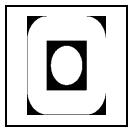
**NTNU**
**Norges teknisk-naturvitenskapelige universitet**

**Fakultet for informasjonsteknologi, matematikk og elektroteknikk**

**Institutt for datateknikk og informasjonsvitenskap**



**Løsning på SIF8042 Distribuerte systemer**
**Tirsdag 27. mai 2003, 0900-1300**

Det ønskes korte og konsise svar på hver av oppgavene. Det vesentlige er å kunne dokumentere forståelse, beherske prinsipper og se sammenhenger - ikke å kunne gjengi en mengde detaljer.

Der det synes å mangle noen opplysninger, må det angis hvilke antagelser som synes å være naturlige. Merk at viktige begreper er angitt på både norsk og engelsk.

## Oppgave 1 – Standarder (Standards) – 20%

a) Angi kort noen viktige mellomvarestandarder som brukes ved implementasjon av distribuerte systemer

SVAR:

Her forventes i alle fall de tre første omtalte varianter,
gjerne også de fire første, og helst de fem første

ISO/RM-ODP
OMG/CORBA
OSF/DCE
MS – DCOM/OLE
X/OPEN – TX/XA
TINA
MSS

b) Diskuter kort fordeler og ulemper med hver av disse mellomvarestandardene

SVAR:

*ISO/RM-ODP*

* OO-type, rammeverk-nivå
+ Omfatter det meste som trengs
- Enormt begrepsmessig apparat

*OMG/CORBA*

* ORB-type, arkitektur-nivå
+ OO-basert
- Mye å sette seg inn i

*OSF/DCE*

* RPC-type, arkitektur-nivå
+ Enkel å ta i bruk
- Ikke OO-basert

*MS – DCOM/OLE*

* RPC/OO-mellomting, blanding av alt
+ MS støtter det
- MS bestemmer alt

*X/OPEN – TX/XA*

* DTP-fokus, blankt på mye
+ Tidlig fokus på transaksjoner, således litt oppdragende på andre
- Inkludert i andre etter hvert, således litt overflødig i seg selv

*TINA*

* Telekom-rettet, spesialisering av RM-ODP
+ Effektivt verktøy i sin valgte nisje
- RM-ODP++, for mye av det gode !

*MSS*

* Multimedia-rettet, spesialisering av CORBA
+ Effektivt verktøy i sin valgte nisje
- CORBA++, for mye av det gode ?

**Oppgave 2 – Synkronisering (Synchronization) – 20%**

a) Drøft kort overordnete krav som bør stilles til algoritmer for å oppnå gjensidig utelukkelse (mutual exclusion) i et distribuert system

SVAR:

*Generelt for distribuerte system*

Unngå avhengighet av enkeltnoder / enkeltmeldinger
Unngå / begrens flaskehalser i systemet (noder / lenker)
Begrens meldingsmengder / prosesseringsforsinkelser

*Spesielt for gjensidig utelukkelse*

Tjenesten må implementere en absolutt gjensidig utelukkelse
En klient som belegger den tilsvarende ressurs må forventes å frigi den til andre klienter
En klient som ikke belegger den tilsvarende ressurs må ikke kunne forhindre andre klienter
Tjenesten må unngå vranglås og/eller utsulting

b) Illustrer kort minst en aktuell algoritme for å løse denne oppgaven

SVAR:

Av de tre nedenfor viste varianter (enkel / avansert / typisk) holder det med en -
helst den typiske eller den avanserte, helst ikke den enkle

*Sentralisert algoritme (enkel)*

| |
|---|
| Eget Inn-ønske:<br>1a) Send Inn-melding til Global Koordinator<br>1b) Vent på OK-melding fra Global Koordinator |

| |
|---|
| Eget Ut-ønske:<br>2a) Send Ut-melding til Global Koordiator |

| |
|---|
| Ulike Inn-meldinger:<br>3a) Ingen inne: Send OK<br>3b) Noen inne: Sett i Kø |

| |
|---|
| Ulike Ut-meldinger:<br>4a) Noen i Kø: Fjern fra Kø og Send OK |

*Distribuert algoritme (avansert)*

| |
|---|
| Eget Inn-ønske:<br>1a) Send N-1 * Inn-meldinger m/ Lokalt tidsmerke<br>1b) Vent på N-1 * OK-meldinger |

| |
|---|
| Andres Inn-meldinger:<br>2a) Ikke ventende: Send OK<br>2b) Ventende, høyere tidsmerke: Send OK<br>2c) Ventende, lavere tidsmerke: Sett i Kø<br>(Like tidsmerker: Identifikator avgjør) |

