Norges teknisk-naturvitenskapelige universitet

Institutt for datateknikk og informasjonsvitenskap

Løsningsforslag til

EKSAMENSOPPGAVE I FAG TDT4186 – OPERATIVSYSTEMER

Versjon: 13.des 2011

Faglig kontakt under eksamen: Svein Erik Bratsberg

Tlf.: 99539963

Eksamensdato: **9. desember 2011**

Eksamenstid: **15.00-19.00**

Tillatte hjelpemiddel: **D: Ingen trykte eller håndskrevne hjelpemiddel tillatt. Bestemt, enkel kalkulator tillatt.**

Språkform: Bokmål

Sensurdato: 10. januar 2012

Oppgave 1 – Generelt – 5 %

Her er det viktig i en eller annen form å få med:

- Hjelpemiddel for brukervennlighet av datamaskin for sluttbrukere og systemutviklere.
- Sørger for å utnytte datamaskinen effektivt og sikkert, også når det er mange som bruker den.

Oppgave 2 – Prosesser, tråder og synkronisering – 25 %

a) En prosess kan ha mange tråder. Prosessens viktigste egenskap er at den har et eget adresserom, som gir stor isolasjon mellom prosesser. Tråder kan gjerne samarbeide med delt minne. Pluss for hvert av elementene som er tatt med fra tabellen under:

Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

b) Semaforer og mutex'er nært beslektet. En semafor er en synkroniseringsdatastruktur med to tilhørende operasjoner *down()* og *up()*. Den kan brukes til gjensidig utelukkelse og generell venting ved hjelp av den tellende variabelen. En mutex har bare to verdier, låst og åpen. Den er spesiallaget til gjensidig utelukkelse og har to operasjoner: *mutex_lock()* og *mutex_unlock()*.

c) Condition-variabler og mutex'er henger sammen. Når man skal lage en «guard», må en tråd ha en mutex for å kunne legge seg til å vente på en condition-variabel. Men for at en annen tråd skal kunne vekke tråden opp, må tråden gi fra seg mutex'en slik at den andre tråden som skal gjøre betingelsen sann, "kommer inn". I tillegg må den opprinnelige tråden få tilbake mutex'en når betingelsen blir sann.

d) For å få til monitorbegrepet i Java bruker man vanlige javaobjekter og utnytter den innebygde låsen av objektet sammen med den innebygde ventekøen ved å lage synchronized-metoder, eller ved å bruke synchronized-blokker når man aksesserer objektet. Tilhørende metoder for venting er wait() og notify() eller notifyAll(). Det er pluss hvis studenten forklarer hvordan de gjør dette i frisøreksempelet fra øving 2.

e) Korteste jobb først er en måte å minimalisere gjennomsnittlige ventetid på.

0 < Y =< 3: Y, 3, 5, 6, 7

3 < Y =< 5: 3, Y, 5, 6, 7

5 < Y =< 6: 3, 5, Y, 6, 7

6 < Y =< 7: 3, 5, 6, Y, 7

Y > 7: 3, 5, 6, 7, Y

Oppgave 3 – Minnehåndtering – 20 %

a) Relokering av adresser betyr å oversette fra et adresserom til et annet (RAM) når programmet lastes. I sidedelte, virtuelle lager brukes ikke relokering i sin klassiske betydning, men en virtuell adresse i programmet oversettes til en adresse i RAM ved hjelp av sidetabeller. I tillegg vil Translation Lookaside Buffer (TLB) håndtere en stor del av relokeringen.

   Relokering kan også referere til omskriving av adresser ved lenking av program.

b) Fordelene med virtuelt minne i forhold til ingen minneabstraksjon:

   1. tillater større adresserom enn fysisk minne
   2. tillater større grad av multiprogrammering
   3. gjemmer fysiske minne
   4. tillater deling av minne
   5. gir beskyttelse mellom prosesser

c) Ved bruk av TLB (Translation Lookaside Buffer) snakker vi om:

   1. "Soft miss": TLB-avbrudd. Oversettingen finnes ikke i TLB, men finnes i minne. TLB kan oppdateres fra minne.
   2. "Hard miss": Oversettingen finnes ikke i TLB, siden er ikke minne og en sidefeil og en disklesing må til for å få siden inn i minnet.

d) Gitt følgende referansestreng: 0, 1, 3, 1, 4, 6, 5, 2, 4

   Regn ut hvor mange sidefeil du får ved de følgende sideerstatningsalgoritmene når du har tre sider i fysisk lager. Anta minnet er tomt ved start.

