NTNU
Kunnskap for en bedre verden

Institutt for Datateknikk of Informasjonsvitenskap

Løsningsforslag for TDT4186 Operativsystemer

Eksamensdato: 7. juni 2016

Eksamenstid (fra-til): 09:00-13:00

Hjelpemiddelkode/Tillatte hjelpemidler:

D: Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.

Annen informasjon:

Det ønskes korte og konsise svar på hver av oppgavene.
Les oppgaveteksten meget nøye, og vurder hva det spørres etter i hver enkelt oppgave/deloppgave.
Dersom du mener at noen opplysninger mangler i oppgaveformuleringene, beskriv de antagelsene du gjør.
Hver av de fem oppgavene teller like mye, og for hver av de seks deloppgavene igjen er den relative vektingen angitt i prosent.

Oppgave 1: Operativsystemer (Operating Systems) / Prosesser og tråder (Processes and Threads)

a) Drøft kort hvorfor operativsystemutviklere må forstå en maskins avbruddssytem (interrupt system) (15%)

SVAR:

Avbruddssystemet har betydning for både funksjonalitet og effektivitet av datamaskin-systemer, og det skaper således både utfordringer og muligheter ved utvikling av operativsys-temer.

b) Angi klart forskjellene mellom operativsystemorganiseringen i tidlig UNIX og prosess-tråd modellen (15%)

SVAR:

Tidlig UNIX tillot felles data å bli brukt uten mye kontroll på tvers av prosesser; det gjør prosess-tråd modellen ikke.

I prosess-tråd modellen reserverer en prosess felles plass som brukes av dens tråder; tråder fantes ikke i tidlig UNIX.

c) Angi klart forskjellene mellom multikjerner (multi cores) og mikrokjerner (micro kernels) (15%)

SVAR:

Multikjerner angir at en har flere miniprosessorer tilgjengelig, mens en mikrokjerne tilsier en bestemt måte å organisere et operativsystem på.

Således er det to helt forskjellige begreper; det er bare de norske navnene som antyder noe felles.

d) Drøft kort hva kjernetråder (kernel level threads) er og brukes til (15%)

SVAR:

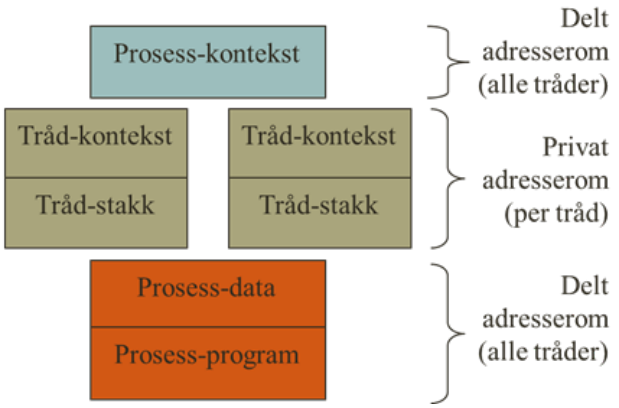
Tråder er miniprosesser som skiller ressurseierskap og CPU-bruk.

Kjernetråder implementeres i operativsystemkjernen slik at slike tråder er kjent for operativ-systemet og kan utnyttes som sådann mht funksjonalitet og ytelse.

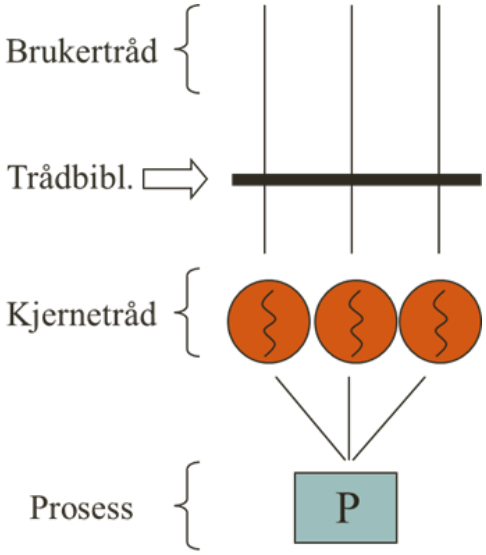
e) Angi konkret hvordan kjernetråder kan implementeres (25%)

