Designdokument – Hjemmeeksamen IN2140

Både serveren og klienten oppfyller alle kravene som oppgaven stiller, og sender og mottar bekreftelse på mottatte pakker som oppgaven beskriver. Jeg har skrevet enkle beskrivelser på alle metodene jeg har skrevet i header-filene, så velger å ikke forklare de fleste ytterligere her. Jeg har i denne oppgaven valgt å implementere stop and wait ved at hver klient har en int pkt_count som inkrementeres hver gang en ny pakke sendes (ikke ved nye forsøk på «samme pakke». Når den når 10 starter den på 0 igjen, slik at pakkenummeret alltid er et tegn langt. Serveren svarer derimot bare på hver pakke med pakkenummeret til pakken, og trenger derfor ikke å gjøre det samme.

I både klienten og serveren har jeg valgt å skrive en metode (parse_pkt()) som tar imot en pakke som den sendes fra senderen (f.eks. «PKT 0 REG Navn») og ekstraherer den nyttige informasjonen og lagrer den i et struct pkt. Struct pkt er implementert forskjellig i klienten og serveren. I serveren lagrer den kun nicket til avsenderen og pakkenummeret. I tillegg returnerer funksjonen en int som beskriver typen pakke ved hjelp av enum pkt_type, for å avgjøre hva som skal gjøres med pkt-structen. Jeg har gjort det på nesten samme måte i klienten, bare at jeg her lagrer pakketypen (enum pkt_type), pakkenummeret, meldingen og nicket til avsenderen. Her har jeg også valgt å bruke melding-pekeren til å ta vare på en eventuell ip-adresse og portnummer, da ved å lagre disse på henholdsvis index 0 og 16 i meldingsstrengen. Det ville kanskje vært ryddigere å bare lagre disse som sine egne char-pekere eller eventuelt en sockaddr_in, men jeg valgte heller å gjenbruke meldingspekeren.

Serveren er ikke veldig kompleks, og etter at den har startet opp reagerer den egentlig bare på innkomne meldinger og svarer på disse. Serveren tar vare på informasjon om lagrede klienter i en lenket liste hvor hver node inneholder et struct sockaddr_in, nicket til klienten og en time_t med info om når den sist mottok et hjerteslag fra klienten. Jeg har, for enkelhets skyld, valgt å bruke denne samme datastrukturen i klienten, selv om den her ikke har behov for informasjon om når den sist sendte et hjerteslag. Metodene for manipulering av listen er også derfor de samme. Den vil gå gjennom alle lagrede klienter og sjekke om de har send et hjerteslag hver gang den mottar en melding og select() ikke blokkerer den. Alle klienter som ikke har send en registreringsmelding de siste 30 sekundene vil da bli fjernet. Måten jeg løser dette på er at hver gang en den mottar et hjerteslag fra en klient, oppdaterer den time_t last_update til den daværende tiden (time(NULL)). Alle klienter hvor last_update er lavere enn det daværende tidspunktet minus 30 sekunder blir frigjort. I klienten har jeg implementert hjerteslag ved å ha et eget timeval-struct, hb, med en tv_sec på 10 sekunder, slik at den minst en gang hvert 10. sekund sjekker om den har sendt et hjerteslag de siste 10 sekundene, og sender et hjerteslag dersom dette er tilfelle. Jeg vurderte å løse dette ved å bruke fork, og la en child-prosess ta seg av dette, men synes egentlig at dette var en enklere løsning.

I klienten har jeg, i tillegg til en konstant BUFSIZE laget en større MSGSIZE som er stor nok til å ta vare på alle meldinger som vil kunne inneholde en melding på 1400 tegn. I både klienten og serveren er BUFSIZE derimot akkurat stor nok til å kunne ta vare på alle andre mulige pakker, hvor den tar utgangspunkt i at nicks ikke kan være lengre enn 20 tegn, og pakkenummeret er mellom 0 og 9 (altså et tegn). I klienten har jeg valgt å skrive en print_msg()-funksjon som skriver ut en melding som mottas på en ryddig måte. Dette gjør det lettere å bare skrive ut innkomne meldinger fortløpende, som er spesielt nyttig når klienten venter på acks. Dersom select() ikke blokkerer på grunn av en ny melding skriver klienten ut denne, og dekrementerer telleren som sjekker hvor mange ganger klienten prøver å sende en melding. Dette vil gjøre at klienten ikke vil gå videre før den har ventet lenge nok på en ack dersom den får inn en melding før den timeouter. Derimot vil klienten potensielt vente lengre på en ack enn den skal, da den begynner å vente på nytt når dette skjer, men jeg mener dette er den mer fornuftige måten å gjøre det på.

Blokkeringen av andre nicks har jeg løst ved å lage en lenket liste som tar vare på blokkerte nicks, og at klienten rett og slett bare ikke skriver ut meldinger mottatt fra blokkerte nicks. Den vil derimot sende acks til avsenderen, slik at avsenderen ikke bruker tid på å sjekke med serveren om klienten er aktiv. Avsenderen vil derfor tro at mottakeren får meldingene. Derimot når klienten forsøker å sende meldinger til blokkerte nicks, sjekker jeg med en gang om mottakernicket er blokkert, og klienten vil ikke forsøke å sende meldingen eller sjekke om klienten eksisterer.