Eget Ut-ønske:
3a) Send $\leq$ N-1 * OK ut fra Kø

*Ringalgoritme (typisk)*

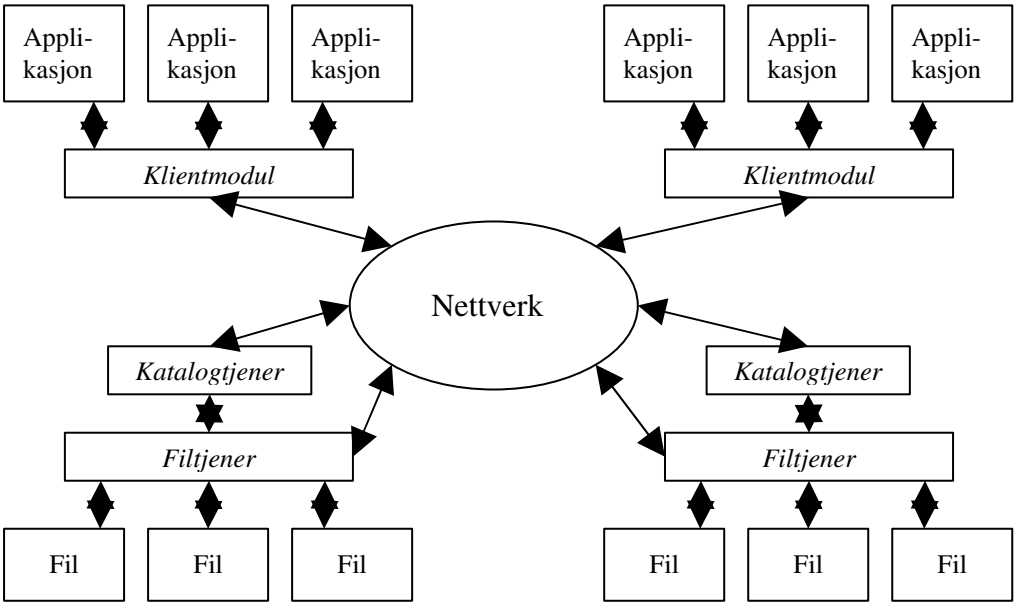
| |
|---|
| Eget Inn-ønske:<br>1a) Vent på OK-melding |

| |
|---|
| OK-melding:<br>2a) Ikke ventende: Send OK-melding Videre<br>2b) Ventende: Behold OK-melding |

| |
|---|
| Eget Ut-ønske:<br>3a) Send OK-melding Videre |

**Oppgave 3 – Distribuerte filsystemer (Distributed file systems) – 20%**

Figuren nedenfor illustrerer tre vanlige komponenter i et distribuert filsystem; klientmodul (client module), katalogtjener (directory server) og filtjener (file server).




a) Beskriv kort hvilke oppgaver som typisk løses av hver av tre angitte komponenttypene

SVAR:

*Filtjener*

Mest maskinnære funksjoner
Operasjonshistorie, hvis aktuelt (¬ Tilstandsløshet)

*Katalogtjener*

Oversetter fra brukernære begreper (for eksempel navn)
til maskinnære begreper (for eksempel adresser)
Samler filer i større klasser / grupper

*Klientmodul*

Mest brukernære funksjoner
Operasjonsgjentak, hvis aktuelt (Idempotens)

b) Drøft kort hvordan samvirket mellom de tre angitte komponenttypene kan fungere

SVAR:

