Server: Serveren lytter hele tiden etter pakker på socket, og sender respons basert på innholdet i pakken. Den fjerner utdaterte klienter etter fd-er har fått noe data, hvis tiden siden forrige opprydding er større enn «rense frekvensen» (90 sekunder). Den lagrer klienter i et array som allokeres og reallokeres med realloc når klienter blir lagt til. Har også lagt inn at man kan avslutte serveren ved å skrive QUIT inn i stdin, slik som i klienter.

Klient: Klienten sin håndtering av pakker ut og inn er asynkron. Oppgaven er løst ved å ha en meldings kø, som holder styr på om meldinger er lookup meldinger, «in flight» (forsøkes å sendes) samt annen relevant info for pakken/meldingen. Klienten lar det bare være 1 lookup pakke som kan være «in flight» om gangen, og det kan være 1 melding «in flight» for hver klient den skal sende til. Select times ut hvert sekund, og det sjekkes da om er lengere enn timeout siden meldings køen har blitt gått igjennom. Hvis det er tilfellet, vil alle meldinger i køen bli gått igjennom og håndtert etter nevnte regler. Klienten lagrer informasjon om andre klienter i en lenke-liste etter hvert som den får kontakt eller søker opp klienter. Har også valgt å legge til klienter etter det mottas en melding fra en ukjent klient, for å håndtere at like pakker ikke vises på nytt. Heartbeat meldinger sendes på formatet «PKT number BEAT nick» til server, som oppdaterer tider for klientene. Har ikke håndtert pakketap for disse da det ikke er nevnt i oppgaven.

Har en konstant i common.h filen som heter DEBUG. Når denne er satt til 1 ser man hva server og klienter sender og receiver (med fargekoding!) samt litt ekstra. Valgte å la den være 0 i innleveringen så man bare ser meldinger i klienter, og bare registreringsmeldinger i server.

Forutsetninger og Uklarheter

- Litt uklart om klienten skulle kontakte serveren 3 ganger eller 1 gang for lookup, etter å ha tapte pakker til en kjent klient. Tolket det som at den her bare skal forsøke å kjøre lookup 1 gang, ellers skrive ut feilmelding. Også usikker på om det her skulle være «NOT REGISTERED» eller «UNREACHABLE», hvis man ikke har fått kontakt med serveren. Endte opp med å bare ha NOT REGISTERED hvis man faktisk fikk respons fra serveren med «NOT FOUND», ellers «UNREACHABLE: No response from server lookup».
- Tolket oppgaveteksten som at klienten bare skulle forsøke å registrere seg én gang hos serveren, og ikke gå inn i main event loop hvis registrerings pakken skulle gå tapt.
- Kunne lagt inn at klienters heartbeat tider også oppdaterte seg hvis klienter sendte lookup forespørsel til serveren, men lot være da dette ikke er nevnt i oppgaveteksten.
- Opprydding av klienter vil ikke skje i løsning hvis ikke serveren sin socket får noe data (select blokkerer). Istedenfor å legge in timeout på selecten, tenkte jeg det er like greit ettersom det ikke vil være så viktig å rydde vei for ny data, uten at nye klienter registrerer seg. (og socket da får data).
- I min løsning legger klientene ukjente klienter til i kjente klienter, hvis de mottar en pakke fra dem og klienten ikke enda er lagt til. Da trenger ikke klienten og sende enda en lookup melding til server for å svare med ack.
- Hvis en pakke ikke når fram til en klient vil neste pakke få samme pakkenummer som tidligere pakke og dermed ikke vises, regner med at denne «edge-casen» ikke trengs å håndteres da dette ikke er nevnt i oppgaveteksten. Klienten kan heller ikke være sikker på at pakken ikke har nådd fram, da det også er mulig at ACK gikk tapt istedenfor. En løsning som velger å ikke endre pakkenummer hvis man pakken går tapt vil dermed ha motsatt problem.
- Ved å sette timeout på select til 1, vil dette være minste mulig timeout for klienter, da det ikke vil kunne sjekkes om man skal sende meldinger oftere en hvert sekund. Regner med at dette går fint da det ikke eksplisitt nevnes at man skal kunne håndtere timeout på 0 i oppgaven. Testet med å sette timeval i select til {0, 0} med timeout også som 0, men virker som systemet uansett rundet opp til 1. Antar at det har noe med dette å gjøre: «Note that the timeout interval will be **rounded up** to the system clock granularity, and kernel scheduling delays mean that the blocking interval may overrun by a small amount.”