INF1060: avsluttende eksamen

Kandidatnummer: _____

1 Oppgave 1 - operativsystemer — *max 39 poeng*

Løsningsforslagene er mer eller mindre hentet fra foilene. Merk at andre forklaringer også kan være fornuftige.

- 1a **(8 poeng)** – NB! I bokmålutgaven av løsningsforslaget er det falt ut at det skal være 5 processer. Dette ble det opplyst om, men hvis man derfor svarer generelt er dette ok.
  - 1a - 1 **(2 poeng)** – Her vil både FIFO og RR kjøre en prosess til den er ferdig og så starte neste pga "non-preemtion". Dvs. at det ikke spiller noen rolle om RR har en time-slice eller ikke siden den ikke vil bli avbrutt uansett:
    START:
    prosess1 - 0s, prosess2 - 50s, prosess3 - 100s, prosess4 - 150s, prosess5 - 200s, ..., prosess*n* - *n-1* x 50s
    STOP:
    prosess1 - 50s, prosess2 - 100s, prosess3 - 150s, prosess4 - 200s, prosess5 - 250s, ..., prosess*n* - *n* x 50s
  - 1a - 2 **(2 poeng)** – nå vil RR avbryte hver process etter 5s
    START:
    prosess1 - 0s, prosess2 - 5s, prosess3 - 10s, prosess4 - 15s, prosess5 - 20s, ..., prosess*n* - *n-1* x 5s
    STOP:
    prosess*n* - [#jobs x (joblength - timeslice)] + *n* x timeslice = 50 x 45s + *n* x 5s
    for 5 prosesser: prosess1 - 230s, prosess2 - 235s, prosess3 - 240s, prosess4 - 245s, prosess5 - 250s
  - 1a - 3 **(2 poeng)** – CPUen vil være (nesten) 100% utnyttet uansett (hvis vi ser bort i fra overhead noe oppgaven ikke tar hensyn til).
    Processeringen vil gå noe slikt hvor DISKprocess1 blir avbrutt etter 1 sekund:
    CPUprocess1: 5s, CPUprocess2: 5s, CPUprocess3: 5s, CPUprocess4: 5s, CPUprocess5: 5s, DISKprocess1: 1s
    –> CPU: 26s versus DISK: 5s –> ∼20% (19.23%) disk utnyttelse
  - 1a - 4 **(2 poeng)** – Et godt forslag ville være kortere time-slices
    (f.eks. vil en time slice på 1 sekund gi 83% disk utnyttelse)
- 1b **(6 poeng)** –
  En prosess deler ofte tilgjengelig minne inn i et tekst/code segment (programinstruksjonene som brukes av exec), et data segment (globale variable og en heap for dynamiske minneregioner) og et stakk segment (funksjons variable, register tilstander, ...). I tillegg er det som regel et system data segment (process contol block, tilstandsinformasjon som PID, pekere, etc.). Her bør det også nevnes at heap og stakk vokser mot hverandre - stakk fra høye adresser mot lave og heap fra lave adresser mot høye. Se fig. 1.
- 1c **(11 poeng)** –
  - 1c - 1 **(2 poeng)** –
    Her bør man nevne magnetiske plater som roterer rundt en "spindle". Arealet på platene er delt inn i tracks (spor), som igjen er delt inn i tracks. I tillegg må man nevne diskhodet som styres over platene for å lese de forespurte bittene fra et bestemt spor på en bestemt overflate.
    Aksesstiden består stort sett i å søke (flytte hodet til riktig sylinder/spor), vente til platene har rotert slik at første

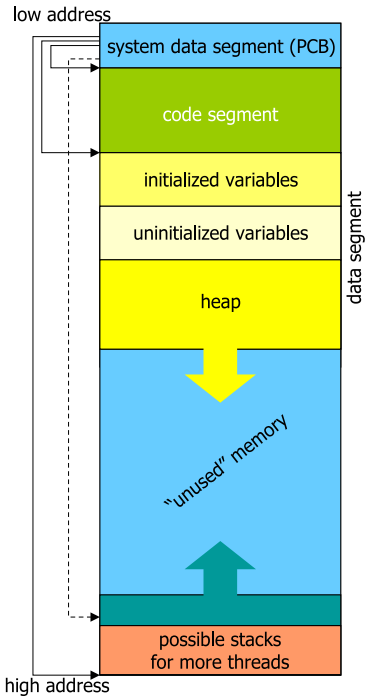

Figure 1: En process sin minnelayout

forespurte bitt ligger under hodet (rotasjonsforsinkelse) og overføre dataene som er tiden det tar for platene å rotere slik at alle bittene i den forespurte blokken passerer under diskhodet. (PS! Det er også andre forsinkelser, men disse ser vi stort sett bort i fra)

