UPush chattjeneste

Introduksjon

Chattjenester har eksistert siden internettets første dager, akkurat som SMS har eksistert siden den digitale mobiltelefoniens første dager. Chat gir en enkel tjeneste som lar folk kommunisere på egenhånd, og tillater, men krever ikke, umiddelbare svar. Den er diskret sammenlignet med telefon- eller videosamtaler, og langt mer lettvektig sammenliknet med e-post.

En slik enkel tjeneste er ikke nødvendigvis pålitelig, men brukere forventer å vite om en melding er mottatt på destinasjonen eller ikke. Meldinger skal ikke lagres på disk dersom mottakeren er frakoblet.

Oppgaven

I denne oppgaven skal du skrive en UDP-basert chattjeneste kalt UPush. Chattjenesten er veldig rask og enkel, og verken pålitelig eller sikker. Den er inspirert av chattjenester i den virkelige verden der brukere registrerer chatteprosessene sine på en kjent server for å bli funnet, men hvor meldinger mellom klienter sendes direkte uten å gå gjennom serveren. Alle kommandoer og alle meldinger er menneskelesbare klartekstmeldinger kodet i ASCII. Serveren er inspirert av DNS-servere, men i stedet for å koble vertsnavn til IP adresser, tilordner den kallenavn til IP-adresser og porter.

Pakketap skal emuleres ved å bruke den gitte prekoden, og pakketap må håndteres av både klientene og serveren. For sending av UDP pakker skal du ikke bruke biblioteksfunksjoner, men prekodefunksjonen `send_packet()`.

Registrering og oppslag

UPush-serveren er det første elementet i UPush-chattjenesten som starter. Serveren lagrer ikke informasjon om klienter på disken, kun i minnet. Den venter evig på meldinger fra UPush-klienter. Disse meldingene kan enten være registreringsmeldinger eller oppslagsmeldinger. UPush-serveren tar et portnummer og en tapssannsynlighet som kommandolinjeargumenter.

```
> ./upush_server <port> <tapssannsynlighet>
```

`port` er portnummeret som serveren bruker for å motta og sende UDP-pakker (datagrammer). Hvis den gitte porten ikke kan brukes (kanskje den allerede er i bruk av en annen prosess), må serveren skrive ut en feilmelding og avslutte.

`tapssannsynlighet` er en verdi mellom 0 og 100 som gir prosentandelen av pakker som skal forkastes av prekodefunksjonen `send_packet()`. Prosentandelen må konverteres av deg til en sannsynlighetsverdi mellom 0 og 1, og du må kalle prekodefunksjonen `set_loss_probability()` med denne sannsynlighetsverdien i `main()` før du sender noen pakker.

Registrering

Når en klient sender en registreringsmelding, betyr det at klienten ønsker å bli funnet av andre klienter som bruker et kallenavn (“nick”). Klienten sender bare registreringskommandoen med sitt nick til serveren, ved å sende en pakke som inneholder en streng på formen `"PKT nummer REG nick"`. Dette kallenavnet kan bare bestå av ASCII-tegn, det kan ikke være lengre enn 20 byte, og det kan ikke inneholde whitespace (mellomrom, tabulator, retur).

Nummeret er sekvensnummeret for en stopp-og-vent-protokoll som du implementerer for kommunikasjon mellom klienten og serveren (i motsetning til vanlig stopp-og-vent har du imidlertid lov til å bruke andre sekvensnumre enn bare 0 og 1). Når serveren svarer på en klients registreringsmelding med sekvensnummer M, bruker serverens svarmelding også sekvensnummeret M.

For å registrere en klient må serveren kjenne klientens IP-adresse og portnummer. Serveren ekstraherer klientprosessens IP-adresse og port fra informasjon gitt av `recvfrom()`-funksjonen, som den bruker for å motta meldingen fra UDP-socketen. Serveren oppbevarer slike registreringer i minnet i en passende datastruktur (kanskje en liste eller en array). Når en nick allerede er registrert, erstattes den gamle IP-/portinformasjonen med den nye IP-adressen/-porten. UPush-serveren bekrefter registreringen ved å sende en OK-melding på formen `"ACK nummer OK"` til klienten.

Oppslag

Når en klient sender en oppslagsmelding, inneholder den en nick. Formatet på oppslagsmeldingen er `"PKT nummer LOOKUP nick"`. Serveren sjekker om denne nicken er registrert. Hvis det er det, returnerer serveren nicken, IP-adressen og porten til klienten som sendte oppslagsmeldingen. Et vellykket oppslag skal resultere i en melding på formen `"ACK nummer NICK nick IP adresse PORT port"`. Hvis den ikke er registrert, returnerer serveren en «ikke funnet»-melding, som har formen `"ACK nummer NOT FOUND"`.

