INF1060 Introduksjon til operativsystem og datakommunikasjon

Dette er et oppgavesett for trening

September 24, 2012

Simpel Ifi-epost system (lokal) og diverse oppgaver

Denne oppgaven går ut på å lage et enkelt e-postsystem hvor dere skal sende meldinger til hverandre lokalt dvs at dere ikke skal sende til ordentlige epost-systemer (som hotmail, gmail osv... med mindre dere vil gjøre dette så vil vi ikke stå i veien for det :-)). Før den harde oppgaven, skal vi gjøre et par C funksjonaliteter for trening slik at dere er forberedt og har nok kunnskap for å gjøre epostsystemet.

Begynn med det enkle først, også gå gradvis oppover..

Del 1 C generelt

Makefile

Denne oppgaven går ut på å lage Makefile med flere funksjonaliteter, den skal både gjøre slik at man kan kompilere flere C filer og ha en som fjerner objektfiler (.o filene) Gjør først om Makefilen din at den kaller på ls, ls -l,. Hvorfor har vi Makefile? Er det noe vits?

Lag et enkelt program som bruker funksjonene til et annet program, ta for eksempel helloworld print på en C fil og la den andre benytte seg av funksjonen (kreves header i tillegg evt og må kompilere 2 C filer, forenkle linja oppgitt: gcc -g -Wall main.c helloworld.c -o test)

Liste med shelloop

Dere skal få til et epostsystem som det vanskeligste oppgaven i settet(dette gjøres på slutten, før dette er det trening på C programmering generelt), men først og fremst gjør vi det enkelt, det dere skal først lage er et lenkaliste (Queue) med shell-løkke som gjør enkle funksjonaliteter.

shellinja kan f.eks se ut slik:

```
brukernavn:[tt:mm:ss]>
```

HINT: For å finne ut brukernavnet kan det være lurt å se på miljøvariablene (environment variables) og bruke funksjonen getenv for å hente verdiene inn i programmet (se man *getenv*). (Du lister alle miljøvariablene i shellet med kommandoen env. Unngå å bruke variabelen 'USERNAME' da den kun settes hvis man sitter fysisk på maskinen, ikke når man fjerninnlogger med ssh). For å finne tiden er gettimeofday en mulighet (se *man gettimeofday*). (denne kan gjenbrukes til siste oppgave om epostsystemet for klienten).

Lese en linje fra standard-inn (tastaturet). Vi kan anta at linjen aldri er lengre enn 200 tegn. HINT: Bruk *fgets* til å lese linjen.

define?

Ok så fleste av dere har sett disse #**define** på nettet eller programmer, hvorfor trenger vi dette? Eller spesifikt hvorfor har vi bruk for det?

Lag et program som da skal definere å sammenlikne to elementer, og en som finner maxverdi av data.

Del 2 Pekere

Pekere generelt er en variabel som inneholder minneadresseinformasjon. Et normalt variabel inneholder spesifikke verdier mens en peker inneholder adresse til variabelen dvs en variabel navn referer direkte til en verdi og peker refererer indirekte til en verdi.(Referere en verdi ved peker er kjent som indirekte).

Arrays og pekere

Du har følgende kode:

```
void function( int a, int* b, char c, char* d, char e[] ) {
        a = 2; *b = 3;
        b = &a;
        c = 'x';
        *d = 'y';
        d = &c;
        e[2] = 'z';
}


int main( ) {
        int k,l;
        int* m;
        char n,o;
        char* p;
```

```
char q[5];

k = 100;
l = 101;
m = &l;
n = 'n';
o = 'o';
p = &o;

q[0] = 'h';
q[1] = 'e';
q[2] = 'r';
q[3] = 'e';
q[4] = '\0';

function( k,m,n,p,q );
printf( "k=%d l=%d *m=%d\n", k, l, *m );
printf( "n=%c o=%c *p=%c\n", n, o, *p );
printf( "q=%s\n", q ); }
```

Hva skrives ut på skjermen? Hvorfor skrives ut akkurat dette?

Lek med arrays og pekere

Arrays og pekere er ofte relatert til hverandre i C programmering og er ofte brukt sammen. Et array navn er oftest en konstant peker, pekere kan bli brukt for å gjøre hvilke operasjoner som er involvert i en array.