- ❏ 1c - 4 **(4 poeng)** –
  FIFO: (43) → 4 → 12 → 98 → 1 → 92 → 78
  SCAN: (43) → 78 → 92 → 98 (opp til 99 og tilbake) → 12 → 2 → 1
- ❏ 1c - 4 **(5 poeng)** – Her kan man regne ut på flere forskjellige måter, men ettersom det ikke er akselerasjon (konstant tid per track for å flytte armen) kan man forenkle litt og summere antall blokker for å finne total overføringstid og summere antall spor hodet må bevege seg over for å finne den totale lengden hodet beveger seg på.
  tidPerRotasjon = 60 sekunder / 2500 RMP = 24 msPerRotasjon
  antallRotasjoner = (6 blokker * 8 sektorerPerBlokk) / 64 sektorerPerTrack = 0.75 rotasjoner
  overføring = 0.75 rotasjoner * 24 msPerRotasjon = *18 ms*
  gjennomsnittligRotasjonsForsinkelse = 6 blokker * $\frac{1}{2}$ * 24 msPerRotasjon = *72 ms*
  søketiden er så avhengig av schedulingalgoritme ():
  FIFOsektorer = (43 - 4) + (12 - 4) + (98 - 12) + (98 - 1) + (92 - 1) + (92 - 78) = 335 spor
  FIFOtid = overføring + gjennomsnittligRotasjonsForsinkelse + (FIFOsektorer * 10 msPerSpor) = 18 ms + 72 ms + (335 * 10) ms = **3440** ms
  SCANsektorer = (78 - 43) + (92 - 78) + (98 - 92) + (99 - 98) + (99 - 12) + (12 - 2) + (2 - 1) = 154
  SCANtid = overføring + gjennomsnittligRotasjonsForsinkelse + (SCANsektorer * 10 msPerSpor) = 18 ms + 72 ms + (154 * 10) ms = **1630** ms

❏ 1d **(8 poeng)** –

- ❏ 1d - 1 **(4 poeng)** –
  Caching er en mekanisme for å (midlertidig) oppbevare/holde/lagre en samling av duplikate data fra en samling data som ligger lagret på et annet nivå i lagringssystemet pga at det er dyrt (f.eks. mhht data overføringstid) og det derfor er "billigere" å operere på dataene i cachen. Eksempler kan være at man cacher data i primærminnet for å ungå å måtte hente data på disk, eller at man holder data i minnet på chipen (også ofte kalt cachen) for å unngå og aksessere primærminnet. Som sagt, grunnen er at det er mye raskere å hente dataene fra det høyere nivået i lagrinshierarkiet.
- ❏ 1d - 2 **(4 poeng)** –
  Det kan være flere muligheter, men figur **??** viser et eksempel som er gjennomgått i forelesning. For utbytting av

enheter har man ofte en liste sortert etter utbyttingsstrategien, f.eks. LRU for å kunne ta alle enheter i cachen i betraktning.
I tillegg er det vanlig å legge på en struktur til for å samtidig kunne gjøre raske oppslag, f.eks. i steden for å måtte søke igjennom hele listen av enheter i cachen, lager man en hash f.eks. på inode-nummer og block-nummer slik at man får mange mindre lister som etterpå er raskt å søke i – noe som reduserer aksesstiden.