   i) LRU: 8

   ii) FIFO: 8

   iii) Optimal: 7

Oppgave 4 – Filsystemer – 20%

a) Filer kan bl.a. implementeres ved de følgende metodene:

i. kontinuerlig allokering:

+ enkel å implementere

+ utmerket leseytelse (brukes på CD-ROM/DVD)

% må vite størrelsen på forhånd

% insert/delete er vanskelig (fragmentering og kompaktering)

ii. lenkede lister

+ god på sekvensiell lesing

+ insert/delete er enkelt

% dårlig på random aksess

% ikke 2^n-størrelse på blokker pga. peker mellom blokker.

% lite pålitelig (når lenkene blir ødelagte)

iii. lenkede lister ved hjelp av en tabell i minne (også kalt FAT)

+ hele blokker tilgjengelig for data

+ går raskt å følge kjeden i RAM

% bør ha hele tabellen i RAM

% må skrives til disk jevnlig

iv. inoder

+ inodetabell: inoden er minne kun når fila er åpen.

+ insert/delete er enkelt

+ sekvensiell og direkteaksess er enkelt

+ kan ha “hull” i fila

% ekstra indireksjonsnivå via inoden

% inoden bestemmer maksimal filstørrelse

b) For kontinuerlig allokering får vi (40000+4)/4096 = 9.8, dvs. vi får **10 blokker** allokert.

For inoder: Write 0 havner i blokk 0, 20000 havner i blokk 5, 30000 havner i blokk 7 og 40000 i blokk 9. Blokk 9 må indekseres via indirekteblokka, som også må allokeres. Da blir det **5 blokker** allokert på disken, i tillegg til **inoden**.

Oppgave 5 – I/O – 10%

a) Vi får 6*400/60 = 40 avbrudd per sekund. Avbruddshåndtering utgjør da 40*20 = 800 mikrosekunder av hvert sekund, noe som er 0.08 % av CPU-tiden.

b) I/O-enheter deles inn i blokk- og tegnenheter fordi de to typene har forskjellige egenskaper og derfor forskjellig grensesnitt. Operativsystemet har en del programkode som er felles for hver klasse av I/O-enheter. Blokkenheter har "seek"-grensesnitt og adresseres blokkvis. Tegnenheter er strømorienterte. Det er også et ytelsesargument, noen devicer gir avbrudd per tegn, mens dette ville vært ugunstig for de som leverer blokker.

Oppgave 6 – Vranglås – 10%

a) Betingelsene for vranglås:

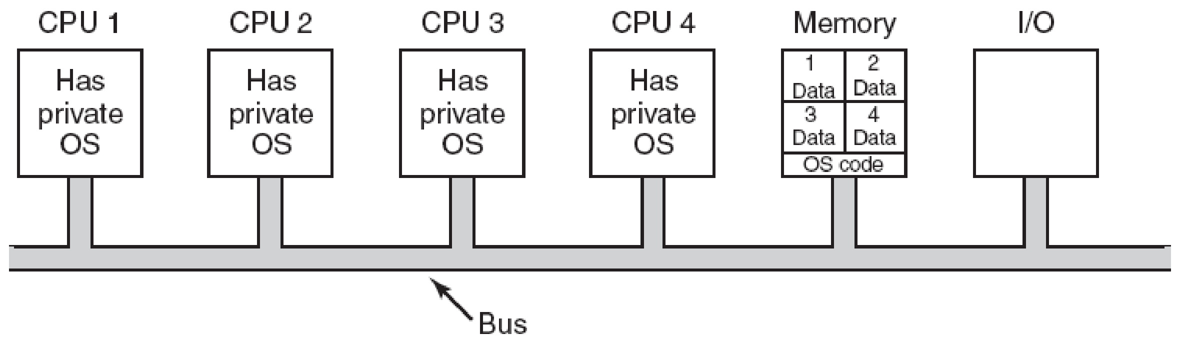
1. gjensidig utelukkelse av ressurser.
2. hold og vent på ressurser
3. ikkeavbrytbare ressurser
4. sirkulær venting

b) Her angriper vi hver av betingelsene.