En **UPush-klient** er et program som tar brukerens nick, IP-adressen og porten til en UPush-server, en *timeout*-verdi i sekunder og en tapssannsynlighet på kommandolinjen. Når UPush-klienten starter, prøver den å registrere nicken sin på serveren umiddelbart (dvs. ved å sende `"PKT nummer REG nick"`). Den venter enten på en OK-melding fra serveren eller på tidsavbrudd (timeout). Hvis tidsavbruddet kommer først, skriver UPush-klienten ut en feilmelding og avslutter. Hvis klienten mottar en OK-melding, går den inn i en hendelsesløkke for å vente samtidig på både UDP-pakker og brukerinndata. Denne hendelsesløkken kalles **hovedhendelsesløkken**.

```
> ./upush_client <nick> <adresse> <port> <timeout> <tapssannsynlighet>
```

`nick` er klientens kallenavn.

`adresse` og `port` er serverens IP-adresse og port.

`timeout` er tiden i sekunder mellom en klients handling (som å sende en pakke) og når klienten antar at konsekvensen av denne handlingen (som å motta en ACK-pakke) ikke vil skje. Den brukes når klienten skal vente til den sender noe på nytt eller til den gir opp noe.

`tapssannsynlighet` har samme betydning her som for serveren. Klienten må kalle prekode-funksjonen `set_loss_probability()` med sannsynligheten beregnet fra den gitte prosenten før en pakke sendes for første gang.

Meldingsutveksling

Input fra socket

Når en UDP-pakke fra en annen klient mottas, forventer klienten at nyttelasten til pakken har formen "`PKT nummer FROM fra_nick TO til_nick MSG tekst`". Klienten må verifisere to forhold:

- Den må verifisere at formatet er korrekt (mer detaljer nedenfor).
- Hvis det er tilfelle, må den verifisere at `til_nick` er identisk med sin egen nick.

Hvis begge betingelsene er oppfylt, skriver den ut "`fra_nick: text`" til *standard output* (`stdout`). Den må også sende en melding tilbake til avsenderen som bekrefter at meldingen ble mottatt med følgende format "`ACK nummer OK`". Hvis én av de to betingelsene ikke er oppfylt, skriver den ut en feilmelding til *standard error* (`stderr`). Hvis dette skjer, bør klienten svare med "`ACK nummer WRONG NAME`" eller "`ACK nummer WRONG FORMAT`" avhengig av feiltilstanden. Hvis begge vilkårene mislykkes, står du fritt til å velge én av dem.

Klienten må kunne oppdage duplikater. Klienten bør bruke stopp-og-vent-protokollen for dette (bortsett fra at du ikke er begrenset til sekvensnumrene 0 og 1). Når stopp-og-vent brukes, er det maksimalt én utestående melding for en klient til enhver tid. Dette betyr at klienten kun trenger å lagre det siste nummeret mottatt fra hver av de andre klientene. Dupliserte meldinger skal besvares som originalen, men ingenting skal skrives ut.

Input fra `stdin`

Når brukerinput mottas fra standard input (`stdin`), forventer klienten at den består av ordet "`QUIT`" eller en linje som inneholder en melding, på formatet "`@til_nick tekst`". En inntasting avsluttes når brukeren trykker return (nylinjen skal **ikke** være en del av meldingen). Maksimal lengde på `tekst` bør være 1400 byte. Hvis brukerinndata er lengre, bør klienten ignorere ytterligere bytes.

Hvis input er "`QUIT`", frigjør klienten alle ressurser (lukker sockets, frigjør minne, etc.) og avslutter.

Hvis input har formen "`@til_nick tekst`", sjekker klienten først om den allerede kjenner IP-adressen og porten for kallenavnet `til_nick`. Den gjør dette ved å slå den opp i en **nick-cache** i klientens minne (denne cachen er tom når klienten starter). Hvis det ikke er tilfelle, går klienten inn i en oppslagsprosedyre. Det er tillatt at klienten blokkerer (ignorer innkommende pakker og ignorer inputtekst fra brukeren) til denne oppslagsprosedyren er fullført. For å utføre et oppslag, sender klienten en melding på formen "`PKT nummer LOOKUP til_nick`" til UPush-serveren. Tre ting kan skje:

