Våren 2021
IN2140 – Introduksjon til operativsystemer

Obligatorisk oppgave 2

I denne oppgaven skal du bruke det du har lært til nå og skrive et større program. Vi skal jobbe med filer, structer, strenger, pekere og systemkall.

Oppgaven skal løses selvstendig. Se reglene for obligatoriske oppgaver og andre innleveringer ved Ifi. Dersom du har spørsmål underveis kan du oppsøke en orakeltime/gruppetime.

Husk å lese hele oppgaveteksten, så du får et helhetsinntrykk av omfanget og hvilke utfordringer du kan møte. 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. Programmet **må** la seg kompilere og kjøre på Ifi sine login-maskiner (login.ifi.uio.no).
Innlevering i Devilry innen 17. mars kl. 23.59.

1 Oppgaven

Vi tenker oss et scenario der Ifi Drift ønsker et «bedre» system for å administrere ruterne de har i drift ved Ole Johan Dahls hus og Kristen Nygaards hus. De ønsker seg et program for å lagre og behandle grunnleggende informasjon om ruterne. Programmet skal ta imot to filer. En fil som beskriver ruterne og kobliger mellom dem. Og en annen fil med kommandoer som skal utføres. Hvordan disse filene ser ut kommer vi til i seksjon 1.1.

Vi må kunne kompilere programmet ved å kalle `make`; **dere må altså lage en makefile**. Programmet må hete `ruterdrift` og startes med to filnavn som argumenter. Det første filnavnet skal være navnet på en fil som inneholder data om rutere som kan leses og behandles av programmet. Den andre filen inneholder en eller flere linjer med kommandoer som skal utføres. For eksempel må det være mulig å utføre følgende kommando:

```
$ ./ruterdrift topology.dat commands.txt
```

Dere kan anta at programmet kjøres med rett antall argumenter, men dere bør håndtere filer som ikke eksisterer. Dere må derfor sjekke om åpning av filene er vellykket.

Som en del av oppgaven utleveres følgende filer:

```
.
├── oblig2.pdf
└── testing_data
    ├── 10_routers_15_edges
    │   ├── commands.txt
    │   ├── topology.dat
    │   └── topology.png
    ├── 50_routers_150_edges
    │   ├── commands.txt
    │   ├── topology.dat
    │   └── topology.png
    └── 5_routers_fully_connected
        ├── topology.dat
        └── topology.png
```

I tillegg til denne oppgaveteksten finnes altså tre filsett, som beskriver tre forskjellige topologier. Hvert filsett består av en datafil, som definerer topologien, en graf som illustrasjon og (bortsett fra den minste topologien) en liste med kommandoer til bruk for testing. Dere bør bruke disse filene i testingen av programmet deres. Det er viktig at systemet ikke har minnelekasjer og andre minnefeil. Dette kan dere sjekke ved å bruke Valgrind.

1.1 Filstrukturer

I denne seksjonen spesifiseres de to filene, som programmet skal lese.

Fil 1: Informasjon om rutere

Denne filen beskriver noder og kanter i en graf. Vi bruker begrep fra nettverk og kaller nodene for **rutere** og kantene for **koblinger**. Ruterne har egenskaper som filen også beskriver.

Filen inneholder binærdata og kan dermed *ikke* leses i sin helhet i en vanlig teksteditor som Atom eller Notepad. Til gjengjeld er det enklere å skrive kode for å lese inn or skrive ut talldata, da man ikke trenger å tenke på å gjøre om mellom tall i tekstform og faktiske tall i minnet.

Det første som ligger i filen er en `int`, et firebytes tall, som forteller oss hvor mange rutere filen har informasjon om. Dette er altså ikke noen tall på leselig form. Kall dette tallet N.

Etter disse fire bytene følger det N informasjonsblokker, da hver informasjonsblokk gir informasjon om én ruter. En gyldig fil uten ruterdata (altså med null informasjonsblokker) inneholder kun fire bytes med verdi null (men alle utleverte filer og testfiler har informasjon om minst én ruter).

En informasjonsblokk er maksimalt 256 bytes lang. Denne restriksjonen skal opprettholdes når du skriver nye data til fil. Hver informasjonsblokk inneholder informasjon på følgende form:

- Ruter ID – fire bytes

  Denne tallverdien er unik.

