INF3190 - Hjemmeeksamen 2

Formelt

Denne oppgaven er karaktergivende og skal løses individuelt. Karakteren som gis teller omlag 20 % på sluttkarakteren. Oppgaven blir vurdert etter hvor stor grad kravene i avsnittet "Oppgave" er oppfyllt.

Målet med denne oppgaven er å tilby pålitelighet i nettverkslaget for overvåknings / administrasjonsverktøyet presentert i "hjemmeeksamen 1" som tillater en administrator å bruke en klient for å fjernovervåke alle prosesser som kjører på et cluster av Linuxmaskiner.

UDP kan lide av dårlig ytelse på grunn av store pakketap ettersom UDP er en upålitelig protokoll, og garanterer derfor ikke mottak av pakker. I denne oppgaven skal du implementere en Go-Back-N kommunikasjonsprotokoll som legger til pålitelighet i kommunikasjon over UDP sockets.

Merk: Løsningen må være skrevet i programmeringsspråket C. Denne oppgaven kan være en utvidelse av "hjemmeeksamen 1" eller "obligatorisk oppgave".

Oppgave

Først, endre "hjemmeeksamen 1" på en slik måte at all kommunikasjon bruker UDP sockets (inkludert både kommandoer og streaming data). Som i "hjemmeeksamen 1", er alle overførte data innkapslet i 100-byte meldinger (pakker). Hvis pakker ankommer i feil rekkefølge, må de sorteres og vises i riktig format. Dette gjelder også "kommandoer", hvor out-of-order ankomst av pakker kan føre til en feil ved kjøring av kommandoen.

For å tilføre pålitelighet, må du implementere en Go-Back-N protokoll som fungerer på følgende måte: når en pakke er mottatt av mottakeren (for eksempel når UDP streaming-pakker som inneholder statistiske data av prosesser kommer til klientmaskinen) sendes en bekreftelsespakke tilbake til avsenderen. Denne pakken inneholder et bekreftelsesnummer (ACK) som indikerer at mottakeren har mottatt alle datapakker før dette nummeret, i en sekvens.

Avsenderen holder utgående pakker i en avsenderbuffer. Størrelsen på denne bufferen er fleksibel i programmet (bruk en standard buffer størrelse på 10 pakker). Når en ACK pakke ankommer og indikerer mottaket av pakker opp til et visst sekvensnummer, sletter avsenderen pakkene opp til det sekvensnummeret fra sin avsenderbuffer.

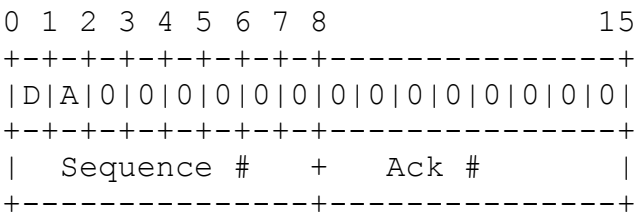
Hver gang en pakke blir sendt, setter avsenderen en timer for sending av pakken på nytt til en fleksibel verdi (angitt nedenfor) hvis det ikke allerede er satt for en annen pakke. Etter sending av pakken fortsetter avsenderen med å sende nyere pakker i bufferen sin, om det er noen. Hvis avsenderen ikke mottar en kvittering for at en enkel pakke er mottatt før timeren utløper, vil den prøve å overføre denne pakken på nytt, samt alle pakker som allerede er sendt med høyere sekvensnumre, EN GANG. Deretter tømmer den sin avsenderbuffer og setter timeren igjen for neste ubekreftede pakke. Retransmitteringen av pakker (bare en gang) forbedrer påliteligheten til systemet. Det at man kun retransmitterer en gang sikrer at forsinkelsen ikke vokser for mye.

Eksempel: La oss anta at pakker 1, 2, 3, 4, 5, 6, 7, 8, 9, er 10 sendt og retransmisjonstimer er satt for pakke 5. Hvis bekreftelsespakker ankommer hos avsenderen i sekvensen 1, 2, 3, 4, 6 (ackno = 4), 7 (ackno = 4) ... , vil avsenderen slette pakkene 1-4 fra avsenderbufferen sin ettersom de er allerede "ACKed", og deretter vente på ankomsten av ACK-5 mens den sender pakkene 11,12,13 som fortsatt er i avsenderbuffer. Dersom retransmisjonstimeren utløper (dvs. ACK-5 ikke kommer frem), etter f.eks 600 millisekunder, retransmitterer avsenderen alle pakker 5-13 og sletter dem fra bufferen sin. Det skjer bare en gang: hvis ACK-5 ikke kommer igjen, ignorerer bare avsenderen dette problemet og går videre. Hvis ACK-5 er mottatt, setter avsenderen retransmisjonstimeren for neste ubekreftede pakke, f.eks ACK-11. Når en ACK for denne pakken er mottatt, starter retransmisjonstimeren på nytt.