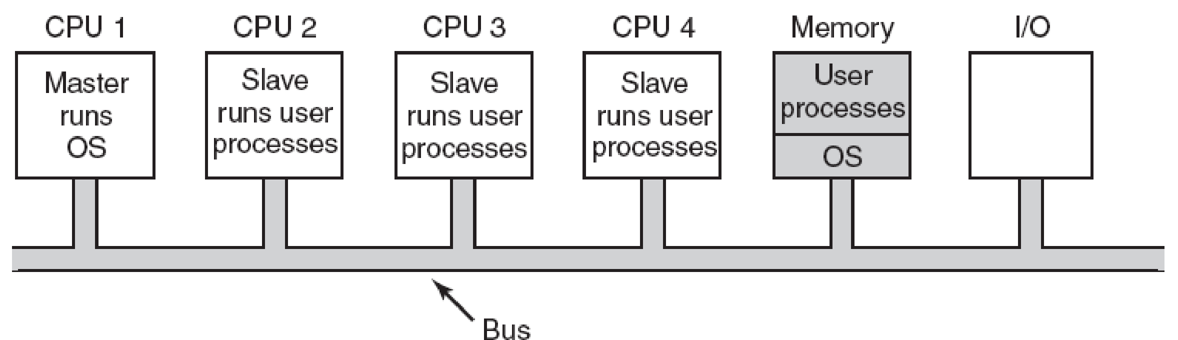
1. bruk spooling
2. spør om alle ressurser du trenger før du starter
3. kan være umulig for mange ressurser
4. spør om ressurser i samme rekkefølge

Oppgave 7 – Multiprosessorer og virtualisering – 15 %

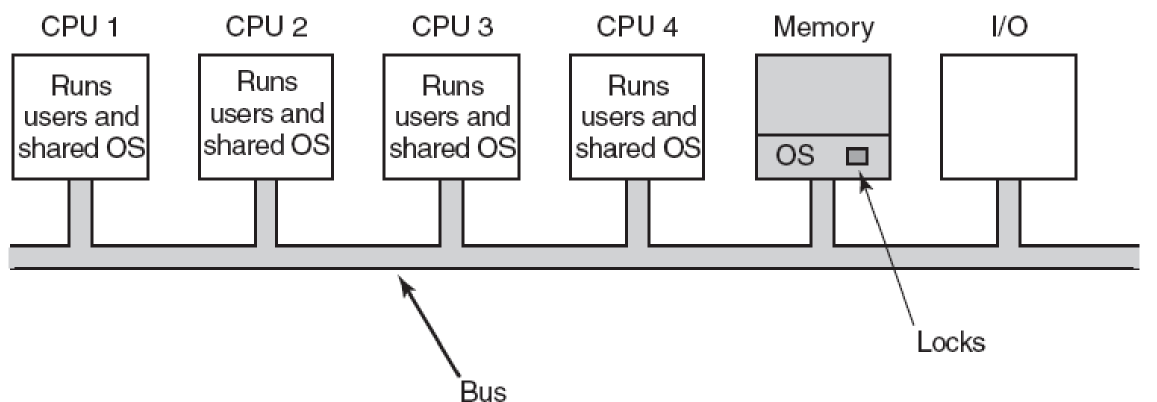
a) 1. Hver CPU har sitt eget OS.