*Funksjonelt*

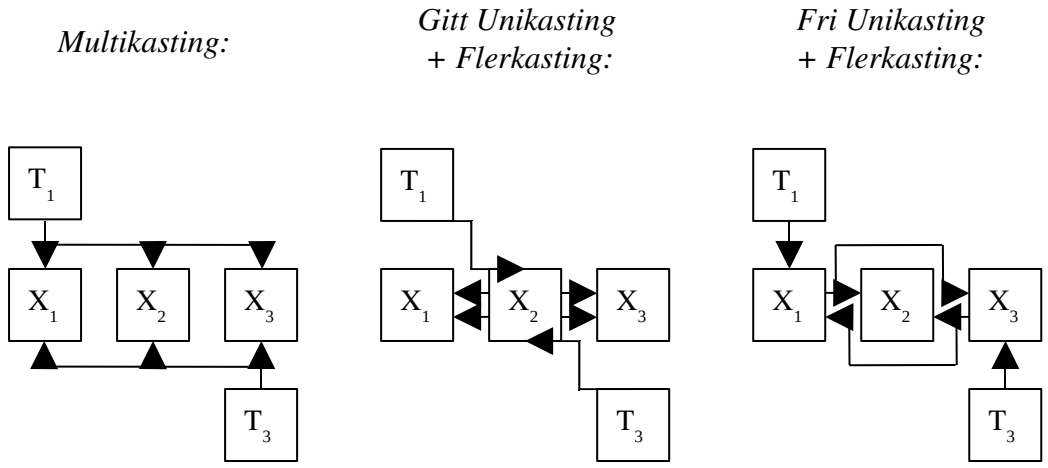
Noen klientmodul funksjoner er direkte koplet til enten filtjener funksjoner eller katalog-tjener funksjoner, mens andre klientmodul funksjoner er koplet til både filtjener og katalog-tjener funksjoner (for eksempel filskaping)

*Caching / Replisering*

Oppdatering i.f.m. fjerncaching (fra tjener til klient) styres gjerne av gjennomskriving (øyeblikkelig kopiering fra klient til tjener)
Oppdatering i.f.m. nærcaching (fra sekundærlager til primærlager på tjener / klient) styres gjerne av tilbakeskriving (senere kopiering fra primærlager til sekundærlager)
Oppdatering av en cachet kopi (på en klient) fører gjerne til invalidering av alle andre cachete kopier (på andre klienter)
Oppdatering av en replisert kopi (på en tjener) fører gjerne til oppdatering av alle andre repliserte kopier (på andre tjenere)

## Oppgave 4 – Distribuerte databasesystemer og distribuert pålitelighet (Distributed database systems and distributed reliability) – 20%

Figuren nedenfor illustrerer tre mulige måter å håndtere oppdateringer av kopier (replicates) på i et distribuert system; multikasting (from an updater to all replicates), gitt unikasting med påfølgende flerkasting (from an updater to a fixed replicate + from this replicate to all other replicates) og fri unikasting med påfølgende flerkasting (from an updater to any replicate + from that replicate to all other replicates).




a) Sammenlign kort de tre angitte måtene å oppdatere kopier på

SVAR:

*Multikasting*

Alle kopier oppdateres øyeblikkelig

*Gitt Unikasting + Flerkasting*

To varianter:
En gitt kopi oppdateres øyeblikkelig, og alle de andre oppdateres deretter (primær/sekundær)
En gitt kopi oppdateres øyeblikkelig, og alle de andre oppdateres senere (herre/tjener)
(ved gitte tidspunkt / etter gitte tidsintervall / på forespørsel)

*Fri Unikasting + Flerkasting*

To varianter:
En vilkårlig kopi oppdateres øyeblik., og alle de andre oppdateres deretter (prim./sekund.)
En vilkårlig kopi oppdateres øyeblik., og alle de andre oppdateres senere (herre/tjener)
(ved gitte tidspunkt / etter gitte tidsintervall / på forespørsel)

b) Drøft kort mulighetene for å oppnå konsistens (consistency) og transparens (transparency) med hver av de tre angitte oppdateringsmåtene

SVAR:

Transparens -
Lett ved multikasting, mulig ved gitt unikasting, vanskelig ved fri unikasting
(og generelt sett ikke 100% transparens og 100% konsistens sammen uansett)