Anta at en integer array *b[5]* og integer peker variabel *bPtr* har blitt definert. Siden et array navn er en peker til første elementet til array, så kan vi sette *bPtr* til adressen til det første elementet i arrayet b f.eks

```
bPtr = b;
```

Denne operasjonen er det samme som å ta adressen til det første elementet i arrayet slik:

```
bPtr = &b[0];
```

Array element b[3] can alternativt bli referert med en peker uttrykk

```
*(bPtr + 3);
```

For å referere adressen til array:

```
&b[3];
```

kan bli skrevet med pekeruttrykk:

```
bPtr+3
```

Arrayet kan i tillegg bli behandlet som peker og brukt som peker aritmetikk for eksempel:

```
*(b+3)
```

som vil referere til array elementet b[3].

La oss lage et program som da tar imot peker og array strenger og kopierer dataen til hverandre, dvs

```
void copy1(char *const s1, char *const s2);
void copy2(char *s1, char *s2);
```

copy1() skal kopiere fra peker til array, mens copy2() skal gjøre det motsatte.

Del 4 Strenglek

Nå skal vi se litt mer på strenger i C,

Strenger i C er ikke noe annet en bokstavarrayer. Det er blant annet ikke mulig å utføre veldig nyttige operasjoner på C strenger som de allerfleste andre språk har. For eksempel kan man ikke bruke plus operatoren for å konkatinere to C strenger. Mange misliker det.

Den store fordelen med C strenger er at man har full kontroll over det som skjer. Man hvet nøyaktig hvor i minne dataene til en C streng ligger, hvor mye plass den bruker, og hva man må sende over nettet for at mottakersiden også kan bruke den som en streng. Det klarer man ikke med strengene i de andre språkene. Og man kan f.eks. trekke ut eller erstatte enkelte bokstaver, regne med bokstaver som om de var tall eller på en enkel måte avbryte strenger. Men det krever litt trening.

Du har følgende kode:

```
void funA( char buf[10] )          {
    int i;

        for( i=0; i<10; i++ )          {
          buf[i] = 'a' + i;
        }
}

void funB( char* buf )             {
   int i;
        for( i=0; i<10; i++ )          {
                buf[i] = 'b' + i;
         }
}
void funC( char* buf )     {
        int i;
        for( i=0; i<10; i++ )          {
```

```
                *buf = 'c' + i;
                buf = buf + 1;
        }
}

int main( )             {
    char buf[11];
    memset( buf, 0, 11 );
    funA( buf );
    printf( "%s\n", buf );
    funB( buf );
    printf( "%s\n", buf );
    funC( buf );
    printf( "%s\n", buf );
 }
```

1. Hva skreves ut på skjermen? Hvorfor skreves ut akkurat dette? (gjerne kopier og lim inn og se hva resultatet blir, endre og lek med det, gjerne tegn og se for dere hva som egentlig foregår.

2. Du har den følgende variablen:

```
char* apple = "                          a\n"
              "                         ppl\n"
              "                       eappl\n"
              "                      eapple\n"
              "                      apple\n"
              "                      appl\n"
              "          eappleappleappleappl\n"
              "      eappleappleappleappleapple\n"
              "    appleappleappleappleappleapple\n"
              "  appleappleappleappleappleappleappl\n"
              " eappleappleappleappleappleappleapple\n"
              "appleappleappleappleappleappleappleapp\n"
              "leappleappleappleappleappleappleapplea\n"
              "ppleappleappleappleappleappleappleappl\n"
              "eappleappleappleappleappleappleappleap\n"
              "pleappleappleappleappleappleappleapple\n"
              "appleappleappleappleappleappleappleapp\n"
              "leappleappleappleappwormpleappleapplea\n"
              " ppleappleappleappleappleappleappleap\n"
              "  pleappleappleappleappleappleapplea\n"
              "   ppleappleappleappleappleappleapp\n"
              "    leappleappleappleappleappleap\n"
```

```
"        pleappleappleappleappleapp\n"
 "          leappleappleappleapple\n"
 "            appleappleappleapp\n";
```

Dette er en lovlig C streng. Spørsmålene er: Hvor langt er strengen? Hvor mange bytes må du allokkere med malloc hvis du ønsker å lage en kopi?

Lek med eplet

Finn ormen i eplen. Forsøk følgende muligheter:

Bruk ikke noen C funksjoner i det hele tatt, bare enkle språkoperasjoner som if/else, for, while, do osv. Bruk søkefunksjonen strchr().