- Flagg – én byte

- Lengde på produsent-/modellstrengen – én byte

  Denne lengden teller bare de menneskelesbare bokstavene i strengen som følger – altså ikke noen nullverdi, tabulatortegn eller linjeskift.

- Produsent/modell – maksimal lengde 249 bytes

  Lengden i bytes er nøyaktig den som er gitt i lengdefeltet.

- Informasjonsblokken avsluttes med en nullbyte.

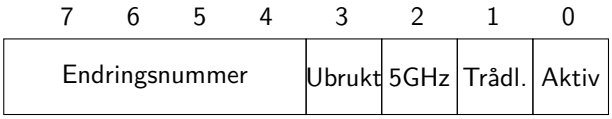
Hver informasjonsblokk i filen begynner altså seks bytes informasjon om ruteren, etterfulgt av lesbare bokstaver for produsent/modell og en avsluttende nullbyte.

I programmet skal hver informasjonsblokk lagres i en struct, som naturlig nok må ha feltene `id`, `flagg` og `modell` (kan selvfølgelig kalles noe annet). Dere bør utvide structen etter behov for å løse oppgaven.

Flaggbyten skal beskrive diverse egenskaper hos en ruter, som beskrevet nedenfor.

Bitposisjon	Egenskap	Forklaring
0	Aktiv	Er ruteren i bruk?
1	Trådløs	Er ruteren trådløs?
2	5GHz	Støtter ruteren 5GHz?
3	Ubrukt	Ikke i bruk
4:7	Endringssnummer	Se lenger nede for info

Tabell 1: Bitene i flaggbyten



Figur 1: Flaggbyten

Etter N informasjonsblokker om ruterne følger et ukjent antall informasjonsblokker om koblinger mellom ruterne. Her består hver informasjonsblokk av åtte bytes og inneholder to ruter-ID-er.

ID-ene er lagret i binært format og består av fire bytes i little endian. Her kaller vi den første ID-en R1 og den andre ID-en R2. En kobling fra ruter R1 til ruter R2 betyr at structen for ruteren R1 skal ha en peker til R2. Koblingene er enveis, så det følger ikke at R2 $\rightarrow$ R1 når R1 $\rightarrow$ R2. Merk at R2 $\rightarrow$ R1 også kan forekomme i filen slik at man får en toveis kobling.

Dere kan anta at det er maksimalt 10 koblinger fra enhver ruter R1 til andre rutere. Det kan derfor allokeres minne til 10 pekere for hver kobling uansett faktisk antall. Dere kan ha en array av pekere i structen med informasjon om en ruter, men koden må kunne skille mellom pekere som er i bruk og pekere som ikke er det.

Alle disse koblingene definerer en rettet graf som dere skal traversere med et grafsøk. Detaljer rundt grafsøket er beskrevet nedenfor.

Fil 2: Kommandoer

Denne filen inneholder en rekke kommandoer. Hver linje inneholder én kommando. De forskjellige kommandoene har forskjellig format. Her følger en beskrivelse av kommandoene.

- `print [ruter-ID]`

  Skriver informasjon on ruteren og ID-ene til dens nærmeste naboer til stdout.

  Utskriften skal inneholde ID-ene både for rutere med kobling til ruter-ID og fra ruter-ID. Print-kommandoen er godt egnet for feilsøking.

- `sett_flagg [ruter-ID] [flag] [verdi]`

  Setter flagg 0, 1 eller 2 to enter verdi 0 eller 1 og flagg 4 til verdi mellom 0 og 15 i ruteren med gitt ruter-ID. Flagg 0 setter aktivbiten til verdi. Flagg 1 setter trådløsbiten til gitt verdi. Flagg 2 setter 5 GHz til gitt verdi. Flagg 3 er ugyldig, så skriv gjerne ut en feilmelding. Flagg 4 setter endringsnummeret til gitt verdi. Ugyldige verdier skal ignoreres, men skriv gjerne en varselmelding.

- `sett_modell [ruter-ID] [navn]`

  Setter navnet til ruteren med ruter-ID.

- `legg_til_kobling [ruter-ID1] [ruter-ID2]`

  Legg til en kobling from ruteren med ruter-ID1 til ruteren med ruter-ID2. Endringsnummeret i ruter ruter-ID1 økes med 1.

