Hjemmeeksamen – INF1060 – høst 2017

I denne oppgaven skal du bruke det du har lært til nå og gjort i oblig 1 og 2, og kombinere det med nettverksprogrammering og IPC (Inter Process Communication). Vi skal med andre ord jobbe med filer, structer, strenger, pekere, systemkall, nettverk, IPC og mer!

Oppgavene skal løses **individuelt**, se ellers forskrift om studier og eksamen. Du vil finne nyttige ukesoppgaver med tilhørende forklaringer av viktige begreper på gruppelærersiden, og som i de fleste fag er semestersiden stedet for forelesningsfoiler og annen informasjon du kanskje er på utkikk etter. Du kan også finne noen relevante eksempelprogrammer på INF1060-githuben.

Dersom du har spørsmål underveis kan du oppsøke en orakeltime eller stille spørsmål på Piazza. Det vil (selvfølgelig) være mye relevant informasjon i plenumstime.

Husk å lese hele oppgaveteksten, så du vet hvor mye arbeid du har igjen totalt sett. *Du vil til slutt i dokumentet finne en liste som viser hva du bør prioritere når du jobber med oppgaven.*

Oppgavene blir testet på Linux på Ifi sine maskiner eller tilsvarende.

Innlevering i Devilry innen tirsag 14.11 23:59. Denne tidsfristen er **hard**, leveringer etter dette blir vurdert som **F**.

Denne hjemmeeksamenen teller 40% av endelig karakter i INF1060 og må bestås for å kunne få endelig karakter.

Oppgaven

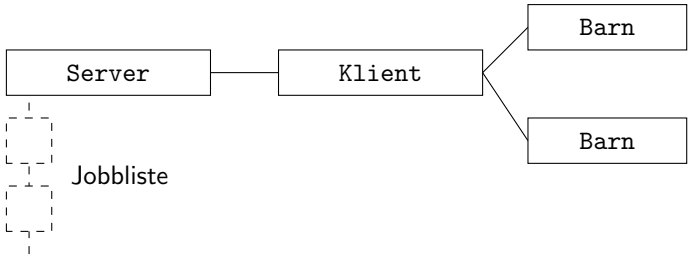
I denne oppgaven skal du programmere en klient og en server som skal snakke med hverandre. Serveren har en del jobber den vil ha utført, og klienten skal utføre disse jobbene. På server-siden ligger det en liste med jobber i en joblist-fil, og jobbene i den skal oversendes til klienten for utførelse. Serveren og klienten skal kommunisere over **TCP**. Forholdet mellom prosessene er illustrert i Figur 1.

Programmene skal kompileres med make (**du må altså lage en makefile**).

Du finner flere joblist-filer du kan teste programmet ditt med i Hjemmeeksamen-repositoriet på github

Spesifikasjoner

Videre følger spesifikk informasjon om hvordan programmene skal fungere, inndelt i en seksjon om klienten og en om serveren. Noe informasjon om klienten vil stå i seksjonen om serveren, og vice versa, så les nøye. Det som er beskrevet som **Bonus**-deler av oppgaven, gir ekstra poeng. Ved å løse disse delene av oppgaven kan du hente inn noen poeng ekstra, men det er ikke mulig å få mer enn 100% uttelling totalt sett.



Figur 1: Arkitektur

Klient

Klienten skal starte to barneprosesser med `fork` for å utføre jobber. Klienten og barneprosessene dens skal kommunisere med *pipes*. Klienten skal deretter koble seg på serveren med kall på `socket` og `connect`. Barneprosessene skal altså ikke være koblet til serveren. Når klienten mottar meldinger fra serveren skal disse behandles, se Tabell 1 lenger nede og seksjonen om jobbtyper og jobbinfo for hva klienten og barneprosessene skal gjøre.

Etter at pipene er satt opp, barneprosessene er forket ut, og klienten har koblet seg på serveren, skal klienten gi brukeren (minst) fire valgmuligheter:

- Hent én jobb fra serveren
- Hent X antall jobber fra serveren (spør bruker om antall)
- Hent alle jobber fra serveren
- Avslutte programmet

Du kan legge til flere valgmuligheter hvis du synes det er hensiktsmessig.