Mer lek med eplet

Husk å inkludere string.h. Bruk erstatningsfunksjonen strstr().Klipp ormen ut av eplen, med å overskrive bokstavene med " ".(første oppgaven)

Hvor dypt i eplen var ormen (Hva er index verdien for den første bokstaven 'w' av ordet worm). Brukt funksjonen strstr() for å finne svaret.(andre oppgaven)

Del 5 Filskriving og lesing

Simpel funksjonalitet

Noen ganger kan det være lurt å ha backup dersom f.eks server eller store maskiner kræsjer, La oss lage et program som skriver data og leser fra data.

Vi vil gjerne opprette en struct som inneholder klientens kontonummer, fornavn, etternavn og pengene(greit å ha kontoid i tillegg for å finne riktig data på filen). Opprett en backup.bak og dersom det eksisterer så leser vi det (sjekk *rb+* notasjon). Vi skal først skrive datainfo ved hjelp av *scanf()* og *fscanf() med stdin* på den sistnevnte, hvor vi lagrer klientens datainnhold. deretter må vi bruke en funksjon for å søke posisjonen i filen for brukerspesifisert post og skriver da selveste dataen inn i filen. Deretter går du gjennom filen og skriver ut struct verdien for alle dataene og sjekker om du har klart å skrive inn structen.

HINT! Se på *fseek()*, *fwrite()* og *fread()* funksjonene.

Bankkonto program (vanskelig)

Med hensyn fra forrige oppgave, lag et C program med samme funksjonaliteter som du har gjort tidligere, men denne gangen vil man gjerne ha en shelløkke med følgende funksjonaliteter:

- opprett tekstfil
- oppdaterposter

- ny post
- slett post

Dere må oppgi riktig kontoid slik at *fseek()* kan finne det riktige område for å gi deg den posisjonen hvor dataen ligger. Gjerne lag meny med *scanf() eller bruk det du gjorde i Del1, men se gjerne litt mer på strtok().*

.Del 6 datastruktur

Ring buffer

I delte resursser genrelt har man i enkelttilfeller et ring buffer for å gi et statisk plass i minnet hvor man da lagrer data modulo av ringbufferstørrelsen. Implementer dette og lag et enkelt test hvor man da lagrer 10 data i ringbufferet og skriver ut dette, deretter legg til 1 data til og skriv ut, hvor overskriver dette? Pros and cons?

Binærtre(middels vanskelig)

Liste er vel ikke veldig optimalt, prøv å gjør et binærtre og sørg for at den kan legge inn, søke og slette data.

Prioritetskø(vanskelig)

Ok så la oss lage et typisk "printer" hvor det logiske ville vært en vanlig kø liste, men la oss tenke at hvis det er viktige dokumenter som burde skrives ut så hadde det vært fint om man prioriterte dette. La oss lage et prioritetskø som vil prioritere filer for å skrive ut(Vi skal ikke skrive ut, men hver Node vil da inneholde bufer av fildata som skal bli prioritert, dette går litt mer inn på INF2220 men la oss gjøre det i C) .

Del 7 Bitoperasjoner

Nødvendig?

Hvorfor har vi akkurat bitoperasjoner? Hvorfor er det nyttig?

Enkoding og dekoding

Gjør det samme som dere har gjort i obligen men,

Filen dere leser inn skal inneholde bare tegnene a, b, c, d, e, f, g, h, i, j. Det gjør det mulig å komprimere bokstavene entydig med 4 bit per bokstav (a=0001, b=0010, c=0011, ..., j=1010).Dere skal kunne både komprimere og dekomprimere strengene.

Del 7 Prosesser og tråder

Prosesser er fint å ha for å gjøre flere arbeider i et, det er jo tråder i tillegg.