- `slett_ruter [ruter-ID]`

  Slett ruterinformasjon. Merk at alle pekere til denne ruteren også må slettes. Endringsnummere i alle rutere som er berørt av slettingen økes med 1.

- `finnes_rute [ruter-ID1] [ruter-ID2]`

  Sjekker om det finnes en rute fra ruteren med ruter-ID1 til ruteren med ruter-ID2. Du skal foreta et grafsøk fra den første ruteren og se om du kan komme deg til den andre ruteren. En mulighet, som er overraskende enkel med rekursjon, er å gjøre et dybde-først-søk. For enkelthets skyld trenger du ikke å skrive ruten du finner, men gjør det gjerne hvis du har tid – og er helt sikker på at alt annet fungerer som det skal.

Alt i denne filen er lesbart med en vanlig teksteditor. `ruter-ID`, `flag` og `verdi` er alle tall på leselig form. De må derfor konverteres til riktige datatyper, for eksempel ved bruk av `strtol`.

1.2 Spesifikasjoner

Her er litt mer spesifikk informasjon om hva en bruker skal kunne bruke programmet til, og hvordan programmet skal fungere. Grovt sett vil programmet være delt i tre:

1. Lese inn filen og opprett de datastrukturer du trenger.
2. Utføre kommandoer gitt i en kommandofil.
3. Skrive til fil og avslutte.

Innlesing og datastrukturer

Dette skal skje med en gang programmet starter. Den første filen gitt som argument skal leses inn og dataene skal lagres i minnet. For hver ruterinformasjonsblokk i filen skal det allokeres plass til en struct (med `malloc`); structen skal fylles med data fra informasjonsblokken.

En peker til structen skal lagres i et globalt, dynamisk allokert array. Dette minnet skal allokeres med plass til N rutere. Når alle ruterinformasjonsblokker har blitt lest inn, fått sin egen struct og sin egen entry i den globale arrayen, går programmet videre til å lese koblingsinformasjonsblokker i en løkke helt frem til filslutten oppdages.

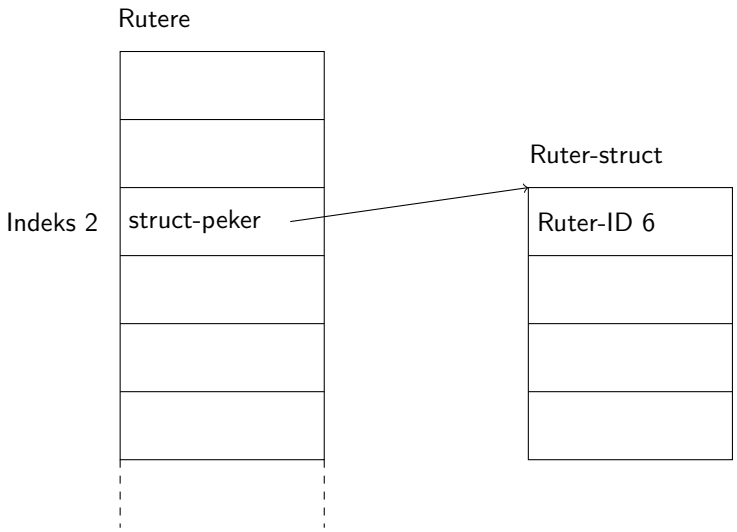
Hver gang en koblingsinformasjonsblokk er lest inn, oppdateres ruterstructen som er fra-siden i en kobling. Er det feil i en koblingsinformasjonsblokk slik at enten fra- eller til-siden ikke finnes, så gir programmet en advarsel og fortsetter med neste koblingsinformasjonsblokk.

Etter at hele informasjonsfilen er lest inn, går programmet videre til kommandofilen.

Avslutning

Når programmet har utført alle kommandoer i kommandofilen er nettverket sannsynligvis endret. Du skal skrive en ny fil, som inneholder all informasjon om grafen. Filen skal bruke det samme formatet som er beskrevet i 'Fil 1: Informasjon om rutere'. Det skal være mulig å kjøre programmet ditt på nytt med filen du skriver og en kommandofil. Filen som inneholder det nye nettverket skal hete 'new-topology.dat'.

Til slutt skal alle allokerte minneområder (allokert med `malloc`) frigis ved kall på `free`.



Figur 2: Global array for rutere

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 er med tanke på hva som blir viktig mot hjemmeeksamen og hva som er det sentrale ved oppgaven. Dette vil bli brukt ved retting.