*Multikasting*

Konsistens - Forholdsvis lett å oppnå

*Gitt Unikasting + Flerkasting*

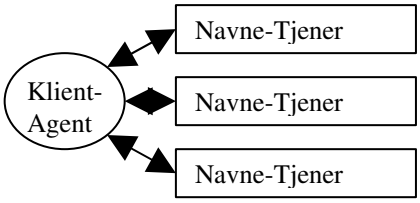
Primær/Sekundær: Konsistens - Mulig å oppnå
Herre/Tjener: Konsistens – Værre å oppnå

*Fri Unikasting + Flerkasting*

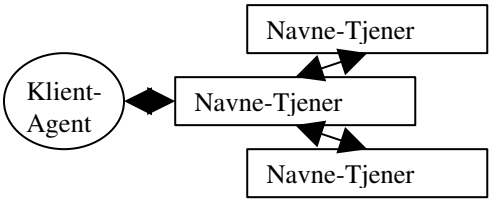
Primær/Sekundær: Konsistens - Vanskelig å oppnå
Herre/Tjener: Konsistens – Nesten umulig å oppnå

**Oppgave 5 – Distribuerte navnetjenester (Distributed name services) – 20%**

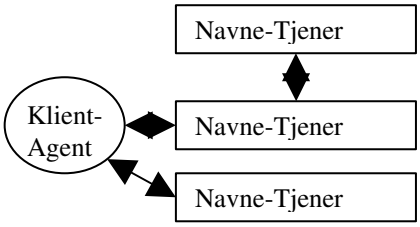
Figuren nedenfor illustrerer fire mulige måter å navigere mot navn på i et distribuert system (resolving multielement names like a.b.c.d …) hvor ulike delnavn kan håndteres av ulike navne-tjenere; mange – iterativt (many, iteratively), en – iterativt (one, iteratively), mange – rekursivt (many, recursively) og en – rekursivt (one, recursively).



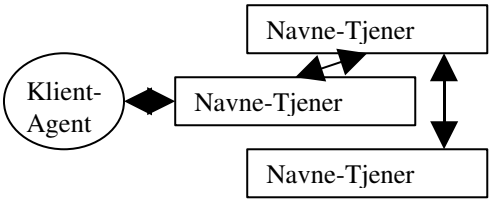

*Mange - Iterativt*




*En - Iterativt*




*Mange - Rekursivt*




*En - Rekursivt*

a) Sammenlign kort de fire angitte måtene å håndtere navn på

SVAR:

*Mange – Iterativt*

Klienten avklarer ett og ett delnavn v.hj.a. hver sin første-tjener

*En – Iterativt*

Klienten kontakter en første-tjener som så avklarer ett og ett delnavn v.hj.a. hver sin andre-tjener

*En – Rekursivt*

Klienten kontakter en første-tjener som avklarer første delnavn før den kontakter en andre-tjener som avklarer andre delnavn før den kontakter en tredje-tjener som avklarer tredje delnavn osv.

*Mange – Rekursivt*

Kombinasjon av Mange-Iterativt og En-Rekursivt

b) Diskuter kort fordeler og ulemper med hver av de fire angitte navigeringsmåtene

SVAR:

*Mange – Iterativt*

-: Klienten blir flaskehals
+: Tillater caching på klienten
-: Tillater ikke sentralisert autorisering
+: Ingen første-tjener blir ”single-failure” utsatt

*En – Iterativt*

+: Klienten blir ikke flaskehals
-: Tillater ikke caching på klienten
-: Første-tjeneren blir flaskehals
+: Tillater caching på første-tjeneren
+: Tillater sentralisert autorisering
-: Første-tjeneren blir ”single-failure” utsatt

*En – Rekursivt*

+: Klienten blir ikke flaskehals
-: Tillater ikke caching på klienten
+: Første-tjeneren blir ikke flaskehals
-: Tillater ikke caching på første-tjeneren
+: Tillater sentralisert autorisering
-: Første-tjeneren blir ”single-failure” utsatt

*Mange – Rekursivt*

-: Klienten kan bli flaskehals
+: Tillater delvis caching på klienten
-: Tillater ikke sentralisert autorisering
+: Er mest fleksibel av alle opplegg