NTNU
Kunnskap for en bedre verden

Institutt for Datateknologi og Informatikk

Løsningsforslag for TDT4186 Operativsystemer

Eksamensdato: 23. mai 2017

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 deloppgave.
Dersom du mener at noen opplysninger mangler i oppgaveformuleringene,
beskriv de antagelsene du gjør.
Hver av 24 deloppgavene teller like mye.

Oppgave 1: Operativsystemer generelt (Operating Systems in General)

a) Hva er karakteristiske trekk ved moderne operativsystemer –
og hvorfor struktureres og implementeres moderne operativsystemer slik?

SVAR:

Moderne operativsystemer er gjerne

- Mikrokjernebaserte &
- Objektorienterte &
- Trådbaserte,

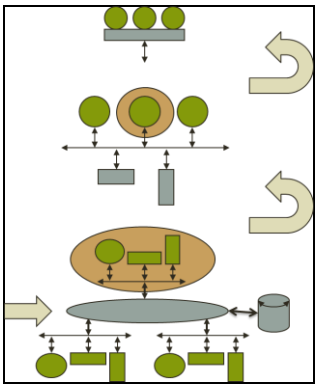
da det ofte er den optimale måten å tilby tjenester så enkelt som mulig og forvalte ressurser så effektivt som mulig per nå. De kan gjerne også

- Virke i en distribuert setting &
- Håndtere flere prosessorer per system &
- Håndtere flere kjerner per prosessor,

da det ofte trengs for å tilpasse seg moderne datamaskinarkitekturer.

b) Hva er forskjellene på et multikjerne system (Multicore System), et multiprosessor system (Multiprocessor System) og et distribuert system (Distributed System)?

SVAR:



Et distribuert system er en samling av datamaskiner forbundet i et nettverk, mens et multiprosessor system har flere prosessorer til å håndtere prosesseringslasten, og i et multikjerne system er det flere prosesseringskjerner i hver prosessor.

Oppgave 2: Prosesser og tråder (Processes and Threads)

a) Hvorfor trengs prosesskonseptet i operativsystemer?

SVAR:

En prosess representerer et program / en applikasjon under utførelse, og prosesskonseptet er en god måte å tillate og håndtere kjøring av flere programmer / applikasjoner samtidig.

b) Hvorfor brukes trådkonseptet i prosessorganisering?

SVAR:

En tråd er en miniprosess som kan initieres, kjøres og initieres på en tidsbesparende og plassbesparende måte, og trådkonseptet tillater et skille mellom eierskap til ressurser - knyttet til prosesser, og kjøring av delprogrammer / delapplikasjoner på en CPU – knyttet til tråder.

Oppgave 3: Synkronisering av prosesser (Process Synchronization)

a) Hvilke problemer brukes synkroniseringsverktøy til å løse?

SVAR:

Synkroniseringsverktøy trengs for å hindre at parallell utføring av programmer / delprogrammer gir gale resultater – slik at ulike enkeltoperasjoner kan ordnes i riktig rekkefølge / at ulike sett av operasjoner kan utføres uten overlapp.

b) Når bør en bruke henholdsvis semaforer (Semaphores), monitorer (Monitors) og meldinger (Messages) som verktøy?

SVAR:

De tre verktøyene er funksjonelt sett likeverdige ved at de kan løse de samme problemene et system med delt lager, men mht anvendelighet er det skiller:

- Bare meldinger kan brukes både i sentraliserte og distribuerte settinger
- Meldinger kan implementere generell tidsordning svært direkte
- Monitorer vil implementere gjensidig utelukkelse automatiske
- Semaforer er et lavnivåverktøy som en ofte implementerer andre høynivåverktøy med

c) I både læreboken og forelesningene diskuteres følgende implementasjon av en bundet buffer (Bounded Buffer) v.hj.a. monitor:

```
monitor boundedbuffer;

char buffer [N];
int nextin, nextout, count;
cond notfull, notempty;

void append (char x)
{  if (count == N) cwait (notfull);
   buffer [nextin] = x;
   nextin = (nextin + 1) % N;
   count++;
   csignal (notempty);  }

void take (char x)
{  if (count == 0) cwait (notempty);
   x = buffer [nextout];
   nextout = (nextout + 1) % N;
   count--;
   csignal (notfull);  }

{  nextin = 0; nextout = 0; count = 0;  }
```

Hvilken type monitor anvendes her – vanlig eller MESA, og hva er begrunnelsen for svaret ditt?