SVAR:

Ved etablering av en prosess med et sett av tilhørende tråder, reserveres følgende dataområder for henholdsvis prosessen og trådene:



Kjernetråder implementeres da i operativsystemkjernen og ikke i et språkbibliotek som brukertråder gjør:



En generalisering av den angitte 1*M kombinasjonen mellom prosess og tråd, er en N*M kombinasjon, mens en spesialisering av den angitte 1*M kombinasjonen mellom prosess og tråd, er en N*1 kombinasjon.

f) Drøft hvor gode resultater en kan oppnå med kjernetråder sammenlignet med andre alternativ(er) (15%)

SVAR:

Hovedalternativet til kjernetråder er altså brukertråder som implementeres i språkbibliotek.

Kjernetråder vil kunne utnytte multiprosessorer og tilbyr ikke-blokkerende systemkall, men gir også dyre systemkall. Brukertråder gir billige systemkall, men tilbyr kun blokkerende systemkall og vil ikke kunne utnytte multiprosessorer.

Oppgave 2: Synkronisering av prosesser (Process Synchronization)

a) Drøft kort om vi helt kan unngå aktiv bruk av CPU-kraft ifm. synkronisering av prosesser / tråder (15%)

SVAR:

Det er en viss mulighet hvis vi bare har en prosessor med en kjerne; da kan vi slå av avbrudds-systemet ved behov for å oppnå kritiske regioner.

Men når vi har flere prosessorer/kjerner og/eller skal løse mer generelle tidsordningsoppgaver enn bare å etablere kritiske regioner, vil en viss aktiv venting uansett trenges på et lavere nivå.

b) Angi klart om Cnotify og Cbroadcast gir samme effekter ifm. bruk av Mesa-monitorer (15%)

SVAR:

Cnotify kan gjenåpne maks en prosess/tråd, mens Cbroadcast kan gjenåpne opptil flere prosesser/tråder, og således har de to mekanismene ulike effekter.

c) Angi klart om vranglås (deadlock) er et større problem enn utsulting (starvation) ifm. synkroniserings-behov (15%)

SVAR:

De to problemområdene har med forskjellige utfordringer å gjøre, og de kan i utgangspunktet ikke sammenlignes. Men vranglås vil typisk påvirke mange prosesser/tråder, mens utsulting typisk vil påvirke få prosesser/tråder. Således kan en argumentere for at vranglås utgjør en større utfordring enn utsulting.

d) Drøft kort hva Bankier-algoritmen (Banker´s Algoritm) er og brukes til (15%)

SVAR:

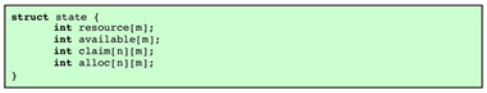
Bankier-algoritmen er en metode for å håndtere vranglås.

Den er av en undergruppe som unngår dem gjennom å sikre seg at de ikke vil skje - med fokus på tidspunktet for allokering av ressurser.

e) Angi konkret hvordan Bankier-algoritmen kan implementeres (25%)

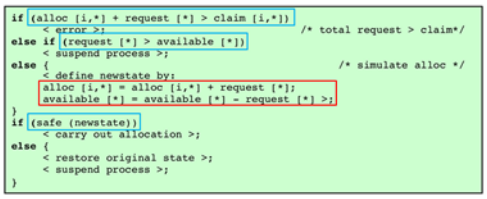
SVAR:

Datastrukturer og algoritmer som trengs er:



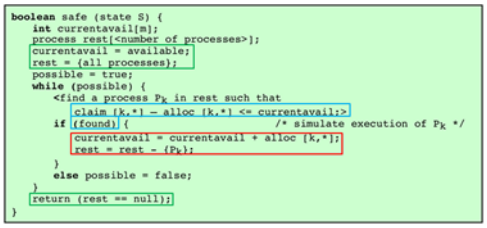
```
struct state {
      int resource[m];
      int available[m];
      int claim[n][m];
      int alloc[n][m];
}
```

