikke å implementere en timeout for l2sap_recvfrom_timeout.

(2) Implementer pakkesending og mottak av pakker på L4. Du må implementere typene, sekvensnummeret og bekreftelsesnummeret i pakkehodene i henhold til stop-and-wait. Hvis du velger "-p 0.0", trenger du ikke å håndtere noen tidsavbrudd og retransmisjon, og du trenger ikke gjøre noe arbeid som muliggjør full-dupleks-oppførsel. Rammer vil ikke gå tapt og derfor vil ikke utsending og mottak overlappe hverandre, og det vil ikke være tidsavbrudd. Du implementerer l4sap_create, l4sap_send, l4sap_recv og l4sap_destroy.

På dette punktet kan du enten introdusere rammetap på L2 og en stop-and-wait-implementering med tidsavbrudd og retransmisjon på L4; eller du kan implementere labyrintløsningen over L2- og L4-implementeringer som ikke støtter rammetap.

Til slutt bør du jobbe med den delen som mangler.

L2 laget

L2-entiteten dere implementerer bør emulere datalinklaget ved å sende UDP-pakker til L2-peer-entiteten. En slik UDP-pakke emulerer L2-rammen deres. Rammen (2-PDU) bør ikke være lengre enn 1024 bytes inkludert rammeheaderen.

Rammeheaderen har følgende struktur:

```
struct L2Header
{
    uint32_t dst_addr;
    uint16_t len;
    uint8_t  checksum;
    uint8_t  mbz;
};
```

Detaljene for disse feltene finnes i l2sap.h. Det er viktig å huske at alle disse feltene sendes over nettverket i network byte order. Siden L2Header har en lengde på 8 byte, er det også viktig å huske at payload (2-SDU) ikke kan være lengre enn 1016 byte.

Hvis et høyere lag prøver å sende payload som er større enn payloadstørrelsen, må L2 forkaste dem.

Hvis en L2-enhet mottar en ramme (emulert med en UDP-pakke) som inneholder feil kontrollsum, må rammen forkastes og ikke sendes til det høyere laget. Kontrollsummen beregnes av en byte-vis XOR-operasjon over rammens lengde, som er `len` bytes.

Feltet `dst_addr` må inneholde den samme IPv4-adressen (også i nettverksbyte-rekkefølge) som du bruker i `sockaddr_in`-strukturen som du bruker for å sende UDP-pakken som emulerer L2-rammen.

`mbz` betyr “must be zero”.

L3 laget

Faktisk, om du ønsker det, kan du legge til et L3-lag som ikke gjør annet enn å passere fra L4 til L3. Headeren må være tom og så videre.

L4 laget

Lag L4 implementerer flytkontroll og feilkontroll ved å bruke den klassiske stop-and-wait-protokollen. Den implementerer ikke metningskontroll, den segmenterer ikke pakker, den gir ingen sikkerhet, den etablerer og river ikke ned forbindelser mellom peer L4-entiteter.

Hver L4-entitet er i stand til å kommunisere med nøyaktig én likestilt L4-entitet, og må avsluttes når disse to enhetene ikke lenger kommuniserer med hverandre. Det er ikke mulig å tilbakestille tilstanden til en L4-entitet, noe som ville tillatt å begynne å kommunisere med en annen L4-entitet. Hvis en annen L4-enhet forstyrrer kommunikasjonen mellom to entiteter, er resultatet uforutsigbart.

Headeren på L4 er veldig enkel:

```
struct L4Header
{
    uint8_t type;
    uint8_t seqno;
    uint8_t ackno;
    uint8_t mbz;
};
```

Den er dermed 4 byte lang, noe som betyr at L4 nyttelasten (4-SDU) kun kan ha 1012 bytes.

Typen til en L4-pakke kan enten være `L4_DATA`, `L4_ACK` eller `L4_RESET`. Verdiene er definert i l4sap.h. Betydningen av de to første er åpenbar, men `L4_RESET` forteller peer-entiteten at en entitet stenger ned og at peer-entiteten burde gjøre det samme. Denne meldingen kan sendes flere ganger.

`seqno` er enten 0 eller 1 og indikerer sekvensnummeret til en datapakke i henhold til stop-and-wait. Startverdien er 0.

`ackno` er enten 0 eller 1. Dette indikerer bekreftelsesnummeret som sendes som svar på en datapakke som er mottatt, i henhold til stopp-og-vent-protokollen.

`mbz` må være null.

L5 laget: labyrinten

Applikasjonen utfører 4 trinn før den slås av:

- Klienten sender en tekstmelding som inneholder "`MAZE <tilfeldig nummer>`".
- Klienten venter på svar. Dette svaret er en serialisert Maze-datastruktur som beskrevet nedenfor.
- Etter å ha løst labyrinten, sender klienten labyrinten tilbake til serveren.
- Klienten sender tekstmeldingen `QUIT` til serveren og slår av L4-enheten.

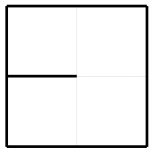
Maze-pakken har følgende format:

```
struct Maze
{
    uint32_t edgeLen;
    uint32_t size;
    uint32_t startX;
    uint32_t startY;
    uint32_t endX;
    uint32_t endY;
    char     maze[];
}
```

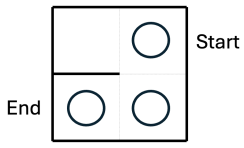
Alle header-verdier er i nettverksbyte-rekkefølge. `edgeLen` indikerer hvor bred og høy en labyrint er. `size` er `edgeLen*edgeLen` for enkelhets skyld. `startX` og `startY` er labyrintkoordinatene der klienten må begynne å søke etter en sti gjennom labyrinten, `endX` og `endY` er posisjonen søket må nå. `maze` er en sekvens av bytes som indikerer i hvilke retninger hver celle i en labyrint er åpen. De mulige verdiene er `left`, `right`, `up`, `down`, og hver kombinasjon av disse. Dette er bitverdier som finnes i `maze.h`.