SVAR:

Her anvendes den vanlige monitortypen. Det reflekteres i både testmekanismen («if» og ikke «while») og i signaleringsformen («csignal» og ikke «cnotify»).

d) Hvilke konkrete endringer trengs i koden i figuren over ved anvendelse av den andre type monitor, og hva er begrunnelsen din for det?

SVAR:

```
monitor boundedbuffer;

char buffer [N];
int nextin, nextout, count;
cond notfull, notempty;

void append (char x)
{  while (count == N) cwait (notfull);
   buffer [nextin] = x;
   nextin = (nextin + 1) % N;
   count++;
   cnotify (notempty);  }

void take (char x)
{  while (count == 0) cwait (notempty);
   x = buffer [nextout];
   nextout = nextout + 1) % N;
   count--;
   cnotify (notfull);  }

{  nextin = 0; nextout = 0; count = 0;  }
```

Ved anvendelse av MESA-monitor i stedet for vanlig monitor må en endre både signaliseringsform (da MESA-metoden er en annen enn for vanlige monitorer) og testmekanisme (da MESA-signalisering ikke direkte gjenstarter en tidligere stoppet prosess).

e) Hvilke utfordringer brukes vranglåsmekanismer (Deadlock Mechanisms) til å håndtere?

SVAR:

Vranglåsmekanismer trengs for å håndtere situasjoner der synkronisering fører til at to / flere prosesser / tråder venter på hverandre i ring.

f) Når bør et system bruke henholdsvis umuliggjøring (Prevention), unngåelse (Avoidance) og oppdaging (Detection) av vranglåser?

SVAR:

Ulike situasjoner vil føre til at ulike valg blir naturlige:

- Umuliggjøring / unngåelse bør brukes når vranglåser ofte kan/vil oppstå –
  da gjenoppretting etter oppdaging av vranglåser er ressurskrevende i etterkant,
  mens oppdaging bør brukes når vranglåser sjelden kan/vil oppstå –
  da sikring av umuliggjøring / unngåelse av vranglåser er ressurskrevende i forkant
- Umuligjøring vil brukes hvis en kan leve med en liten resulterende parallellitet –
  da umuliggjøring er ganske begrensende slik sett,
  mens unngåelse vil brukes hvis en ikke kan leve med en liten resulterende parallellitet –
  da unngåelse er mindre begrensende slik sett

Oppgave 4: Håndtering av lager (Memory Management)

a) Hvilke forhold brukes virtuelt lager (Virtual Memory) til å utnytte?

SVAR:

Virtuelt lager utnytter lokalitetsprinsippet som tilsier at lagerreferanser klumper seg, ved at en gitt lagerreferanse gjerne etterfølges av nye lagerreferanser i nærheten av den gitte – ved aksess av både kode og data.

b) Når bør et system bruke henholdsvis segmentering (Segmentation), sidedeling (Paging) eller en kombinasjon av dem?

