NTNU
Kunnskap for en bedre verden

Institutt for Datateknikk of Informasjonsvitenskap

Løsningsforslag for TDT4186 Operativsystemer

Eksamensdato: 13. august 2015

Eksamenstid (fra-til): 15:00-19:00

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.
Dersom du mener at opplysninger mangler i oppgaveformuleringene, beskriv de antagelsene du gjør.
Hver av de seks oppgavene teller like mye, og hver av de tre deloppgavene teller like mye.

Oppgave 1: Operativsystemer (Operating Systems)

a) Angi kort hvilke mål en har ved utvikling av operativsystemer

SVAR:

- Å kunne tilby tjenester så enkelt som mulig for brukere / program
- Å kunne forvalte ressurser så effektivt som mulig for systemet
- Å kunne fortsette utvikling over tid så fleksibelt / billig som mulig for brukere / programmer & systemet

b) Sammenlign singelprosessor (singleprocessor) systemer og multiprosessor (multiprocessor) systemer mht funksjonalitet og implementering

SVAR:

- Funksjonalitet: Multiprosessor systemer tilbyr muligheter for større pålitelighet, større ytelse og mer réell parallellitet enn singelprosessor systemer
- Implementering: Multiprosessor systemer tilsier større utfordringer mht forhold som prosess-synkronisering, lagerhåndtering & tidsstyring enn singelprosessor systemer

c) Diskuter fordeler og ulemper med multikjerne (multicore) arkitekturer og multiprosessor (multiprocessor) arkitekturer

SVAR:

- Multiprosessor arkitekturer: Tilbyr god funksjonalitet på flere forhold, men har utfordringer mht god implementering - korrekthet & skalerbarhet
- Multikjerne arkitekturer: Tilbyr nyttig caching på flere nivåer, med har utfordringer mht effektiv utnyttelse - virtuell maskin vs. sentral støtte

Oppgave 2: Prosesser og tråder (Processes and Threads)

a) Angi kort forskjellene mellom prosesser og tråder i operativsystemsammenheng

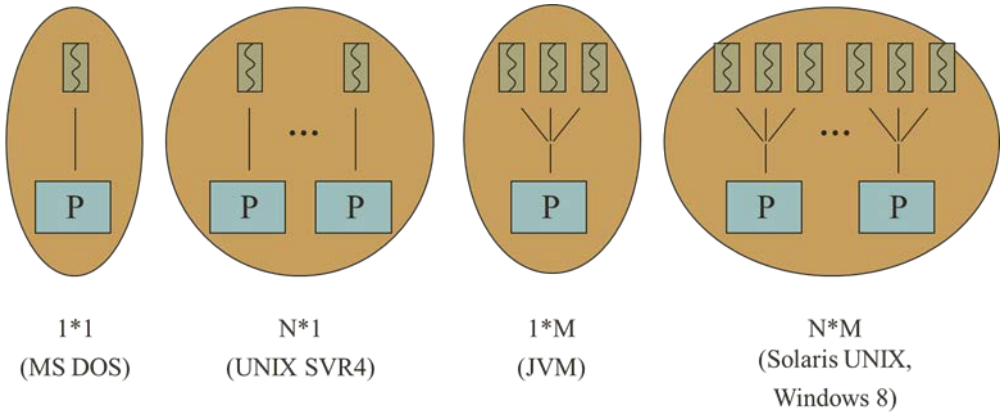
SVAR:

- Prosesser tilsvarer mer isolerte aktiviteter, har mer kontekst knyttet til seg og kan deles i tråder
- Tråder tilsvarer mer samhørende aktiviteter, har mindre kontekst knyttet til seg og kan samles i prosesser