(a) global data structures



```
if (alloc [i,*] + request [*] > claim [i,*])
     < error >;                                  /* total request > claim*/
else if (request [*] > available [*])
     < suspend process >;
else {                                           /* simulate alloc */
     < define newstate by:
     alloc [i,*] = alloc [i,*] + request [*];
     available [*] = available [*] - request [*] >;
}
if (safe (newstate))
     < carry out allocation >;
else {
     < restore original state >;
     < suspend process >;
}
```

(b) resource alloc algorithm



```
boolean safe (state S) {
   int currentavail[m];
   process rest[<number of processes>];
   currentavail = available;
   rest = {all processes};
   possible = true;
   while (possible) {
      <find a process Pk in rest such that
         claim [k,*] - alloc [k,*] <= currentavail;>
      if (found) {                              /* simulate execution of Pk */
         currentavail = currentavail + alloc [k,*];
         rest = rest - {Pk};
      }
      else possible = false;
   }
   return (rest == null);
}
```

(c) test for safety algorithm (banker's algorithm)

Datastrukturene initialiseres en gang, mens ressursallokeringsalgoritmen kjøres hver gang en prosess/tråd ønsker å allokere mer ressurser.

f) Drøft hvor gode resultater en kan oppnå med Bankier-algoritmen sammenlignet med andre alternativ(er) (15%)

SVAR:

Hovedalternativet til Bankier-algoritmen innen undergruppen av dem som unngår dem gjennom å sikre seg at de ikke vil skje - er en med fokus på tidspunktet for initiering av prosesser/tråder som senere vil kreve ressurser. Dette er en ganske grovkornet strategi som er mye mindre praktisk anvendbar enn Bankier-algoritmen.

Alternativene til undergruppen som unngår vranglås, er en undergruppe som umuliggjør vranglås - ved å sikre seg at de faktisk ikke kan skje, og en undergruppe som oppdager og retter opp vranglås - ved først å la dem skje i seg selv. Umuliggjøring av vranglåser og unngåelse av vranglåser tilsier ressurskrevende systemer for at vranglåser ikke skal skje og derfor mest aktuelle der vranglåser forekommer ofte, mens oppdaging og oppretting av vranglåser tilsier ressurskrevende prosesser for å håndtere vranglåser som har skjedd og derfor mest aktuelle der vranglåser forekommer sjeldent.

Oppgave 3: Håndtering av lager (Memory Management)

a) Drøft kort om det er viktig å ha både god maskinvarestøtte og god programvarestøtte ifm. lagerhåndtering (15%)

SVAR:

Noen oppgaver må utføres veldig ofte, og de må da løses i maskinvare. Heldigvis er dette forholds enkle oppgaver som da også kan utføres i maskinvaren. Mens andre oppgaver er mer

komplekse, og de må da løses i programvare. Heldigvis er dette forholdsvis sjeldne oppgaver som da også kan utføres i programvaren.

Således er en avhengig å ha både god maskinvarestøtte og god programvarestøtte.

b) Angi klart om virtuelt lager (virtual memory) er en naturlig måte å utnytte lokalitetsprinsippet (locality principle) på (15%)

SVAR:

En slik kombinering av primærlager og sekundærlager er en både god og nødvendig utnyttelse av lokalitetsprinsippet. Det tillater kjøring av både flere og for store program - på en naturlig måte. Cache(r) i kombinasjon med primærlager utgjør en annen lignende naturlig måte å utnytte det på.

c) Angi klart om Buddysystem-modellen bare er av teoretisk interesse ifm. lagerhåndtering (15%)

SVAR:

Den brukes bl.a. i LINUX-systemer, og den brukes både for brukerdata og systemdata; ergo er den i bruk i praktisk lagerhåndtering.

d) Drøft kort hva LRU- & LFU- (Least Recently Used & Least Frequently Used) algoritmene er og brukes til (15%)