Du vil trenge å opprettholde noen verdier og definere en del parametere:

Retransmisjonstimer: Senderen har en retransmisjonstimer med en konfigurerbar time-out verdi. Denne timeren skal initieres hver gang pakken er sent og det ikke for øyeblikket er en pakke som blir tatt tiden på. Hvis det ikke blir mottatt en bekreftelse (ACK) for denne pakken innen tiden går ut, skal alle pakker som er sendt bli sendt på nytt. Retransmisjonstimeren skal startes på nytt når pakken vi tar tiden på kommer frem. Den anbefalte verdien på retransmisjonstimeren er 600 millisekunder.

Pakkeheadere: Formatet på pakkeheaderene for pålitelige UDP og ACK pakker skal defineres på følgende måte:



Reliable-UDP header

Kontroll bits: Kontrollbittene i pakkeheaderen indikerer hva som er tilstede i pakken. ACK bittet (A) indikerer at acknowledgementnummeret i headeren er gyldig (mao. at pakken er en bekreftelse/acknowledgment). D-bittet indikerer at pakken faktisk bærer data.

Til slutt, når en data pakke er mottatt, skal datasegmentet av pakken hentes ut og brukes riktig i forbindelse med utskift av prosesstatistikk eller utføring av kommandoer.

Merk #1: For å vise noe meningsfulle resultater, **bør** du øke tidsintervallet på utføringen av «prosestatistikk» til en høyere verdi enn det som ble anbefalt i hjemmeeksamen 1, f.eks. 2-3 sekunder. Å bruke en liten verdi kan føre til at utdaterte pakker blir retransmittert når de ikke er brukbare lenger.

Merk #2: Om det skulle dukke opp duplikater av en pakke, kan denne ignoreres av mottakeren så den ikke blir vist eller brukt flere ganger. F.eks. når den originale pakken kommer fram hos mottakeren mens retransmisjonen allerede har startet. I dette tilfellet blir pakken mottatt to ganger.

Merk #3: For å teste programmet ditt, ordne en tilfeldig pakketap "protokoll" på en hvilken som helst måte du foretrekker (f.eks. Ved bruk av Linux kommandoen tc) og sjekk om de tapte pakkene blir retransmittert eller ikke. Sjekk også om kommandoene blir utført riktig og statistikk blir vist ordentlig i en situasjon der pakketap oppstår.

Spørsmål: Protokollen som er nevnt over er en enkel måte å sikre pålitelighet for UDP. Tenk på hvilken svakhet denne protokollen har og foreslå en måte å forbedre effektiviteten til denne.

Besvarelse

Dere skal levere følgende:

1) Et design dokument som inneholder:
- En frontside med kandidatnummer, oppgavetittel, kurs og semester, vi vil ikke ha navn eller brukernavn.
- Hvordan programmet er designet. Gjerne med en tegning (flytdiagram) som viser hvilken rekkefølge de forskjellige funksjonene blir kalt.
- En dokumentasjon av hvordan programmet skal startes evt. avsluttes.
- Hvilke filer programmet består av (C filer, headerfiler osv.).

2) Programfilene, hvor koden er fyldig kommentert. Dokumenter alle variable og definisjoner. For hver funksjon i programmet skal følgende dokumenteres:
- Hva funksjonen gjør
- Hva inn og ut parametre betyr og brukes til
- Hvilke globale variable funksjonen endrer
- Hva funksjonen returnerer

Levering

Designdokumentet skal skrives vha. et egnet verktøy, for eksempel LaTeX, Word, etc. Dokumentet skal inneholde besvarelsen og de etterspurte figurer, samt ha en forside hvor følgende opplysninger er angitt: kandidatnummer, oppgavetittel, kurs og semester. Før levering skal dokumentet konverteres til PDF format. Omfanget av dokumentet trenger ikke nødvendigvis være så stort, men må inneholde tilstrekkelig informasjon til å oppfylle kravet som beskrevet under avsnittet oppgave. Vi stiller krav til ryddighet og struktur.

Elektronisk innlevering: Alt skal leveres elektronisk hvor alle filer (Makefile, *.c, *.h, README.pdf, etc.) er samlet i en katalog med kandidatnummeret som navn. Av denne katalogen lager du en komprimert TAR-ball (ikke ZIP eller RAR). Bruk kommandoen tar --owner=root --group=root -zcvf knr.tgz knr hvor knr er ditt kandidatnummer. Den elektroniske innleveringen skal leveres via web. Linken finnes på kursets hjemmeside.

Innleveringsfrist: Onsdag 18. mai 2010, klokken 23:59

Merk at denne tidsfirsten er HARD, oppgaver levert etter fristen vurderes med karakteren "F", altså stryk. Det forutsettes at studenten har lest forskriften om studier og eksamener ved Universitetet i Oslo, karaktergivende oppgave. Sykemelding med legeerklæring leveres til studieadministrasjonen og fører til at eksamen ikke teller
i sluttresultatet.