Kommunikasjon med server

Klienten sender meldinger til serveren bestående av maksimalt 4 byte. Hvordan du velger å bruke de 4 bytene er opp til deg. Klienten må kunne sende minst de følgende type meldinger til serveren:

- En melding som henter én jobb.
- En melding som varsler serveren om at klienten terminerer normalt.
- En melding som varsler serveren om at klienten terminerer på grunn av en feil.

Du skal altså selv bestemme hvordan du bruker de fire bytene du har til rådighet til å formidle disse meldingene. Du kan legge til andre meldinger hvis du synes det er fornuftig. Hvordan du har designet denne delen av protokollen, inkludert eventuelle andre meldinger du har lagt til, må du beskrive i en fil, **protokoll.txt**, som du også må levere inn. Beskrivelsen skal være ryddig og forståelig, og i protokollen skal bruken av de 4 bytene som er tilgjengelig skal være fornuftig og konsekvent.

Kjøring

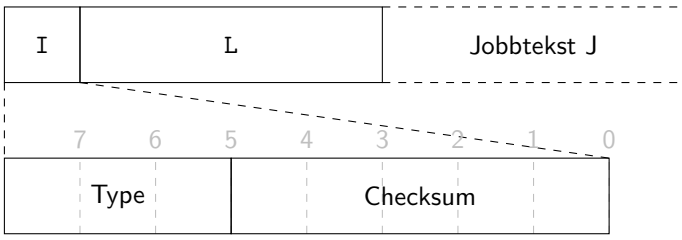
Klienten startes med (minst) følgende argumenter:

- Adresse – (IP-) adressen der serveren kjører
- Port – Porten serveren lytter på

Serveren må være startet først på den angitte adressen med samme port-nummer for at klienten skal lykkes med å koble seg opp. For eksempel startes klienten slik:

```
$> ./klient 127.0.0.1 4323
```

Du kan legge til ytterligere argumenter om du trenger det, men sørg for å dokumentere det så rettere vet hvordan de skal starte programmet. **Bonus:** gjør det mulig å koble seg opp med maskinnavn i stedet for IP-adresse, f.eks. `nordur.ifi.uio.no`.



Figur 2: Formatet på en melding fra serveren. Øverst hele meldingen, nederst er `I`-byten forstørret.

Server

Serveren skal åpne en joblist-fil for lesing (se egen seksjon om filstruktur), og sette opp en *socket* ved hjelp av kall på `socket`, `bind` og `listen`, og akspetere tilkobling fra klienten med `accept`. Serveren skal kun håndtere én klient, og **avvise tilkoblingsforsøk** fra andre enn den første klienten. Klienter som blir avvist skal ikke henge eller krasje, men avslutte med en feilmelding til bruker.

Når serveren mottar en melding fra klienten som spør om en ny jobb, skal serveren svare med en jobb fra joblist-filen den har åpnet. En jobb skal leses fra filen først når klienten spør om en jobb, og serveren skal da sende en melding til klienten. Altså skal *ikke* hele filen leses inn i minnet fra starten av programmet.

Feltene i meldingen som serveren skal sende til klienten, illustrert også i Figur 2:

- `I` – Jobbinfo, `unsigned char`
- `L` – Tekstlengde, `unsigned int`
- Jobbtekst – `char[]`

Jobbinfo-charen (byten) er forsøkt illustrert i nederste del av Figur 2. Her representerer de tre bitene i posisjon 5 til 7 jobbtypen (referer til Tabell 1), og bitene i posision 0 til 4 representerer en checksum. Denne checksummen skal regnes ut etter at en jobb er lest fra filen (på serversiden), og igjen regnes ut når meldingen er mottatt av klienten. Mer om checksummen står i egen seksjon.

Serveren må kunne håndtere alle meldinger klienten sender, som definert i din egen protokoll. Når det ikke er flere jobber igjen i jobliste-filen skal serveren sende en melding med jobbtype `'Q'` og tekstlengde og checksum lik 0, for å fortelle klienten at det ikke er flere jobber. Deretter skal serveren vente på at klienten bekrefter at den avslutter, før serveren selv avsluttes.