SVAR:

Ulike situasjoner vil føre til at ulike valg blir naturlige:

- Segmentering – med programbiter/databiter av ulik størrelse som søkes plassert i hull av forskjellig størrelse, er et konsept som er tilpasset et programsynspunkt,
  krever søking for optimal plassering av biter i hull, og gir ekstern fragmentering. Dette kan f.eks. velges hvis søking etter hull til bitene ikke blir for ressurskrevende enkeltvis og samlet
- Sidedeling – med programbiter/databiter av lik størrelse som ønskes plassert i rammer av tilsvarende størrelse, er et konsept som er tilpasset et maskinsynspunkt, unngår
  søking ved plassering av biter i rammer, men gir intern fragmentering. Dette kan f.eks. velges hvis den forhåndsdefinerte rammestørrelsen ikke gir for mye intern fragmentering enkeltvis og samlet
- Kombinasjon av segmentering og søking gir fordelene og ulempene fra begge typer, og det gir større overhead mht plassbehov for tabeller og oppslagsbehov i tabeller. Dette kan f.eks. velges hvis en har råd til tabelloverheaden som oppstår

c) Hvilke utfordringer brukes sideutbyttingsalgoritmer (Page Replacement Policies) til å håndtere?

SVAR:

Sideutbyttingsalgoritmer trengs for å fram til en best mulig ramme å overskrive innholdet av når en ny side må hentes fra sekundærlageret og det ikke er noen ledig ramme i primærlageret – hvor «best mulig» vil tilsi at en må prediktere framtidige hendelser ut fra fortidig oppførsel.

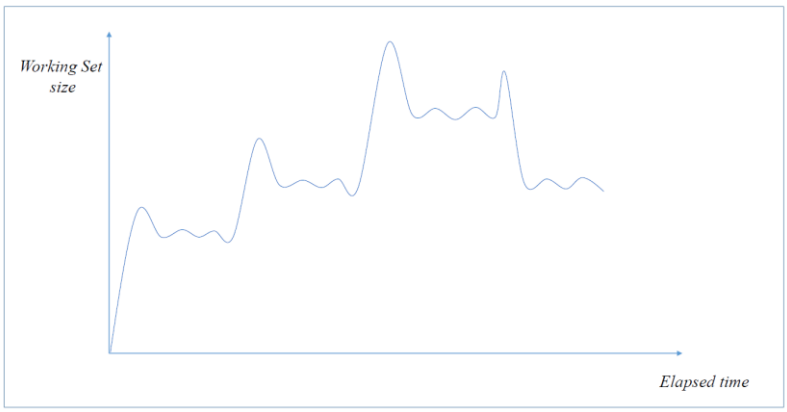
d) Når bør et system bruke henholdsvis LRU (Least Recently Used), LFU (Least Frequently Used), U-CLOCK (2-chance Clock) og UM-CLOCK (4-chance Clock) som algoritme?

SVAR:

Ulike situasjoner vil føre til at ulike valg blir naturlige:

- LRU & LFU er i utgangspunktet for ressurskrevende til å brukes for slik implisitt I/O (men kan brukes – gjerne i kombinasjon, for håndtering av eksplisitt I/O)
- U-CLOCK & UM-CLOCK er mindre ressurskrevende og passer godt for slik implisitt I/O
- UM-CLOCK er litt mer plasskrevende og tidskrevende enn U-CLOCK, men gir samtidig bedre resultater – og er således å foretrekke hvis en har råd til denne ekstra overheaden

e) I både læreboken og forelesningene diskuteres følgende sammenheng mellom arbeidssettstørrelsen (Working Set size) og forløpt tid (Elapsed time):

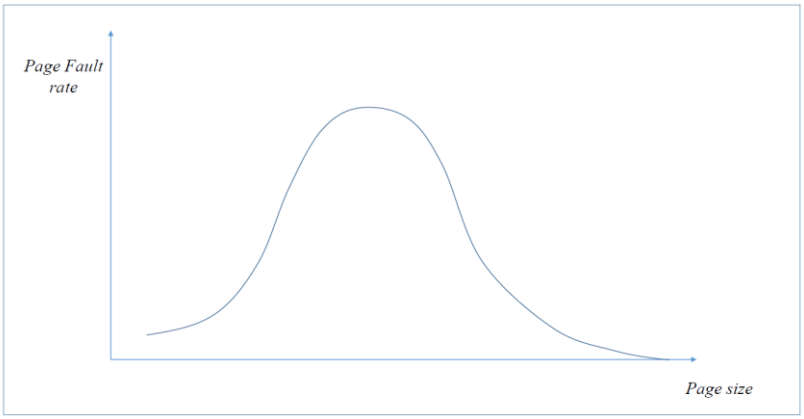


Hvorfor varierer arbeidssettstørrelsen med forløpt tid som angitt i figuren?

