Norges teknisk-naturvitenskapelige universitet

Institutt for datateknikk og informasjonsvitenskap

Løsningsforslag til

EKSAMENSOPPGAVE I FAG TDT4186 – OPERATIVSYSTEMER

Versjon: 17.jan 2013

Faglig kontakt under eksamen: Svein Erik Bratsberg

Tlf.: 99539963

Eksamensdato: **10. desember 2012**

Eksamenstid: **9.00-13.00**

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

Språkform: Bokmål

Sensurdato: 11. januar 2013

Oppgave 1 – Generelt – 5 %

Operativsystemet er en ressurshåndterer og abstrakt maskin. Dette er mye lettere å programmere en abstrakt maskin enn en fysisk maskin. Operativsystemet tilbyr et pent grensesnitt som er lett å forholde seg til. I tillegg håndterer operativsystemet ressursene slik at det ikke blir konflikter mellom forskjellige programmer eller brukere omkring bruken av delte ressurser.

Oppgave 2 – Tråder, synkronisering og tidsdeling – 20 %

a) Kodesnutten er hentet fra bokas eksempel på Producer/Consumer implementert ved hjelp av pthreads. Her er det consumeren som venter på at bufferet inneholder data. Først henter tråden en mutex. Når den har kommet inn, så tester den om bufferet har noe data. Hvis det ikke er noe data der, legger tråden seg til å vente på en condition-variabel. I det samme kallet blir mutex'en da sluppet. Tråden venter altså "utenfor monitoren" her. Dette gjøres i en while-løkke, slik at det blir en "guarded wait", dvs. når tråden vekkes opp, sjekker den med en gang betingelsen (buffer==0) på nytt. Etter å ha behandlet data, signalerer den en condition-variabel og slipper mutexen.

b) "Guarded wait" er at man sjekker en betingelse (f.eks. er bufferet tomt?) ved venting, og så skal man sjekke opp betingelsen på nytt når man blir vekket opp fra ventingen. I a) er det brukt en "guarded wait". Mange studenter har sett denne. Grunnen til å bruke en slik løsning er at en annen tråd kan ha endret betingelsene etter "du" har sovet og du må være sikker på at tilstanden er riktig når du skal gå videre på innsiden.

c) Denne oppgaven er hentet fra øving 3. Prosessene som står i klarkøa til CPUen blir tidsdelt ved hjelp av round-robin. Spørsmålet er da om tidskvantumet som brukes i round-robin-algoritmen påvirker throughputen til systemet. Her er det tre mulig svar. 1. Generelt kan vi si at overheaden ved prosesskifte påvirker throughputen, slik at en lang tidskvant kan hindre ofte prosesskifte og øke throughputen. 2. Men i øvingen var det ingen overhead ved prosesskifte slik at **ingen påvirking** er svaret da. 3. Det er også et tredje mulig svar. Øvingen antok at en prosess ikke kunne gi fra seg CPUen midt i et tidskvantum, men at tidskvantumet må fullføres uten noe CPU-bruk. Da vil et langt kvantum være ugunstig for gjennomstrømningen.

Oppgave 3 – Minnehåndtering og ressursbruk – 10 %

a) Hjelper ingenting med større disk. Det gir bare mer plass til swap.

b) Hvis vi øker graden av multiprogrammering, dvs. flere prosesser kjører, blir det enda større minnebehov og enda mer paging og CPU-utnyttelsen går enda mer ned. Mange studenter har her referert til formelen CPU-utnyttelse = $1 - p^n$ som viser sammenhengen mellom I/O-grad og antall prosesser. p er andelen av tiden en prosess venter på I/O. Denne modellen forutsetter at alle prosesser er i minnet og kan ikke brukes her.

c) Ved å installere mer fysisk minne blir det mindre paging og CPU-utnyttelsen går opp.

Oppgave 4 – Minnehåndtering – 15 %

a) 64 bits adresserom betyr at sidetabellene blir veldig store, kanskje større enn fysisk minne. Dette kan løses ved å innføre multinivå sidetabeller eller ved å bruke inverterte sidetabeller.

b)

i) NRU 2

ii) FIFO 3

iii) LRU 1

iv) Second chance 2.

c) Aging er en tilnærming til LRU. LRU holder eksakt oversikt over Least Recently Used-relasjonen mellom forskjellige sider, mens Aging registerer kun bruk innenfor et "klokketikk", slik at alle innenfor tikket er brukt samtidig. I tillegg har aging en begrensning i hvor lenge den husker bruk ved antall bits som brukes i agingregisterne.