1. Klienten mottar ikke svar fra serveren innen tidsavbruddsperioden. Den gjentar oppslagsprosedyrer 2 ganger til før den skriver ut en feilmelding til `stderr` og utfører QUIT-prosedyren.
2. Klienten mottar en melding på formen "`ACK nummer NOT FOUND`", hvor nummeret er en kopi av nummeret som klienten sendte. Klienten skriver ut feilmeldingen "`NICK til_nick NOT REGISTERED`" til `stderr`. Den går tilbake til hovedhendelsesløkken
3. Klienten mottar en melding på formen "`ACK nummer NICK til_nick adresse PORT port`". Klienten lagrer denne informasjonen i minnet i sin **nick-cache.**

Når avsenderen kjenner mottakerens IP-adresse og portnummer, enten fra sin egen cache eller via serveren, sender den sin melding direkte til adresse/port-paret.

Når en klient sender en melding, må den håndtere det mulige tapet av enten selve meldingen eller den tilsvarende bekreftelsen. Den bruker en protokoll inspirert av stopp-og-vent-protokollen for å gjøre dette (du har lov å bruke andre tall enn bare 0 og 1), men med et ekstra oppslagstrinn:

- Når en melding sendes og ingen ACK mottas før `timeout` sekunder (verdien for `timeout` er tatt fra parameteren til `upush_client`-kommandoen), gjentar klienten overføringen én gang.
- Hvis den andre overføringen mislykkes, utfører klienten oppslagsoperasjonen for nicken igjen (fordi den andre klienten kan ha registrert en ny IP/port i mellomtiden).
- Hvis oppslagsoperasjonen mislykkes, skriver klienten ut meldingen "`NICK til_nick NOT REGISTERED`" til `stderr`. Den stopper sine forsøk på å levere meldingen.
- Hvis oppslagsoperasjonen er vellykket, prøver klienten to ganger til å sende meldingen, men nå til det nye mottatte IP/port-paret.
- Hvis dette også mislykkes, skriver klienten ut meldingen "`NICK til_nick UNREACHABLE`" til `stderr`.

Sende flere meldinger og meldinger til flere nick

Om du velger å lagre meldingene i en lenket liste eller lese fra `stdin` etter behov er opp til deg. Andre datastrukturer for lagring av meldinger er også fine. Det kan være vanskelig å lese etter behov når klienten kommuniserer med flere klienter, så vi anbefaler at meldinger leses og lagres i klienten. Merk at dette må gjøres separat for hver mottaker. Med andre ord må hver mottaker som klienten kommuniserer med ha sitt eget forventede antall og kø av meldinger.

Hjerteslag

Alle klienter avslutter uten å informere serveren. For å forhindre at foreldede oppføringer akkumuleres over tid, må klienter sende en registreringsmelding hvert 10. sekund.

Serveren skal ikke bruke noen registrering som ikke har blitt oppdatert i løpet av de siste 30 sekundene til å svare på en klients oppslagsforespørsel. Serveren bør returnere "`ACK number NOT FOUND`" i stedet og fjerne nicket fra serverens liste over registrerte nicks på et passende tidspunkt.

Blokkering og avblokkering av brukere

Klienter skal kunne blokkere nicker som de ikke ønsker å motta meldinger fra. Pakker fra en blokkert nick mottas og pakkene deres analyseres, men ingenting skrives ut på `stderr` eller `stdout`. Blokkering av en nick forhindrer også at klienten selv sender meldinger til den blokkerte nicken. Når en melding sendes eller mottas, må klienten sørge for at nicken ikke er på blokkeringslisten før meldingen behandles.

For å legge en nick til blokkeringslisten, må brukeren skrive "`BLOCK nick`" på `stdin`. For å fjerne en nick fra blokkeringslisten, må brukeren skrive "`UNBLOCK nick`" på `stdin`. Blokkering og oppheving av blokkering er helt lokale operasjoner som håndteres inne i klienten.

Dokumentasjon

Skriv et designdokument som forklarer dine designvalg samt koden. Forklar hva som fungerer, og hvis deler av oppgaven ikke er løst vellykket, forklar på hvilken måte de mislykkes. Dokumentasjonen din må være en PDF-fil, og den bør ikke være lengre enn 1 side (10 pkt skrift eller større).

Meldingssammendrag

Alle meldinger er ASCII-meldinger, inkludert teksten i tekstmeldingen. Det betyr at du ikke vil kunne bruke tegn som ø og å. Linux tilbyr flere hjelpefunksjoner for å sjekke om en `char` inneholder et ASCII-tegn, et tall eller et mellomrom (mellomrom, tabulator, retur): `isalpha(c), isnum(c), isspace(c)` og mer.

