Dokumentasjon

NB: LES README.md

Alle deler av oppgaven er løst. Både klient og server avsluttes uten minnelekkasjer uansett tilstanden ved avslutning (så vidt jeg har klart å teste). Klientene tillater alltid at man skriver input, selv ved lookup. Jeg har valgt å bruke lenkeliste som datastruktur gjennomgående. Jeg har en klientliste som holder styr på alle klientene man kan snakke med. Hver av disse har en meldingsliste i seg som fungerer som en kø for meldinger som skal bli sendt med stop-and-wait. I tillegg har jeg en blokkeringsliste som holder styr på nicks man har blokkert. Meldinger fra brukeren blir lagt bakerst i meldingslista til den tilhørende klienten. Timeouten på select() er uavhengig av timeouten til meldinger (og er satt til 60 ticks per sekund).

For å lettere takle meldingene som blir sendt har jeg laget en meldingsstruct. Hver gang en melding blir mottatt blir den dekonstruert av deconstruct_message-funksjonen, som gjør den om til en meldingsstruct om mulig og gir den en type avhengig av innholdet. En respons (også en meldingsstruct) blir konstruert basert på verdiene i denne structen og tilstanden hos mottakeren (responsen får f.eks type MSG_WR_NAME dersom meldingen som ankom var av type MSG_TEXT men to_nick ikke var lik klientens egen nick). Denne responsen blir gjort om til en char-streng rett før den blir sendt av construct_message. Meldingsstructen inneholder også antall sendeforsøk og om nicken i meldingen har ført til en LOOKUP eller ikke. En melding av type MSG_AFFIRM_ACK har f.eks attempts == 1 (fordi den bare skal sendes en gang), mens en melding av type MSG_LOOKUP har attempts == 3 fordi den skal prøve å sendes tre ganger før man gir opp.

Serveren er relativt enkel. Den har en liste med klienter som den oppdaterer hver gang det kommer en registreringsmelding, eller dersom en av de lagrede klientene går tom for tid. Den mottar meldinger, dekonstruerer dem, oppdaterer klientlista si om nøedvendig og konstruerer en respons avhengig av meldingen og klientlistas tilstand.

Klienten er mer interressant. Det er mye som skal holdes styr på her. Jeg har valgt å implementere stop-and-wait ved hjelp av en meldingskø og en "postkasse" (client->active_msg) per klient. Meldinger ligger i meldingskøen til postkassen blir ledig. Meldinger sendes bare fra postkassen (bortsett fra ACKs). Slik overholdes stop-and-wait, meldinger ligger i postkassen til de har mottatt en riktig ACK eller til de går tom for attempts. Jeg har i hovedsak delt arbeidsoppgavene knyttet til denne implementasjonen i tre funksjoner:

Funksjonen send_active_msg() ser på det neste som ligger i postkassen, og sjekker om det har brukt opp sine attempts eller ikke. Dersom det ikke har det, blir meldingen konstruert av construct_message og sent til mottakeren (ingen meldinger kommer til postkassen uten en struct sockaddr_in). send_active_msg returner en int med verdi definert i upush_helper.h. Dette brukes av client_send() for å vite hva den skal gjøre (ACTIVE_MSG_SENT: vent i timeout sekunder, ACTIVE_MSG_EXPIRED/ACTIVE_NO_MSG: send den neste så fort som mulig, ACTIVE_SERV_NO_RESPONSE: avslutt, serveren er sannsynligvis nede). Denne funksjonen kalles hvert tick på hver klient.

Funksjonen prepare_next_message() blir kalt hvert tick per klient, så fremt postkassa er tom. Den ser på den neste meldingen i meldingskøen og gjør den klar for å bli puttet i postkassa. Om det er en melding som skal til en klient sjekker funsjonen om klienten som "kalte" på denne funksjonen (ikke egentlig da, dette er ikke java) har en IP lagret. Om den har dette kopierer man addressen fra klienten over til meldingen og legger den i postkassa. Dersom man ikke har noen addresse, blir en lookupmelding med nicken i meldingen lagt i postkassa, og meldingen blir værende først i køen. Meldingens looked_up blir satt til 1. Neste gang postkassa er tom vil det være fordi lookupmeldingen er ferdig behandlet. Meldingen blir da behandlet av prepare_next_message() igjen, og dersom lookupen var vellykket vil IP-en ligge i klienten og meldingen blir sendt. Dersom den fortsatt ikke ligger der er det fordi lookupen var mislykka. Da sjekker funksjonen meldingens looked_up, som nå er satt til 1. I dette tilfellet blir meldingen bare kastet ut av meldingskøen.

Funksjonen handle_incoming() kalles hver gang vi leser noe fra socket. Den dekonstruerer meldingen som har ankommet og utfører ulike funksjoner avhengig av meldingstypen. Om en respons kreves konstruerer den en slik og plasser den i riktig kø (ved hjelp av add_to_queue). Om det var en ACK fjerner den meldingen som ligger i riktig postkasse dersom noen av klientene hadde en melding som ventet på denne. Om det var en tekstmelding skriver den denne ut dersom alle kriterier er oppfylt (ikke blokkert, riktig navn, ikke duplikat).