SVAR:

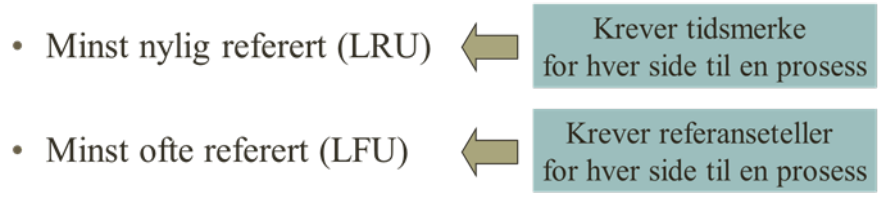
LRU og LRU er algoritmer for sideutbytting.

De brukes til å finne fram til hvilken eksisterende sides ramme det er best å velge når en ny side må hentes inn fra sekundærlageret og det ikke er mer plass i primærlageret.

e) Angi konkret hvordan LRU- & LFU-algoritmene kan implementeres (25%)

SVAR:

Databehovene er som følger:



Tidsmerket for en gitt side, som angir når en side ble aksessert sist - og gjerne er implementert vhja. en sidestakk / referansetelleren for en gitt side, som angir hvor ofte en side har blitt aksessert i et gitt intervall - og gjerne er implementert vhja. en sidetabell, oppdateres ved hver

aksess av siden. Ved behov for sideutbytting velges da den siden som det er lengst siden har blitt referert / som har blitt referert færrest ganger i gjeldende intervall.

f) Drøft hvor gode resultater en kan oppnå med LRU- & LFU-algoritmene sammenlignet med andre alternativ(er) (15%)

SVAR:

LRU & LFU gir gode resultater, men de er særs ressurskrevende i praktisk bruk for sideutbytting ifm virtuelt lager. FIFO (Først Inn – Først Ut) er en enkel algoritme som gir dårligere resultater enn både LRU & LFU, mens OPT (Optimal) er en ideell algoritme som bare fungerer som en teoretisk målestokk. To algoritmer som er mindre ressurskrevende i praktisk bruk og som gir brukbare resultater, er U-CLOCK (2-sjansers klokke) & UM-CLOCK (4-sjansers klokke).

LRU & LFU er mer aktuelle for eksplisitt beordret I/O enn for implisitt påført I/O ifm virtuelt lager, men da gjerne også i en kombinasjon - som algoritmen FBS (Frekvensbasert stakk).

Oppgave 4: Tidsstyring av prosesser (Process Scheduling)

a) Drøft kort om de vanlig brukte kriteriene for tidsstyring er uavhengige av hverandre (15%)

SVAR:

Disse kriteriene deles gjerne langs en dimensjon opp i noen med brukerfokus og noen med systemfokus, og de deles gjerne langs en annen dimensjon opp i noen med ytelsesorientering og noen uten ytelsesorientering.

Innen hver av de fire resulterende gruppene er det flere avhengigheter; ergo kan en vanligvis ikke velge et bestemt mål helt fritt uten også å påvirke et annet mulig mål.

b) Angi klart om de samme algoritmene brukes for tidsstyring med multiprosessorer (multi processors) og med multikjerner (multi cores) (15%)

SVAR:

Hvis en har fokus på bare utnyttelse av prosesseringskraft, kan gjerne lignende algoritmer brukes.

Men hvis en har fokus på andre aspekter som f.eks. tilgang til felles lageraksess, blir gjerne andre algoritmer valgt for multikjerner enn for multiprosessorer.

c) Angi klart om invertering av prioriteter (priority inversion) bare er en teoretisk løsning ifm. tidsstyring (15%)

SVAR:

Invertering av prioriteter som problem løses gjerne med to lignende mekanismer - prioritetsarving og/eller prioritetstak, og dette er i praktisk bruk i både UNIX- og WINDOWS-systemer.

d) Drøft kort hva rettmessig tidsstyring (Fair Share scheduling) er og brukes til (15%)

SVAR:

Rettmessig tidsstyring er et sett med algoritmer som tildeler CPU-tid i overensstemmelse med en forhåndsdefinert vekting av prosesser/tråder.