2. Master-slave OS.



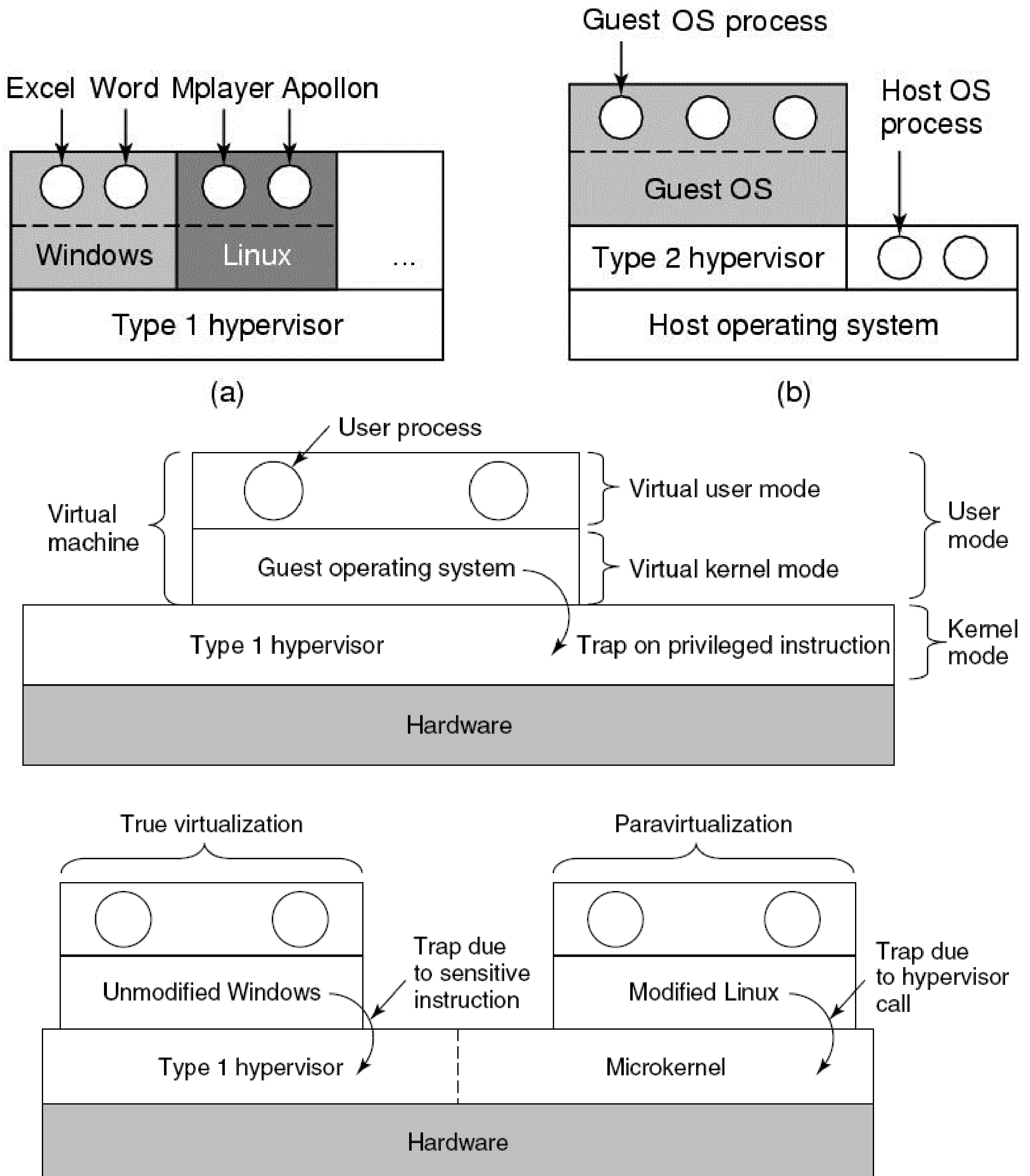
3. Scalable multiprocessor (SMP).



b) Det mest spesielle er at tidsdelingen foregår både i tid og *rom*, dvs. vi må også bestemme hvilken CPU som skal kjøre tråden. Dette gjør at gruppevis tidstildeling blir aktuelt. I tillegg er det viktig at en tråd/prosess tildeles samme CPU slik at cache evt. kan gjenbrukes.

Følgende metoder er nevnt i boka. 1. tidsdeling med delt kø. 2. affinity scheduling (tilordne CPU ved prosessoppretting) 3. space sharing (grupper av tråder som kjører samtidig uten multiprogrammering). 4. gang scheduling (relaterte tråder kjører samtidig med felles timeslice).

c) Forklar forskjellen på Type 1 hypervisorere, Type 2 hypervisorere og paravirtualisering.



Oppgave 8 – Sikkerhet – 5 %

Det følgende er nevnt i læreboka, men er nok ikke alt som finnes:

- Kan finne feil og utnytte dem

1. portscan for å finne datamaskiner som aksepterer login

2. prøv login med kombinasjoner av brukernavn/passord

3. når du er kommet inn, kjør programmet som utnytter feilen

4. lag et SETUID-root-shell

5. hent og start et program som lytter på en port og kjører kommandoer

6. sørg for at programmet alltid starter når systemet bootes

- **Bufferoverflow-angrep**: Legg inn kode forkledd som strenger som input til et program

- **Formatteringsstreng-angrep**: Ubeskyttet input av tekst kan la brukeren endre koden i et program

- **"Retur fra libc"-**angrep

- **"Integer-overflyt"-**angrep

- **"Codeinjection"-**angrep: Legg inn kode i input

- **Privilegieeskaleringsangrep**: Infiltrer en daemon som kjører som root.