Oppgave 5 – Filsystemer – 15%

a) Se figur 10-34 i læreboka. Som UNIX V7. Hver fil er implementert med en *inode* med 10 direktepekere, en indirekte peker, en dobbelt, indirekte peker og en trippelt, indirekte peker.

b) Antar en diskaksess for å lese katalog.

i) Pluss 25 diskaksesser for å lese alle blokkene fram til og med 100000+100 bytes. Antar det er plass til 4092 bytes med 4 bytes til blokkpeker i hver blokk.

ii) Pluss 3 diskaksesser da det holder å skanne gjennom FAT-tabellen i minnet for å finne fram til de forskjellige adressene.

Oppgave 6 – I/O – 10%

a) - implementere funksjoner som read, write, get og put

- håndtere strømforsyning (spare strøm)

- logge hendelser

- se etter og korrigere feil

b) - vedlikeholde veggklokke

- hindre prosesser i å kjøre for lenge

- CPU-regnskap for prosesser

- callback-timers (alarmer)

- watchdog - overvåk hardware

- profilering - hvem bruker CPuen?

Oppgave 7 – Multiprosessorer og virtualisering – 15 %

a) "Cache coherence protocol": Dette er noen regler som brukes ved caching av data i en distribuert kontekst. I læreboka er dette brukt i forbindelse med UMA-multiprosessorer med bussbasert arkitektur. Her kan cachelinjer fra minne caches i hver CPU. Hver cachelinje markeres som read-only eller read-write. Hvis en CPU skriver til en cachelinje som finnes i en eller flere cacher rundtomkring, så settes det et signal på bussen om dette. De som har en "clean copy" kan bare kaste denne. Hvis noen har en "dirty copy" må denne skrives til minne før den som skal skrive får lest cachelinja fra minnet.

b) 4 løsninger:

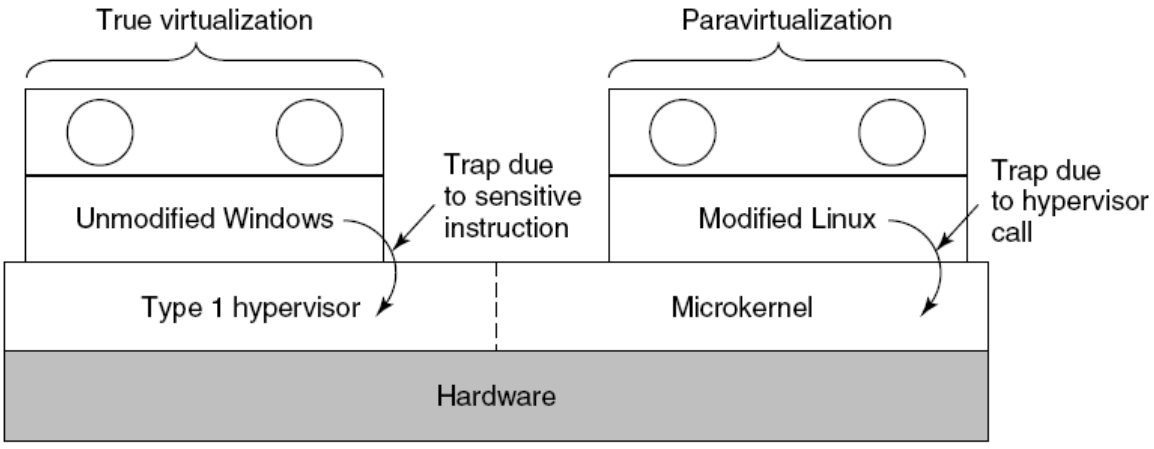
1. bruk kun blokkerende send

2. Ikke-blokkerende send med kopi av meldingsbufferet

3. Ikke-blokkerende send med avbrudd når meldingen er sendt

4. Ikke-blokkerende send og bruk av copy-on-write av meldingsbufferet

c) **Type 1**: trap til hypervisoren når gjesteoperativsystemet prøver å utføre en sensitiv instruksjon. **Type 2**: Sensitive instruksjoner blir emulert av hypervisoren. Såkalt binary translation. **Paravirtualisering**: Sensitive instruksjoner blir erstattet av systemkall til hypervisoren/mikrokjerna.



Oppgave 8 – Sikkerhet – 10 %

- Trojansk hest: Gjem malmware i en annen innpakning (program)

- Virus: Et program som kan installere seg selv ved å henge seg på et annet program
    1. minneresidente: Gjemmer seg i avbruddsrutiner
    2. boot sector-virus
    3. device driver-virus
- Worms: Selvreplikerende program som sprer seg selv via nettet
- Spyware: Program som gjemmer seg og samler informasjon.
- Rootkits: Program som gir root-aksess og gjemmer seg.