Slike algoritmer prøver å balansere CPU-tildelingen over tid ved i et gitt intervall å gi mindre CPU-ressurser til en prosess/tråd som i tidligere intervaller har hatt mer CPU-ressurser enn vektingen skulle tilsi og tilsvarende å gi mer CPU-ressurser til en prosess/tråd som i tidligere intervaller har hatt mindre CPU-ressurser enn vektingen skulle tilsi.

I mange (UNIX-)systemer kombinerer en ofte dette med håndtering av grupper av prosesser/tråder, men det ligger ikke til konseptet rettmessig tidsstyring som sådann

e) Angi konkret hvordan rettmessig tidsstyring kan implementeres (25%)

SVAR:

En vanlig (gruppebasert) implementasjon definerer følgende størrelser:

Definisjoner

W_k = Vekt til $Grup_k$

$U_j(i)$ = CPU-bruk av $Pros_j$ i Int_i

$CPU_j(i)$ = CPU-hale for $Pros_j$ før Int_i

$GU_k(i)$ = CPU-bruk av $Grup_k$ i Int_i

$GCPU_k(i)$ = CPU-hale for $Grup_k$ før Int_i

BP_j = Basis Prio til $Pros_j$

$P_j(i)$ = Effektiv Prio for $Pros_j$ før Int_i

En slik (gruppebasert) implementasjon gjør følgende beregninger ved starten av hvert intervall:

Beregninger

$$\Sigma W_k = 1;\ k: 1..M$$

$$CPU_j(i) = [U_j(i-1) + CPU_j(i-1)]/2$$

$$GCPU_k(i) = [GU_k(i-1) + GCPU_k(i-1)]/2$$

$$P_j(i) = BP_j + CPU_j(i)/2 + GCPU_k(i)/(4*W_k)$$

Den prosess/tråd *i* som har lavest P-verdi ved starten av intervall *j* kjøres da.

f) Drøft hvor gode resultater en kan oppnå med UNIX sin vanligste rettmessig tidsstyring sammenlignet med andre alternativ(er) (15%)

SVAR:

I UNIX-systemer med rettmessig tidsstyring nedvektes CPU-tid brukt i tidligere intervaller logaritmisk med hvor langt et gitt intervall er fra nåtid. Denne logaritmiske nedvektingen har sammen med en basisprioritet gitt til en prosess/tråd i henhold til ønsket vekting vist seg å gi en brukbar god tilpasning til den ønskete vektingen selv om det ikke blir helt nøyaktig over kortere tidsintervaller.

Et alternativ til Rettmessig tidsstyring med algoritmisk nedvekting er Garantert tidsstyring uten algoritmisk nedvekting. Det kan også gi brukbare resultater med forholdsvis stabile vektinger, men ikke like gode resultater med vektinger som varierer over tid.

Oppgave 5: Håndtering av I/O (I/O Management)

a) Drøft kort om virtuelt minne (virtual memory) implementeres ved hjelp av disklagerplass (15%)

SVAR:

Det implementeres med en kombinasjon av så vel sekundærlagerplass som primærlagerplass.

Nødvendig sekundærlagerplass kan være disklagerplass eller tilsvarende, men det vanligste er å bruke disklagerplass.

b) Angi klart om RAID-konseptet kan implementeres i både maskinvare og programvare (15%)

SVAR:

Den opprinnelige måten å implementere det på var i maskinvare, men senere begynte en også å implementere det funksjonelt i programvare - selv om dette har konsekvenser mht. ytelse.

En del WINDOWS-systemer implementerer det f.eks. nå i programvare.

c) Angi klart om plassallokering (file allocation) og plasstilsyn (free space management) på disk implementeres på samme måter (15%)

SVAR:

For kjedete enheter/blokker og indekserte enheter/blokker implementeres begge konsepter på lignende måter.