1. Oppretting av arraystrukturen
2. Innlesing av data fra fil
3. Riktig innsetting av data i arrayen
4. God bruk av minne (`malloc` og `free`).
5. Print.
6. Legg til kobling.
7. Slett en ruter.
8. Skriving av data til fil
9. Endring av et navn.
10. Endring av falggbyten i en ruterstruct.
11. Sjekke om det finnes en sti mellom to rutere.

Med andre ord: **Sørg for at innlesing og oppretting av datastrukturen fungerer først. Sørg deretter for å skrive til fil på riktig vis. Pass på minnebruk hele veien.** Først når dette fungerer bør du begynne å se på kommandohåndtering.

Kommentarer

Vi krever at dere kommenterer følgende i programmet deres:

- Kall til `malloc`: ca. hvor mye minne i bytes som allokeres.
- Kall til `free`: hvor minnet ble allokert.
- Funksjonsparametere: typer og kort hva de er.
- Funksjonsflyt: kort hva funksjonen gjør – med mindre det er åpenbart.

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`.

- `malloc/calloc/realloc`
- `fgetc`
- `fread/fwrite`
- `fopen/fclose`
- `scanf/fscaf`
- `strcpy`
- `memcpy`
- `memmove`
- `atoi`
- `strtol`
- `isspace/isdigit/alnum` m.fl.
- `strdup`
- `strlen`
- `strtok`
- `printf/fprintf`

Andre hint

Da ruterdataene ikke kan leses som vanlig tekst, er det lurt å ha en annen måte å inspisere filen(e) på. Til dette finnes blant annet terminalverktøyet `xxd`. Dette viser filen som heksadesimale tall ved siden av en tekstlig representasjon. Man kan da for eksempel enkelt se at de første fire bytene inneholder tallverdien som sier hvor mange rutere det er i filen.

Kjør Valgrind!

2 Teorioppgaver

Hodet på en harddisk må bevege seg mellom sporene på en disk for å svare på lese- og skriveforespørsler. Dette er den tregeste delen av diskdriften og krever gode strategier.

Se for deg følgende scenario:

Det innerste sporet på en disk er spor 0; det ytterste er spor 31. Diskhodet er plasert på spor 18 og beveger seg utover.

Disken har mottatt følgende forespørsler i gitt rekkefølge:

yngste forespørsel	4
	30
	10
	20
	14
	7
eldste forespørsel	24

Etter at disken har fullført flyttingen til det andre sporet den skal lese (her spor 7 fordi FIFO benyttes som strategi), mottar den en annen forespørsel som følger:

ny forespørsel	23

2.1 Spørsmål 1

Hvis strategien er C-SCAN med leseretning utover og den ikke-lesende bevegelsen innover, i hvilken retning beveger diskhodet seg umiddelbart etter å ha lest spor 30? Kryss av riktig svar:

innover	
utover	

2.2 Spørsmål 2

Hva er svaret på spørsmål 2.1 hvis strategien er C-LOOK? Kryss av riktig svar:

innover	
utover	

2.3 Spørsmål 3

Hva er summen av avstander som hodet må bevege seg fra sin nåværende stilling (18) til det siste sporet etter at disken har besvart alle forutgående forespørsler? Fyll inn summene for følgende strategier:

FIFO	
SSTF	
LOOK	

Hint

Et hode som beveger seg direkte fra spor 5 til 22, beveger seg over en distanse på 17, ikke 18. Sporet som hodet forlater teller ikke.

Format

Svaret for oppgave 2 må leveres i en egen PDF-fil med navn `oppgave2.pdf`.

3 Levering

1. Lag en mappe med ditt brukernavn:
   `mkdir bnavn`
2. Lag mapper for begge deloppgavene:
   `mkdir bnavn/oppgave1` og `mkdir bnavn/oppgave2`
3. Kopier alle filer som er en del av innleveringen inn i deloppgavens mappe:
   `cp *.c bnavn/oppgave1/` og `cp oppgave2.pdf bnavn/oppgave2/`
4. Komprimer og pakk inn mappen:
   `tar -czvf bnavn.tgz bnavn/`
5. Logg inn på Devilry
6. Lever under IN2140 → Vår 2021 → Obligatorisk oppgave 2.