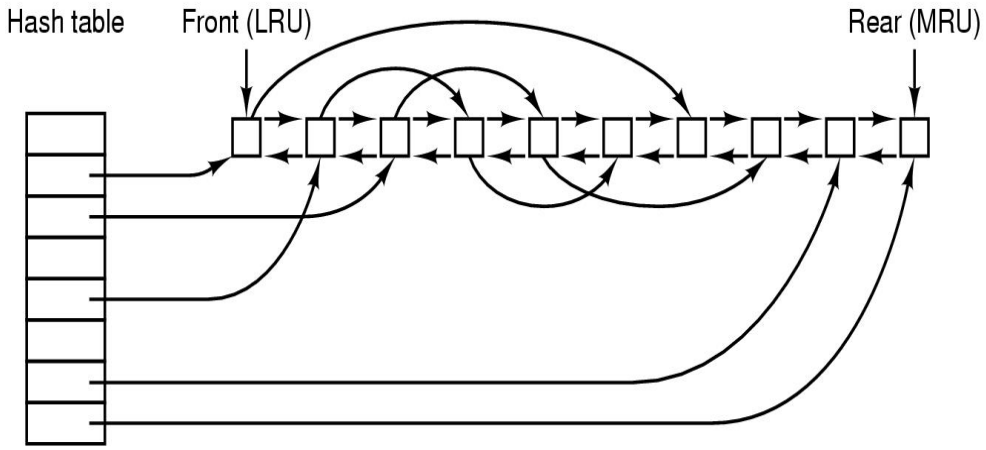

Figure 2: Caching

- 1e **(6 poeng)** –
  En inode (se figur **??**) er en datastruktur i et tradisjonelt UNIX-filsystem. Denne lagrer metadatainformasjon om vanlige filer, kataloger eller andre filsystem enheter.
  Inoden inneholder både metainformasjon som filnavn, eier, rettigheter, modifikasjonstid, ... og datapekere (12 direkte, 1 indirekte, 1 dobbelt-indirekte og 1 trippel-indirekte)

Figure 3: Inode

2 Oppgave 2 - Datakom — *max 40 poeng*

Løsningsforslagene er mer eller mindre hentet fra foilene. Merk at andre forklaringer også kan være fornuftige.

❑ 2a **(3 poeng)** –
Internet er et nettverk av nettverk, dvs. mange proprietære, regionale og offentlige nett knyttes sammen via rutere i kjernenettverket. Internet er delvis hierarkisk.

❑ 2b **(3 poeng)** –
Protokoller definerer formater, rekkefølgen for sending og mottak av meldinger, samt hvilke aksjoner som mottak av meldinger initierer.

❑ 2c **(5 poeng)** –
De omtalte referansemodellene er lagdelte (hierarkiske). Hovedgrunnen til at de har blitt spesifisert er at kommunikasjonssystemer er meget komplekse, og det viste seg å være nødvendig å modularisere. Dette forenkler bl.a. design, vedlikehold og oppdatering av et system. Lagdelingen gjør det mulig å samle og identifisere ulike funksjonelle egenskaper i ulike lag og å spesifisere relasjoner mellom dem. Intensjonen er også at ulike implementasjoner av et lag skal kunne fungere sammen med alle implementasjoner av andre lag (dvs. det er funksjonelle modeller). En god besvarelse bør inneholde en beskrivelse av hovedfunksjonene til hvert lag.
Hovedforskjellen mellom TCP/IP og OSI modellen er at førstnevnte mangler Sesjons- og Presentasjonslag. Dette innebærer at Applikasjonene i Internet selv må ivareta de funksjonene som disse lagene utfører (mp være innebakt i applikasjonsprotokollene).

❑ 2d **(6 poeng)** –
Linjesvitsjing: galvanisk forbindelse mellom endesystemene. I dag gjøres dette elebluektronisk slik at ressurser kan deles, men brukeren vil bli garantert tilgjengelige ressurser. Fordel: Etter at en forbindelse er etablert har brukeren en garantert båndbredde så lenge forbindelsen er oppe. Ulempe: Siden brukeren er garantert en gitt overføringskapasitet, vil ressurser kunne stå ubrukt når det ikke sendes noe.
Pakkesvitsjing:
Overføringen deles opp i pakker som rutes gjennom nettet på en mest mulig optimal måte. Hver pakke benytter den totale kapasiteten på linken. Det er konkuranse om bruk av nettressursene. Dette innebærer at pakkene kan ta ulike veier gjennom nettet alt etter hvordan de enkelte linkene/nodene er belastet. Denne formen for ressursdeling er den viktigste fordelen (men det kan også være en ulempe).Den viktigste ulempen er at det ikke gis garantier for at pakker kommer fram og det kan ikke gis en garantert overføringshastighet (throughput).
Følgende egenskaper gjelder for bruk av henholdsvis linjesvitsjing og pakkesvitsjing:

