## Antakelser

* I denne besvarelsen gjorde jeg noen antakelser, fordi oppgaveteksten kunne tolkes på flere måter.
* Dersom en klient prøver å sende en melding til klient 2 fra cache og ikke får ACK to ganger vil han gjøre et lookup call. Etter dette vil klienten prøve å sende meldingen på nytt med oppdatert informasjon. Ut ifra oppgaven valgte jeg å ikke bruke stop and wait på lookup kallet eller det siste kallet. Dette er fordi oppgaven spesifiserte at klienten skulle prøve å sende melding to ganger først. Dermed antar jeg at oppgaven ville at den tredje sendingen til klient 2, etter lookup kallet tilsvarer det tredje forsøket i stop-and-wait protokollen.
* Jeg antok at både heartbeat og første registreringsmelding skal implementere stop and wait.
* Rundt i programmet kaller jeg på ulike check\_error-funksjoner. Disse tar inn en rekke argumenter der minnet skal bli frigjort. Dette fungerer helt fint i upush\_server.c og upush\_client.c, det kan være noen få variabler ikke er mallocet i noen tilfeller i hjelpemetodene i de andre klassene. Alle variablene er initialisert i common.c så alle filene har tilgang til variablene. Jeg valgte å ikke fokusere veldig mye på dette da det ble sagt i en forelesning at dette ikke var så nøye.
* I oppgaven sto det at alt skulle være ascii. Fordi det ikke sto noe om noen feilmelding dersom meldinger ikke kommer frem antok jeg at dette bare var veiledende for å vise at dette ikke var et krav at skulle fungere. Jeg sjekket derimot om brukernavnet er ascii ettersom det muligens kan oppstå problemer ellers.

## Sending og mottak av pakker med stop and wait

Begge programmene starter med å sette opp en socket og mallocer en rekke variabler. Dette er de variablene som alltid blir frigjort dersom noe skjer feil. Deretter starter en while-loop som kjører helt til et nøkkelord blir skrevet inn fra tastaturet. Begge bruker select og har if sjekker for nettverksinput og input fra tastaturet. Ved nettverksinput blir meldingen delt opp i et dobbelt char peker array delt på mellomrom. Brukere blir lagt til i en lenkeliste. Når du legger til en bruker sletter den og frigjør alle med samme navn for å senke minnebruk.

Serveren sjekker hvilken type melding det er og sender et svar tilbake basert på inputet direkte med send\_packet() metoden som ble vedlagt oppgaven. Serveren benytter ikke stop and wait når den sender selv.

Klienten bruker et køsystem implementert som en lenkeliste for å sende meldinger. Det er en teller «counter» som oppdaterer seg hvert sekund ved hjelp av clock\_gettime(). Når klienten skal sende en melding legger den til en struct Message inn i lenkelisten. Structen her inneholder blant annet en meldingstekst, mottaker, en int status og et tidspunkt den skal bli sendt basert på «counter» (for eksempel så sendes første stop and wait melding tidspunktet «counter + timeout», den andre «counter + (2×timeout)», osv.). Alle stop and wait meldinger blir lagt til samtidig som den første meldingen. Det fins tre ulike status verdier, som sier oppgaven til meldingen: 1 er pakker som sendes på tidspunktet sitt, 0 skal ikke sendes, -1 sjekker ut alle meldinger med samme nummer og sletter mottakeren fra cache og -2 stopper programmet (brukes bare etter de tre første forsøkene på å koble seg til serveren). Programmet itererer gjennom alle meldingene og fullfører oppgaven til alle meldinger med samme eller lavere tidspunkt basert på counter, avhengig av status. Hver gang en klient mottar en ACK blir alle meldinger med samme nummer, få statusen sin satt til 0. For at minnet ikke skal bygge seg opp blir alle meldinger som har tidspunkt som er «8×timeout» mindre enn counter bli slettet fra lenkelisten og frigjort fra minne. Dermed er det ikke noen minnelekasjer dersom serveren kjører i en lang periode. For at en melding ikke skal bli printet to ganger dersom en ACK blir et pakketap har jeg et peker-array med int som nummeret til meldingen blir lagt til dersom klienten printer. Dersom nummeret allerede fins blir ikke noe printet.

## Heartbeat

Heartbeat blir implementert ved at det sendes en registreringsmelding hvert 10. sekund. I starten av while-loopen sjekker programmet om «counter % 10 == 0». Dette gjelder båder server og klient. Dersom dette er sant sendes det heartbeat. Det er en variabel heartbeat\_amount som passer på at dette ikke skjer mer enn en gang hvert sekund.Brukere blir lagret i en lenkeliste struct User som blant annet har en verdi int counter. Når serveren oppretter brukeren setter den denne verdien til counteren til serveren. Funksjonen i lenkelisten «lookup\_user()» som finner en bruker returnerer ingen brukere der «serverCounter – userCounter > 30».