Men for sammenhengende enheter brukes bittavmerkete blokker i stedet.

d) Drøft kort hva filsystemer (file systems) er og brukes til (15%)

SVAR:

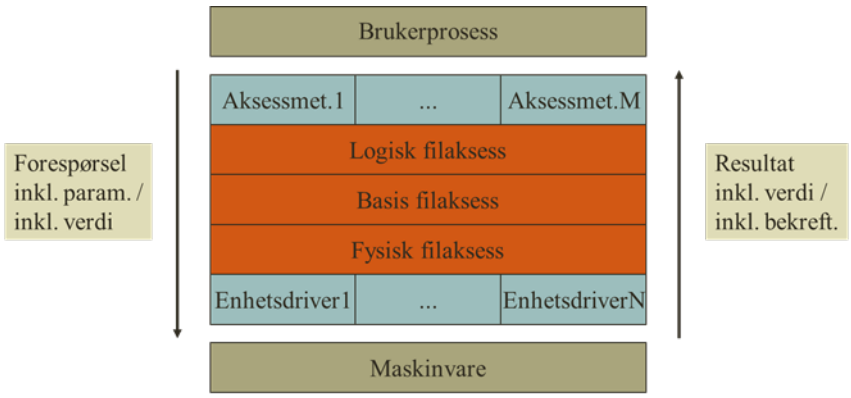
Filsystemer er en delvis integrert del av et operativsystem.

Det brukes til lagring av informasjon på permanent basis - med ulike muligheter for rettighetsangivelse, gjenfinning, utvelgelse og fremvisning etc. av tilhørende data.

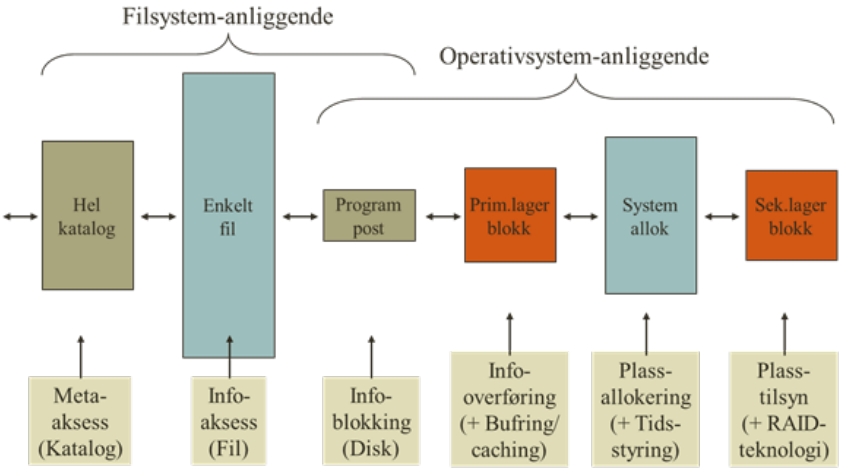
e) Angi konkret hvordan filsystemer kan implementeres (25%)

SVAR:

Filsystemer implementeres gjerne lagdelt som følger:



Komponenter som implementeres henholdsvis innenfor og utenfor operativsystemkjernen er som følger:



Ved implementasjon må en både balansere hva som skal ligge henholdsvis innenfor og utenfor operativsystemkjernen – og balansere den funksjonelle fleksibiliteten med det ytelsesmessige resultatet.

f) Drøft hvor gode resultater en kan oppnå med WINDOWS sine vanligste filsystemer sammenlignet med andre alternativ(er) (15%)

SVAR:

Filsystemers funksjonalitet og ytelse er vanskelig å sammenligne direkte.

WINDOWS-systemer tilbyr uansett både god funksjonalitet - med stor fleksibilitet mht. bl.a. filstørrelser og aksesseringsmåter, og god ytelse - med stort fokus på bl.a. uthentings-effektivitet og datasikring – sammenlignet med flere andre leverandørers filsystemer. Men kostnaden er dog at de er store og komplekse filsystemer – igjen sammenlignet med flere andre leverandørers filsystemer.