1. Linjesvitsjing
   - maks 10 brukere
   - Sannsynlighet for tap: 0%
   - Ubrukt kapasitet: ca.90%
2. Pakkesvitsjing
   - >10 kan v¾re active samtidig!
   - Sannsynlighet for tap >0%
   - Ubrukt kapasitet: < 90%

❑ 2e **(5 poeng)** –
Hovedteknikkene er: Tidsmultipleksing, frekvensmultipleksing og pakkemultipleksing.
Kort beskrivelse:
Tidsmultipleksing: hver kanal får tilgang til mediet i en viss tid (tidsluke). Hver kanal avtastes i sekvens om igjen og om igjen. Ledig kapasitet i en kanal kan ikke utnyttes av en annen kanal.
Frekvensmultipleksing: mediet deles inn i frekvensbånd og hver kanal tilordnes hvert sitt bånd. Ledig kapasitet i en kanal kan ikke utnyttes av en annen kanal.
Pakkemultipleksing: Pakker plukkes fra innlinjene og legges i en FIFO utkø i ruteren. Pakkene i denne køen legges etter hverandre ut på utlinja. I mottager-ruteren foretas demultipleksing basert på adresser i pakkene.

❑ 2f **(5 poeng)** –
Hovedkomponentene i en ruter er inn og utkøer, rutetabell samt preprosseserings-, ruting- og framsendings-prosesser. Preprosseserings-prosessen plukker pakker fra inn-køen og leverer adresse-info (samt linjenummer) til ruting-prosessen. Denne oppdaterer rutetabellen vha. den mottatte informasjonen, og overlater pakka til framsendings-prosessen som legger pakka i riktig utkø.

❑ 2g **(5 poeng)** –
UDP: Ingen forbindelser, upålitelig, uordnet, kan miste pakker, pakke-orientert (data en bruker sender sendes som en pakke), ingen 'flyt-kontroll' (sender så fort som mulig), ingen 'congestion control' som vil si at man ignorerer hva som skjer i netverket.
TCP: Forbindelse opprettes ved '3-veis handshake'/nedkobles ved 'teardown', tilstandsinformasjon, pålitelig (mister ikke pakker vha. retrans.), leveres i samme rekkefølge, strømorientert (bufrer opp data og sender en pakke når det ser 'fornuftig' ut, dvs en sendt mengde data fra applikasjonen sendes ikke nødvendigvis ut som en enkelt pakke), har flyt kontroll ved at man ikke sender fortere enn mottaker kan motta, 'congestion control' ved at vi adapterer til problemer i nettverket.

❑ 2h **(5 poeng)** –
Generelt pakkeformat (rammer, pakker, meldinger):

Som vist på figuren nedenfor, får hvert lag på sendersiden data fra det laget over, adderer header informasjon for å generere en ny protokoll data enhet, PDU, (melding, segment, ramme, pakke) etterhvert som PDU-ene flytter oppover. Det vil ofte også være kontroll-informasjon i en 'hale' bakerst i PDUene som behandles tilsvarende.

❑ 2j **(3 poeng)** –
Kommunikasjonen over nettet mellom distribuerte applikasjons-prosesser må foregå i en (standardisert) syntaks som begge sider oppfatter på samme måte. En slik syntaks kaller vi 'overføringssyntaks'. Denne er nødvendig fordi kommunikasjonen foregår mellom inhomogene ende-systemer (eks. ulik maskinvare, ulike OS, ulike programmeringsspråk).

3 Oppgave 3 - flervalgsoppgaver — *max 21 poeng*

Hvert riktige gir 1 poeng.

❑ 3a – **OS (11 poeng)**: 2, 6, 8, 10 og 11 er riktige

❑ 3b – **Datacom (10 poeng)**: 2, 4 og 8 er riktige

➽ **Poeng totalt:** _____ (*max 100 poeng*)