Checksum

Checksummen regens ut på følgende måte:

1. Summer alle karakterene i jobbteksten (J), altså summer verdiene, byte for byte.
2. Beregn modulo 32 på denne summen.

$$S_c = \sum_{i=0}^{L-1} J_i \ \% \ 32 \tag{1}$$

Dette er også vist i Ligning (1). I ligningen er S_c cheksumen, L lengden på jobbteksten, og J_i er byten på plass i i jobbteksten J. og Resultatet er et tall mellom 0 og 31 (inklusivt). Dette tallet skal legges inn i de 5 bitene på poisjon 0 til 4, som illustrert i Figur 2.

Figur 3: Formatet på en jobb i en jobbfil.

Jobbtype	bitmønster	Sendes til barneprosess nr	Jobbbeskrivelse
`'O'`	000	1	Jobbteksten skal printes til standard out (`stdout`).
`'E'`	001	2	Jobbteksten skal printes til standard error (`stderr`).
`'Q'`	111	1 og 2	Barneprosessene skal terminere

Tabell 1: Jobbtyper

Dersom checksummen i meldingen fra serveren ikke stemmer med den checksummen klienten selv regner ut når meldingen mottas, skal jobben ignoreres, og ikke sendes til en barneprosess.

Filstruktur

Hver av jobbene i filen ser ut som følger:

- Jobbtype – `T` – `char`
- Tekstlengde – `L` – `unsigned int`
- Jobbtekst – `char[]`

Lengden på jobbtekst-arrayet er gitt i tekstlengde-feltet. **Denne teksten er ikke nullterminert.** Merk at jobbene i filen ikke er adskilt med linjeskift, null-byte eller noen annen separator. Figur 3 illustrerer strukturen til en jobb i filen.

Jobbinfo og jobbtyper

Tabell 1 viser de forskjellige jobbtypene en server kan sende til sin klient. Den første kollonnen, Jobbtype, er den som står i filen. Bitmønsteret sier hvordan denne jobbtypen skal representeres i de tre bitene som er på posisjon 5 til 7 i jobbinfo-feltet i meldingen serveren sender, som nevnt over. De resterende kolonnene sier hva klienten skal gjøre når den mottar jobber av de ulike typene.

Klienten skal altså sende jobbteksten videre til en av barneprosessene, for jobbtyper `'O'` og `'E'`. Hvordan denne kommunikasjonen mellom klienten og barneprosessene foregår er det opp til deg å definere. Tenk også på hvordan formidle melding med type `'Q'` til barneprosessene. Barneprosessene har ingen oppgaver andre enn å printe den teksten de får.

Kjøring

Serveren startes med (minst) følgende argumenter:

- Filnavn – navnet på en fil som inneholder en liste med jobber som leses og sendes til klienten på forespørsel.
- Port – porten som serveren skal lytte på.

For eksempel kan man starte serveren på denne måten:

```
$> ./server alice_short.job 4323
```

Du kan legge til ytterligere argumenter om du trenger det, men sørg for å dokumentere det så rettere vet hvordan de skal starte programmet.

Feilhåndtering

Programmene bør ikke krasje under noen omstendighet. Returverdier fra system-/funksjonskall bør sjekkes og feil bør håndteres på en god, oversiktlig og brukevennlig måte.

Programmene bør testes med `valgrind`. Eventuelle feil som valgrind påpeker bør utbedres. Dette blir brukt som en del av vurderingsgrunnlaget.

Debug-output

Både serveren og klienten skal kunne startes i debug-modus. I debug-modus skal programmene printe hva de gjør, hva som skjer, og annen relevant informasjon. Informasjonen skal være lettlest og formattert på en måte som gjør den brukbar. Alle meldinger skal starte med teksten `>>> PID <<<`, der `PID` er prosess-IDen (PIDen) til prosessen som printer. Minimum følgende hendelser må omtales i debug-outputen:

- Oppstart og avslutning
- IP-addresser/porter i bruk
- Meldinger inn/ut av sockets/pipes
- Lesing fra fil

For maksimal uttelling på dette punktet er det nødvendig med mer enn det som er nevnt i listen.