SVAR:

Arbeidssettstørrelsen speiler omfanget av det settet med sider som til enhver tid aksesseres. Lokalitetsprinsippet tilsier at arbeidssettet og dermed også arbeidssettstørrelsen holder seg stabile over visse tidsrom. Med jevne mellomrom endrer dog arbeidssettet seg, og arbeidssettstørrelsen vil svinge tilsvarende opp og ned. I slike overgangssituasjoner vil sidene som aksesseres i praksis tilhøre «to arbeidssett», og den tilhørende «arbeidssettstørrelsen» vil da være større enn både før og etter.

f) I både læreboken og forelesningene diskuteres følgende sammenheng mellom sidefeilsraten (Page Fault rate) og sidestørrelse (Page size):



Hvorfor varierer sidefeilsraten med sidestørrelse som angitt i figuren?

SVAR:

Figuren reflekterer to ulike forhold:

- Hvis sidestørrelsen blir stor nok, rommes en hel prosess/tråd innen en enkelt side – og en får aldri sidefeil, men en har heller ikke plass til så mange prosesser/tråder inne i primærlageret

- Mens hvis sidestørrelsen blir liten nok, rommes mange nok deler av angjeldende prosess/tråd i tildelt primærlagerområde – og en får sjelden sidefeil, men en har igjen ikke plass til så mange prosesser/tråder inne i primærlageret grunnet enorm overhead til sidetabeller etc

Målet blir å finne en sidestørrelse mellom de to ytterpunktene i figuren slik at resulterende sidefeilsfrekvens dog holdes under en angitt grense.

Oppgave 5: Tidsstyring av prosesser (Process Scheduling)

a) Hvilke spørsmål brukes tidsstyringsalgoritmer til å svare på?

SVAR:

Tidsstyringsalgoritmer må avgjøre i hvilken rekkefølge ulike prosesser / tråder som ønsker adgang til en felles ressurs, skal få adgang til denne ressursen – i dette spesifikke tilfelle en / flere CPU(er)

b) Når bør et system bruke henholdsvis FCFS (First Come First Served), SPN (Shortest Process Next), RR (Round Robin) og SRT (Shortest Remaining Time) som algoritme?

SVAR:

Ulike situasjoner vil føre til at ulike valg blir naturlige:

- FCFS: Kan brukes i satsvise systemer - altså systemer uten interaktiv tilgang for brukere til programmene / applikasjonene som kjøres, hvor høy gjennomstrømning ikke oppfattes som viktig
- SPN: Kan brukes i satsvise systemer - altså systemer uten interaktiv tilgang for brukere til programmene / applikasjonene som kjøres, hvor en viss gjennomstrømning ønskes oppnådd gjennom prioritering av små prosesser
- RR: Bør brukes i systemer med interaktiv tilgang for brukere til programmene / applikasjonene som kjøres
- SRT: Kan brukes i satsvise systemer - altså systemer uten interaktiv tilgang for brukere til programmene / applikasjonene som kjøres, hvor en viss gjennomstrømning ønskes oppnådd gjennom prioritering av både små og store prosesser

c) Hvilke ekstra tidsstyringsutfordringer introduseres med henholdsvis multiprosessorer (Multiprocessors) og multikjerner (Multicores)?