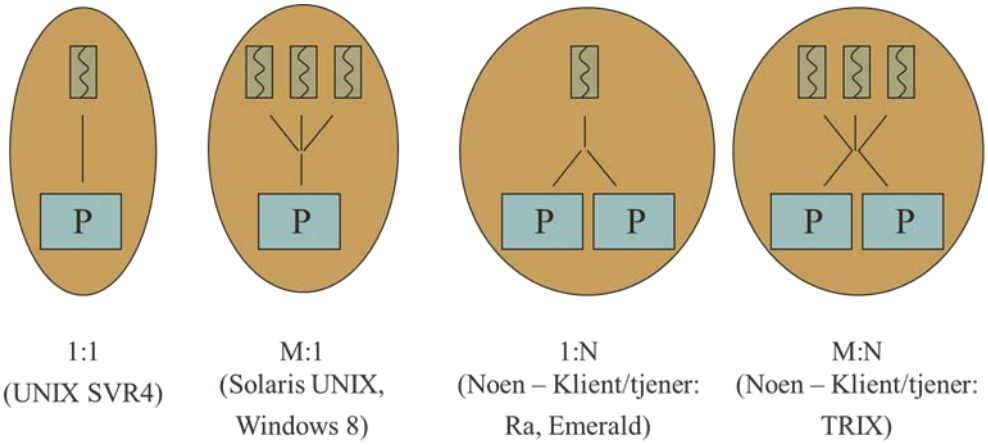
b) Illustrer med figur og tekst ulike måter å kombinere prosesser og tråder i eksisterende operativsystemer

SVAR:

- Enkle kombinasjoner



- Avanserte kombinasjoner



c) Sammenlign bruk av brukertråder (user-level threads) med bruk av kjernetråder (kernel-level threads) i programvare

SVAR:

- Ytelse: Brukertråder gir billige, men også blokkerende systemkall – mens kjernetråder gir dyre, men også ikke-blokkerende systemkall
- Anvendelse: Brukertråder vil ikke kunne utnytte multiprosessorer, mens kjernetråder vil kunne utnytte multiprosessorer

Oppgave 3: Synkronisering av prosesser (Process Synchronization)

a) Forklar kort hva semaforer (semaphores) er i operativsystemsammenheng

SVAR:

- Semaforer er et programmeringsverktøy for å håndtere uønsket parallellitet mellom ulike prosesser/tråder i programmer – og består av en teller og en ventekø samt to tilhørende operasjoner

b) Angi hvordan semaforer formelt defineres for prosessynkroniseringsformål

SVAR:

```
struct semaphore {
      int count;
      queueType queue;
};
void semWait(semaphore s)
{
      s.count--;
      if (s.count < 0) {
            /* place this process in s.queue */;
            /* block this process */;
      }
}
void semSignal(semaphore s)
{
      s.count++;
      if (s.count <= 0) {
            /* remove a process P from s.queue */;
            /* place process P on ready list */;
      }
}
```

c) Beskriv med kode og kommentarer minst to konkrete eksempler på hvordan semaforer brukes til å løse aktuelle utfordringer i programvare

SVAR:

- Gjensidig utelukkelse

Var R.S: Semaphore := 1

EnterCritical (R):
Wait (R.S)

ExitCritical (R):
Signal (R.S)

- Generell tidsordning

```
Var B: Semaphore := N / 1;
    Q, A: Semaphore := 0;

KlientProsess:
  Repeat
    <Generate Question>; Wait (B);
    <Write Message>; Signal (Q);
    Wait (A); <Read Message>;
    Signal (B); <Apply Answer>
  Forever

TjenerProsess:
  Repeat
    Wait (Q); <Read Message>;
    Signal (B); <Apply Question>;
    <Generate Answer>; Wait (B);
    <Write Message>; Signal (A)
  Forever
```

Oppgave 4: Håndtering av lager (Memory Management)

a) Forklar kort hva arbeidssett (working sets) er i operativsystemsammenheng

SVAR:

- Arbeidssett er et modelleringsverktøy som har som mål å fastlegge de og bare de sidene som til enhver tid trengs lett tilgjengelig for ulike prosesser / tråder. En betrakter da gjerne de sider som har vært brukt av prosessen / tråden over en viss tidsperiode. Omfanget av slike sider avhenger av valgt tidsperiode og vil svinge over tid men med lange stabile perioder innimellom.

b) Angi hvordan arbeidssett teoretisk kan benyttes for lagerhåndteringsformål – og diskuter om arbeidssett praktisk kan benyttes for lagerhåndteringsformål

SVAR:

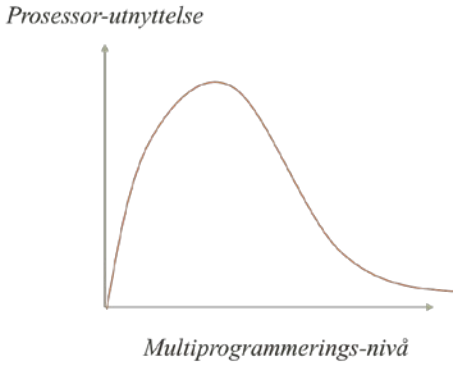
- Teoretisk benyttelse tilsier å tilpasse sidesett (RS) i primærlageret som følger:
  - Monitorer alle prosessers WS for gitt Δ (Optimal Δ-verdi?)
  - Inkluder kontinuerlig de S innenfor WS i RS (Virtuelt Lager-innretning!)
  - Ekskluder periodisk de S utenfor WS fra RS (∃ fortid! => ∃ fremtid?)
  - Kjør kun prosesser der RS omfatter WS (Swappende Lager-innretning!)