1. Lag en prosess som skriver ut helloworld mens du er i shell løkka se gjerne eksempler på bruk av prosesser: fork tutorial
2. Lag en tråd istedenfor en prosess se gjerne på eksempellink her: pthread tutorial
3. Gjenbruk noe fra oppgave en men prøv å få til slik at vi har et delt ressurs(datastruktur) hvor trådene skal låses i form av mutex (sjekk oppgave 2) dersom en annen tråd er innenfor det kritiske sonen. (vanskelig).
4. Gjenbruk noe av oppgave 1 med shelløkka og opprett en prosess som konstant evig skriver ut ***"hei og hå jeg plager deg!!!"*** la prosessen sove i 1 sekund, mens du holder på i shelløkka.
5. Gjør det samme som oppgave 4, men istedenfor prosesser, bruk tråder.(middels vanskelig)

Del 8 Interprosess kommunikasjon

TCP select

Lag et chatteprogram med TCP støtte hvor dere lager klient og server som skal kommunisere med hverandre, dersom serveren mottar forespørsel fra klienten, skal den begynne å skrive til klienten (dette blir mer 2 som chatter med hverandre, alternativt vil være å gjøre programmet til å kommunisere og chatte med flere brukere, da må serveren sentraliseres og motta forespørsel fra flere klienter ved hjelp av select()).

For mer om server og klient så anbefaler jeg å sjekke dette:

Beej socket programming

Berkeley socket UNIX

Del 9 Ifi Epostsystem. (vanskelig)

Server

Serveren burde implementeres først for å sette opp slik at clientene kan kobles opp mot dette. Lag et TCP forbindelse hvor serveren skal lagre dataene persistent. Dette vil si at dersom serveren har avsluttet, må dere lagre dataene i en fil og dersom serveren kommer opp igjen, kan den laste ned dataene fra denne filen.

Et server skal bare motta data og videresende dataene til oppgitt client som er pålogget, dersom den ikke er det, må den lagre meldingen i form av en køliste.

(Merk! Lista kan fortsatt inneholde flere meldinger til clienten, og man må gjerne fikle litt med fjerning av objektet fra lista dersom det er flere meldinger!).

Serveren skal i tillegg lagre clienter på en viss måte (dette må dere finne ut selv, f.eks liste, arrays etc...) og dere må i tillegg frigjøre minne dersom serveren skal avslutte. Dere må i tillegg gjøre noe form for signal behandling(ctrl-c) dvs dersom dere trykke på ctrl-c, så må det frigjøres minnet.

Bruk DEBUG for å gjøre det enkelt for dere (DEBUG er typisk bruk for å sjekke om feil, retting osv osv) så dere må ha på Makefilen deres -DDEBUG

et eksempel:

```
#ifdef DEBUG
 fprintf(stderr,"Dette er kanksje noe feil eller riktig eller noe?");
#endif
```

Client

Clienten skal typisk lagre meldingene på innboks, utboks, kladd, spam, sendt. Dere kan selv velge hvordan lagringsmetode å gjøre, et forslag er bare en liste eller array.

En typisk melding vil inneholde:

- Fra person
- Til person
- Dato for melding sendt
- Emne hva er det meldingen handler om
- Melding
- Flag for å indikere statusen til meldingen.

Dere må i tillegg legge til flags, dvs det blir litt bitshifting for å indikere om meldingen er ulest, lest, spam osv. Det du kan gjøre er å bruke OR funksjonaliten generelt.

Clienten skal automatisk få meldingsvarsel (sjekk på funksjonen ***select()***).

Det dere burde begynne med, er å sende chars pekere eller array, deretter kan dere prøve dere på å sende structs videre til servern. Sjekk gjerne denne linken for sending av structer!

http://www.uio.no/studier/emner/matnat/ifi/INF3190/v12/orakel/faktastrukt.html

NB! Dersom dere skal begynne å sende 64 bits data som long, må dere gjerne prøve selv å finne ut hvordan dere skal oversette dette for 32 bits maskiner å kommunisere med 64 bits maskiner (Men for enkelhetsskyld jobber vi data lavere enn det).

Del 11 Brukergrensesnitt NCurses (vanskelig)

Et terminal er jo ikke brukervennlig for mennesker som ikke driver med programmering, det som hadde vært greit er å implementere Ncurses i programmet deres. Frigjør minne generelt fra de øvrige oppgavene, for ved erfaring så har Ncurses helt pinlige minnelekkasjer.For mer om Ncurses, sjekk denne linken:

Ncurses

For å gi dere et bedre innblikk på Ncurses funksjonaliteten med typisk client-server struktur kan dere sjekke linken fra Khiem-Kim sin side (sjekk linken fra sida NCurses made by Mats Rosbach):

http://folk.uio.no/kkho/programs.html