INF3190 - Home Exam 2 (Description in English)

The goal of this exercise is to provide network layer reliability for the monitoring/administration tool presented in “home exam 1” which allows an administrator to remotely use a client to monitor all processes running on a cluster of Linux machines. UDP may suffer from poor performance in the presence of heavy loss since it is an unreliable protocol and does not guarantee the receipt of the packets. In this assignment, you are supposed to implement a Go-Back-N communication protocol that adds reliability to the communication over UDP sockets.

Note: The solution must be written in the C programming language. This assignment can be an extension of either "home exam 1" or "mandatory assignment" .

Task

First, modify the “home exam 1” in a way that all communication is carried out with UDP sockets (including both commands and streaming data). Similar to the “home exam 1”, all transferred data are encapsulated in 100-bytes messages (packets). If packets arrive out of order, they must be reordered and shown in the correct format. This also applies to the “commands”, where out-of-order arrival of packets could result in an execution failure.

To provide reliability, you need to implement a Go-Back-N protocol which works in the following way. Once a packet is received at the receiver side (e.g. UDP streaming packets containing statistical data of processes arriving at the client machine) an *acknowledgement* packet is sent back to the sender. This packet contains an *acknowledgement number* indicating that the receiver has received all data packets preceding it in a sequence.

The sender keeps the outgoing packets in a sending buffer. The size of this buffer is tunable in the program (use a default buffer size of 10 packets). When an ACK packet arrives indicating the receipt of the packets up to a certain sequence number, the sender erases the packets up to that sequence number from its sending buffer.

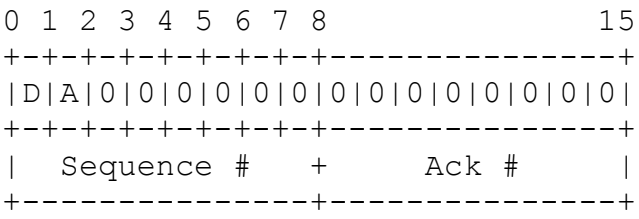
Every time a packet is sent, the sender sets a retransmission timer to a tunable value (specified below) if it is not already set for another packet. After sending that packet, the sender continues to send newer packets in its buffer if there are any. If the sender does not receive an acknowledgment for that certain packet until the retransmission timer expires, it will try to retransmit that packet and all packets already transmitted with higher sequence numbers only ONCE; then, it flushes its sending buffer and sets the retransmission timer again for the next unacknowledged packet. Retransmitting packets only once somewhat improves the reliability of the system while ensuring that the delay does not grow too much.

Example: Let’s assume that packets 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 are sent and the retransmission timer is set for packet 5. If the *acknowledgement* packets arrive at the sender in the sequence of 1, 2, 3, 4, 6 (ackno=4), 7 (ackno=4) ... , the sender will erase the packets 1 to 4 from its buffer since they are already ACKed, and it will wait for the arrival of ACK-5 while sending the packets 11,12,13 that are still in its buffer. If the retransmission timer expires (i.e. ACK-5 does not arrive), after e.g. 600ms, the sender retransmits all packets from 5 to 13 and erases them from its buffer. That happens only ONCE: if ACK-5 does not arrive again, the sender simply ignores this issue and moves on. If ACK-5 is received, then the sender sets the retransmission timer for the next unacknowledged packet, e.g. ACK-11. Once an ACK for that packet is received, the retransmission timer is restarted.

You will need to maintain some values and define some parameters:

Retransmission timer: The sender has a retransmission timer with a configurable time-out value. This timer is initialized every time that a packet is sent and there is not a packet currently being timed. If no acknowledgement for this data packet is received before the timer expires, all packet that have been sent are retransmitted. The retransmission timer is restarted when the timed packet is received. The recommended value of the retransmission timer is 600 milliseconds.

Packet headers: The packet header format for reliable-UDP and ACK packets should be defined as follows:



Reliable-UDP header

Control bits: The control bits in the packet headers indicate what is present in the packet. The ACK bit (A) indicates that the acknowledgement number in the header is valid (i.e. the packet is an acknowledgement). The D bit indicates the packet is carrying data.

Finally, upon receipt of the data packets, the data section should be extracted from the packet and used properly on demonstration of process stats or executing the commands.

Notice #1: To have a meaningful result, you **should** increase the "process statistics" execution (e.g. ps aux) time interval to a larger value than what is used in "home exam 1", e.g. 2-3 seconds. Using a small value may lead the outdated packets to be retransmitted while they are not useful anymore.

Notice #2: In case of duplicate packets they should be ignored at the receiver and not be displayed or used e.g. when original packets arrive at the receiver while the retransmission has started. In that case, the packets will be received twice.

Notice #3: To test your program, generate random packet loss in any way you prefer (e.g., using the Linux *tc* command) and check if the lost messages are retransmitted or not. Also check if commands are executed successfully and statistics are displayed properly in the presence of loss.

Question: The above protocol is a simple way of providing reliability for UDP. Think about the weaknesses of this protocol and propose some idea on how to improve its efficiency.