Tabellen nedenfor gir en oversikt over meldingstypene og deres formater. Ordene som alle er store bokstaver bør ikke endres. Ordene som alle er små kan og bør endres. Tilstedeværelsen av `[ ]` indikerer at ordet er valgfritt. For eksempel vil en positivt tekstrespons være "`ACK nummer OK`", men hvis det er en feil med formatet skal det være "`ACK nummer WRONG FORMAT`".

Meldingstype	**Format**
Registreringsmelding	`"PKT nummer REG nick"`
Registrering OK	`"ACK nummer OK"`
Oppslagsmelding	`"PKT nummer LOOKUP nick"`
Oppslagssvar ved feil	`"ACK nummer NOT FOUND"`
Oppslagssvar ved suksess	`"ACK nummer NICK nick adresse PORT port"`
Tekstmelding	`"PKT nummer FROM fra_nick TO to_nick MSG tekst"`
Tekstrespons	`"ACK nummer [OK/WRONG NAME/WRONG FORMAT]"`

Råd

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

```
valgrind \
    --track- origins=yes \
    --malloc-fill=0x40 \
    --free-fill=0x23 \
    --leak-check=full \
```

```
--show-leak-kinds=all \
DITT_PROGRAM
```

Netcat

Netcat er et program som kan brukes til å sende og motta UDP-meldinger til og fra en bestemt port. Du kan kjøre Netcat på følgende måte:

```
> nc -u <adresse> <port>
```

Dette kan hjelpe deg med å feilsøke og teste serveren og klientene før alt er fullt implementert. Det du skriver inn etter påkalling av kommandoen vil bli sendt som en UDP-pakke til mottakeren. Hvis programmet avsluttes med en feil, er det sannsynligvis forårsaket av at ingen lytter til den gitte porten på den gitte IP-adressen. Sørg for at serveren lytter til riktig port på riktig maskin.

Utviklingstrinn

En mulig måte å løse denne oppgaven på er i følgende trinn. Vi anbefaler at du setter tapssannsynligheten til 0 før du begynner å teste at tapsgjenopprettingen og gjenoverføringene fungerer.

1. Lag serveren
    a. Les parametere fra kommandolinjen og sett tapssannsynlighet ved å kalle `set_loss_probability()`.
    b. Lag datastrukturer for registrerte klienter
    c. Lag en socket og prøv å sende registrerings- og oppslagsmeldinger ved hjelp av Netcat (bare for å se at mottaket fungerer)
    d. Implementer registrering. Legg til informasjon i datastrukturen som svar på registreringsmeldinger.
    e. Implementer oppslag. Svar på oppslag ved hjelp av datastrukturen og informasjonen i forespørselen.
    f. Test den ved hjelp av print-statements og Netcat for å sende registrerings- og oppslagsmeldinger.
2. Opprett klienten
    a. Les parametere fra kommandolinjen og sett tapssannsynlighet ved å kalle `set_loss_probability()`.
    b. Opprett en socket og prøv å sende noe til serveren.
    c. Implementer registrering.
    d. Implementer et oppslag av en kjent nick (dvs. en nick som er skrevet rett inn i koden) og lagre informasjonen i en passende datastruktur. Dette er et trinn som skal gjøre det lettere å oppdage feil tidlig.
    e. Implementer sending og svar på meldinger. Prøv å sende data til en kjent nick ved å bruke informasjonen som er lagret. Implementer et svar på innkommende meldinger.
    f. Implementer parsing av `stdin` slik at du kan erstatte den kjente nicken med en brukergitt nick. Dette vil innebære å gjøre oppslag for å få nødvendig informasjon
    g. Gjør klienten i stand til å lytte til socket og `stdin` samtidig
3. Implementer timeout for tapsgjenoppretting for både server og klient. Dette er trinnet når tapssannsynlighet skal settes til en verdi større enn enn 0.

    a. Implementer stop-and-wait for registreringsmeldinger. Bruk select for å få et tidsavbrudd for gjenforsøk.
    b. Implementer stop-and-wait for tekstmeldinger. Utvid hovedhendelsesløkken for å få et tidsavbrudd for gjenforsøk.
4. Implementer hjerteslagmeldinger
    a. Implementer på serveren og test med Netcat
    b. Implementer på klienten
5. Implementer blokkering av sending og mottak fra spesifiserte nicks

Levering

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

Inkluder Makefilen din og inkluder forhåndskoden.