Bytene i labyrinten lagres som en sekvens av rader uten skilletegn, så hvis labyrinten er en 2x2 labyrint, vil den variable labyrinten inneholde 4 bytes, den første byten er øverst til venstre, den andre øverst til høyre, den tredje nederst til venstre, den siste nederst til høyre. Labyrinten vi bruker kommer alle til å være mindre enn 1012, så den passer inn i en enkelt L4-pakke.

En 2x2 labyrint med verdiene
`{ right, left | down, right, left | up }`
vil representere følgende labyrint:



Klienten som mottar denne labyrinten må løse stien fra startX, startY til endX, endY. Hvis `startX==1, startY==0` og `endX==0, endY==1`, vil løsningen være
`{ right, left | down | mark, right | mark, left | up | mark }`
og se slik ut:



Klienten serialiserer labyrinten etter å ha merket den løsende ruting med `mark`-bits og sender den tilbake til serveren.

Precoden

De 3 hovedfilene der dere implementerer L2, L4 og labyrintløsningsfunksjonalitet:

- l2sap.c
- l4sap.c
- maze.c

Dere bør implementere dummy-funksjonene i disse filene. Vi oppfordrer dere til å legge til hjelpefunksjoner. Det vil gjøre oppgaven med å implementere den nødvendige funksjonaliteten

veldig mye enklere. Du kan også lage flere header- og kodefiler, men du må selvfølgelig redigere CMakeLists.txt for å inkludere dem.

Header-filene deres er de neste 3. Dere bør ikke endre maze.h. Du kan utvide L2-entitets-informasjonen i `struct L2SAP` i l2sap.h, men det er sannsynligvis ikke nødvendig. Dere må utvide L4-entitets-informasjonen i `struct L4SAP` i l4sap.h. Dere bør ikke endre rammeheaderen `struct L2Header` eller pakkeheaderen `struct L4Header`.

Dere må ikke endre parametrene eller returverdiene til L2- eller L4-funksjonene, fordi vår testkode krever at de er nøyaktig som utlevert.

- l2sap.h
- l4sap.h
- maze.h

De tre klientfilene er ment å teste L2-funksjonaliteten mot vår server datalink-test-server, L4-funksjonaliteten mot vår server transport-test-server, og maze-client.c er klienten som ber om labyrinten fra vår labyrint-server. Filen maze-plot.c plotter en labyrint, der 'X's vegger og 'o'er er merker.

- datalink-test-client.c
- transport-test-client.c
- maze-client.c
- Maze-plot.c

Til slutt er det CMake-filen som hjelper til med å lage Makefiles for kompilering av koden din.

- CMakeLists.txt

Råd

Oppdateringer

Dette er en ny oppgave som ikke ligner noen tidligere. Det er derfor sannsynlig at noen feilrapporter kommer inn, og forhåndskoden vil kreve forbedringer. Ta jevnlig titt på Beskjeder på kurssiden https://www.uio.no/studier/emner/matnat/ifi/IN2140/v25/index.html .

Den oppdaterte versjonen av prekoden, disse dokumentene og de binære serverne er tilgjengelig fra Github her: https://github.uio.no/IN2140v2/in2140-v25-he

Bruk valgrind

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

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

Submission

For å sende inn koden deres må dere bruke Inspera som nevnt ovenfor.

For å lage zip-filene dine, bruk kommandoen "make package_source" i byggekatalogen din. Dette vil opprette en fil HomeExam-1.0-Source.zip som inneholder kildekoden, CMakeLists.txt og .txt-filer.

Sørg for at katalogen med kildekoden inkluderer GROUP.txt og også README.txt hvis du forklarer deler av løsningen din.

Om evalueringen

Hjemmeeksamen vil bli evaluert på datamaskinene som heter login.ifi.uio.no. Programmene må kompilere og kjøre på disse datamaskinene. Det vil være uklarheter i oppgaveteksten. Dere kan peke på dem i kommentarene i koden og i deres README.txt. Skriv om deres valg og forutsetninger i deres README.txt som leveres sammen med koden.

Det er mulig å bestå eksamen uten å løse oppgaven helt. En A vil kreve en stort sett komplett og veldig god, men ikke en perfekt løsning. Faktisk anser vi det som svært vanskelig å løse oppgaven perfekt. Spesielt en implementering av en full-duplekssløsning på L4 og teste den med transport-test-server og en pakketapsverdi >0 vil være en utfordring (og ikke engang prøv tapssannsynligheter høyere enn 0,1).

Vi vil teste løsningen din med programmene vi deler ut, men det er veldig viktig for deg å vite at vi også vil teste implementeringene av funksjonene dine separat ved hjelp av små testprogrammer. Det er viktig at du ikke endrer navn, parameter eller returverdi for funksjonene, fordi hver endring vil føre til at noen tester mislykkes.

Vi forventer **som et minimum** at L2-testene fungerer uten feil (ingen minnelekkasjer osv.), og at funksjonen mazeSolve() skaper en perfekt vei fra start til slutt (ingen omveier, ingen blindveier, kobler sammen start og end).

For å presisere, krever ikke minimum at mazeSolve() fungerer i programmet maze-client.c. Vi vil koble det til et ikke-nettverksbasert testprogram, og det er tilstrekkelig om det løser hver løsbare 31x31 labyrint som vi bruker til å teste den.