Avlutning

Når programmene avslutter skal alle allokerte minneområder (allokert med `malloc`) være frigjort ved kall på `free`. Sockets skal stenges i en rekkefølge som gjør at programmene ikke krasjer eller henger. Pipene som klienten og barneprosessene bruker skal lukkes. **Obs!** Husk på å frigjøre/lukke i barneprosesser. Pass også på at du ikke får zombie-prosesser (zombie-prosesser er barneprosesser som lever etter at foreldreprosessen har avsluttes, som aldri avslutter av seg selv).

Ctrl+C (sigint)

Dette anbefales det å gjøre først når resten av programmet fungerer.

Sett opp en signal-handler som gjør at brukeren også kan trykke `Ctrl-C` for å terminere server og/eller klient, uten at dette fører til minnelekasjer eller at det andre programmet fryser/krasjer.

Dokumentasjon

Koden du leverer skal være godt dokumentert. Foran hver metode du lager skal det være en kommentarblokk som inneholder en beskrivelse av hva funksjonen gjør, hva input-parametrene betyr og hva funksjonen returnerer. For eksempel:

```
/* This function takes two integer arguments, adds them
 * together and multiplies the result by 2. Only works if
 * the result of the addition is 0 or above.
 *
 * Input:
 *      a: the first int
 *      b: the second int
 *
 * Return:
 * The result of the calculation, or -1 if the result of the
 * addition was negative.
 */
int add_and_double_result(int a, int b) {
    return a+b >= 0 ? (a+b) << 1 : -1;
}
```

Selv om koden inne i funksjonen er litt kryptisk er det lett å forstå hva funksjonen gjør fordi den er godt dokumentert!

Viktighet av de forskjellige funksjonalitetene

Her er en liste over viktigheten av de forskjellige delene av oppgaven. Bruk listen som en guide for hva du bør jobbe med (og i hvilken rekkefølge). Dette vil bli brukt ved retting.

1. Fungerende kommunikasjon mellom server og klient.
   Herunder utregning av checksummer og ditt eget protokolldesign.
2. Fungerende kommunikasjon mellom klient og barneprosesser.
3. God og korrekt bruk av minne (heap og stack) (`free`).
4. God programstruktur (filer, funksjoner).
5. Debug-mode
6. Godt dokumentert kode

Levering

1. Lag en mappe med ditt kandidatnr: `mkdir 12345`
2. Kopier alle filene som er en del av innleveringen inn i mappen:
   `cp *.c 12345/` (f.eks.)
3. Komprimer og pakk inn mappen:
   `tar -czvf 12345.tgz 12345/`
4. Logg inn på Devilry
5. Lever under INF1060 Hjemmeeksamen

Kandidatnummer finner du på studentweb.

Relevante man-sider

Disse man-sidene inneholder informasjon om funksjoner som kan være relevante for løsing av denne oppgaven. Merk at flere av man-sidene inneholder informasjon om flere funksjoner på én side, som `malloc/calloc/realloc/free`. Nummeret foran hver funksjon er hvilken seksjon i manualen siden ligger i. Får å få informasjon om f. eks. `read`, skriv `man 2 read`.

- 3 `malloc/calloc/realloc/free`
- 3 `fgets/fgetc/getchar`
- 3 `fread/fwrite`
- 3 `fopen/fclose`
- 3 `scanf/fscanf`
- 2 `read`
- 2 `write`
- 2 `socket`
- 2 `bind`
- 2 `listen`
- 2 `accept`
- 2 `connect`
- 2 `signal`
- 2 `pipe`
- 3 `strcpy`
- 3 `memcpy`
- 3 `memmove`
- 3 `atoi`
- 3 `strtol`
- 3 `isspace/isdigit/alnum` m.fl.
- 3 `strdup`
- 3 `strlen`
- 3 `printf/fprintf`