SVAR:

Med multiprosessorer og / eller multikjerner tilgjengelig skifter fokuset ofte fra å holde hver prosessor / hver kjerne maks utnyttet til å få til effektivt samarbeide mellom delprogrammer/ delapplikasjoner. Dette fordrer igjen at tråder tilhørende en gitt prosess gjerne ønskes kjørt samtidig. Og hvis tråder fortsatt kan bli avbrutt, er det videre ønskelig at en tråd kan gjenstartes på en prosessor/kjerne med rask tilgang til samme cache som sist. For multikjerner

med felles cache innen en felles brikke blir dette spesielt viktig, mens for multiprosessorer uten felles cache innen en felles brikke blir dette mindre viktig.

d) Når bør et system bruke henholdsvis HPF (Highest Priority First), EDF (Earliest Deadline First) og RMS (Rate Monotonic Scheduling) for tidsstyring i sanntidssystemer?

SVAR:

Ulike situasjoner vil føre til at ulike valg blir naturlige:

- HPF: Vil brukes for en gruppe prosesser / tråder med klar prioritering dem imellom, i betydning fra harde til bløte prioriteter
- EDF: Kan brukes når snittgarantier holder, altså når absolutte garantier for hver enkelt prosess/tråd i et sett med ellers likeverdige prosesser/tråder ikke er nødvendig
- RMS: Må brukes når variansgarantier trengs, altså når absolutte garantier for hver enkelt prosess/tråd i et sett med ellers likeverdige prosesser/tråder er nødvendig

Oppgave 6: Håndtering av I/O (I/O Management)

a) Når bør en bruke henholdsvis caching (Caching), bufring (Buffering) eller ingen av delene for dataoverføring mellom eksternlager og internlager?

SVAR:

Ingen av delene bør brukes hvis en fokuserer på hastigheten til en isolert dataoverføring for et enkelt dataelement.

Mens bufring bør brukes når en fokuserer på effektiv dataoverføring for mange ulike dataelementer samlet sett.

Og caching bør brukes når en fokuserer på effektiv dataoverføring for mange ulike dataelementer samlet sett – og dataelementene kan bli gjenbrukt flere ganger av samme eller ulike program(er) /applikasjon(er).

b) Når bør et system bruke henholdsvis sammenhengende (Contiguous), kjedet (Chained) og indeksert (Indexed) plassallokering i filsystemer?

SVAR:

Sammenhengende plassallokering kan brukes i statiske situasjoner med små endringer av filinnhold underveis.

Kjedet plassallokering bør brukes i dynamiske situasjoner med store endringer av filinnhold underveis - når en har råd til stor overhead mht tidsforbruk ved gjennomsøking av ofte lange lenker.

Indeksert plassallokering bør brukes i dynamiske situasjoner med store endringer av filinnhold underveis - når en har råd til stor overhead mht plassbehov ved oppretting av ofte store tabeller.

Oppgave 7: Operativsystemsikkerhet (Operating System Security)

a) Hvilke sikkerhetsutfordringer må moderne operativsystemer løse?

SVAR:

Hovedutfordringene sikkerhetsmessig er:

- Inntrengende aktør – a) en uautorisert bruker som utgir seg for å være en autorisert bruker, b) en autorisert bruker som går ut over sine tilhørende rettigheter, c) en bruker som tar over systemet som superbruker og derigjennom unngår / omgår kontroll av rettigheter etc.
- Ondsinnet programvare – a) et parasittprogram av virustype eller bakdørstype – som er avhengig av et annet program å kople seg på, b) et selvstendig program av ormtype eller robottype – som er uavhengig av et annet program å kople seg på.

b) Hvordan løser moderne operativsystemer sikkerhetsutfordringene knyttet til aksess av filer (File System access)?

SVAR:

Hovedmekanismene filaksessmessig er:

- Brukere vil tildeles hver sin identitet
- Brukere kan tilordnes flere ulike roller
- Identiteter / roller vil tilkjennes rettigheter til objekter
- Identiteter / roller kan tilordnes nivåer av rettigheter
- Systemet holder oppsyn med rettigheter og nivåer
- Systemet sjekker objektrettigheter / rettighetsnivåer ved aksesser