Hvis filen din heter `<kandidatnummer>.tar` eller `<kandidatnummer>.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 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 en Makefile , som skal ha minst disse alternativene:

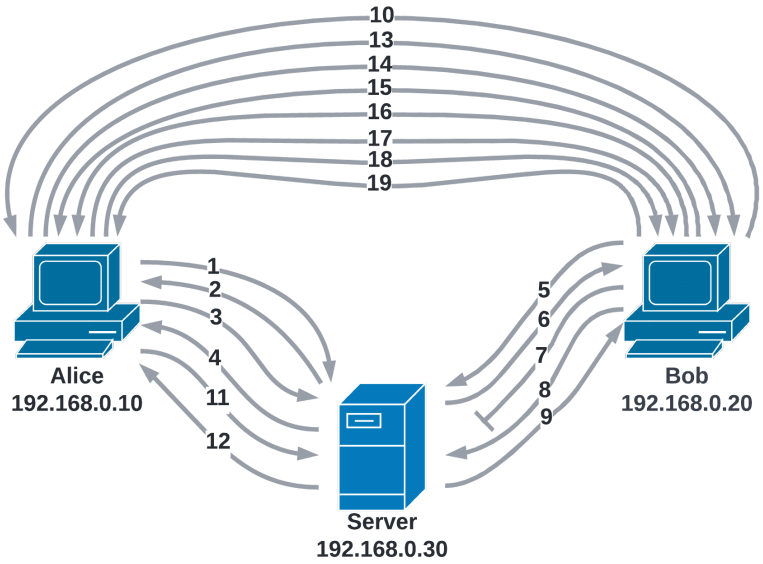
- `make` – kompilerer begge programmene dine til kjørbare binære filer `upush_server` og `upush_client`
- `make all` – gjør det samme som make uten noen parameter
- `make clean` – sletter de kjørbare filene og eventuelle midlertidige filer (f.eks `. *.o`)

Om evalueringen

Hjemmeeksamenen vil bli evaluert på datamaskinene til `login.ifi.uio.no`-poolen. Programmene må kompilere og kjøre på disse datamaskinene. Dersom det er uklarheter i oppgaveteksten bør du påpeke dem og skrive dine valg og forutsetninger i et eget dokument og levere det ved siden av koden.

Vedlegg: Eksempelscenario

Dette vedlegget viser hvordan et bestemt scenario utspiller seg, som involverer meldingsutveksling mellom to klienter og en server på et lokalnettverk. Først registrerer Alice seg. Hun prøver å slå opp Bob, men han er ennå ikke online. Så registrerer Bob seg og prøver å slå opp Alice. Imidlertid forsvinner hans andre pakke på mystisk vis underveis. Heldigvis slipper den første automatiske gjenoverføringen igjennom. Etter å ha fått Alices adresse, sender Bob henne en melding. Hun er i stand til å svare etter et nytt oppslag – denne gangen med suksess. Til slutt utveksler de et par meldinger. Merk at det er tre par kommuniserende parter, men fire serier med sekvensnumre (for enkelhets skyld starter hver med et unikt siffer i listen nedenfor) siden, i motsetning til TCP, etablerer ikke UPush en forbindelse mellom partene.



1. [Alice->Server] PKT 0 REG Alice
2. [Server->Alice] ACK 0 OK
3. [Alice->Server] PKT 1 LOOKUP Bob
4. [Server->Alice] ACK 1 NOT FOUND
5. [Bob->Server] PKT 0 REG Bob
6. [Server->Bob] ACK 0 OK
7. [Bob->Server] PKT 1 LOOKUP Alice
8. [Bob->Server] PKT 1 LOOKUP Alice
9. [Server->Bob] ACK 1 NICK Alice IP 192.168.0.10 PORT 42394
10. [Bob->Alice] PKT 0 FROM Bob TO Alice MSG God morgen, Alice!
11. [Alice->Server] PKT 0 LOOKUP Bob
12. [Server->Alice] ACK 0 NICK Bob IP 192.168.0.20 PORT 18690
13. [Alice->Bob] ACK 0 OK
14. [Alice->Bob] PKT 0 FROM Alice TO Bob MSG Du og, Bob!
15. [Bob->Alice] ACK 0 OK
16. [Bob->Alice] PKT 1 FROM Bob TO Alice MSG Kaffe eller te?
17. [Alice->Bob] ACK 1 OK
18. [Alice->Bob] PKT 1 FROM Alice TO Bob MSG Kaffe, takk!
19. [Bob->Alice] ACK 1 OK