- Praktisk utnyttelse blir problematisk av følgende grunner:
  - Fortid angir ikke alltid framtid
  - Kontinuerlig måling av WS er vanskelig
  - Fastlegging av optimal Δ er vanskelig

c) Illustrer med figur og tekst avveiinger som må foretas for å fastlegge hvor mange prosesser/ tråder som kan holdes i primærlageret og hvor mye kode & data hver av disse prosessene/ trådene kan ha inne i primærlageret til enhver tid

SVAR:

- Sammenheng mellom prosessor-utnyttelse og multiprogrammerings-nivå



  - For få prosesser / tråder inni i primærlageret gir dårlig CPU-utnyttelse da det således ofte ikke er noen prosesser / tråder tilgjengelig for bruk av CPU-kraft
  - For mange prosesser / tråder inni i primærlageret gir dårlig CPU-utnyttelse da det dermed brukes mye CPU-kraft på ut- & innswapping av hele prosesser / tråder
  - Et brukbart balansepunkt mellom disse to ytterligheter etterstrebes ved å overvåke sidefeilfrekvensen til hver enkelt prosess / tråd
  - Hver enkelt prosess / tråd må ha så mye kode & data inni i primærlageret at dens sidefeilfrekvens holder seg under et visst tak
  - Antall aktive prosesser / tråder justeres da til enhver tid ut fra svingningene i hver enkelt prosess / tråd sin tilhørende sidefeilfrekvens

Oppgave 5: Tidsstyring av prosesser (Process Scheduling)

a) Forklar kort betydningen av tidsfrister (deadlines) i operativsystemsammenheng

SVAR:

- Tidsfrister betyr at fokus ikke bare er på at et korrekt resultat oppnås i beregninger, men også på når resultater av beregninger oppnås. Dette er ofte tilfelle når beregninger gjøres mer på vegne av maskiner enn på vegne av mennesker.

b) Illustrer med figur og tekst hvordan tidligste-tidsfrist-først (earliest deadline first) algoritmen kan virke godt for noen prosesstidsstyringstilfeller – og ikke like godt for andre prosesstids-styringstilfeller

SVAR:

- Eksempel hvor algoritmen kan virke godt

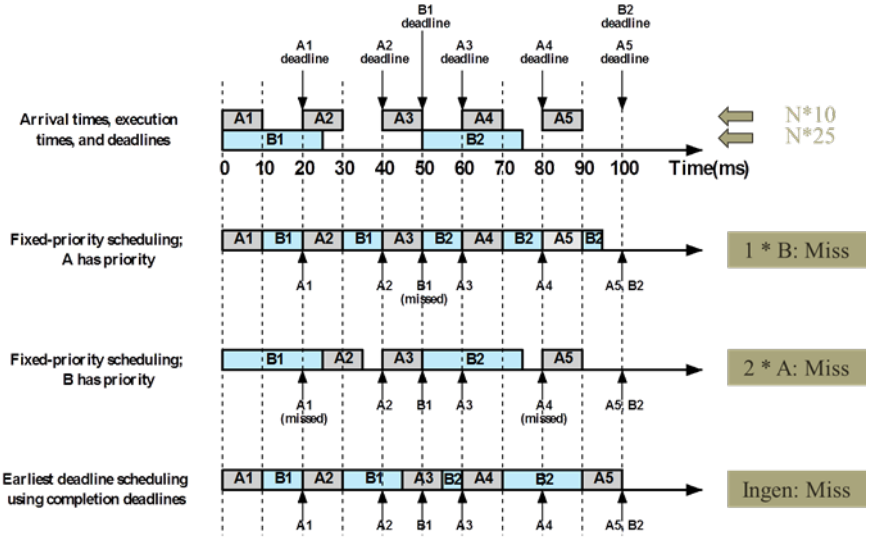


Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines (based on Table 10.2)

- Eksempel hvor algoritmen kan kreve tilpasning

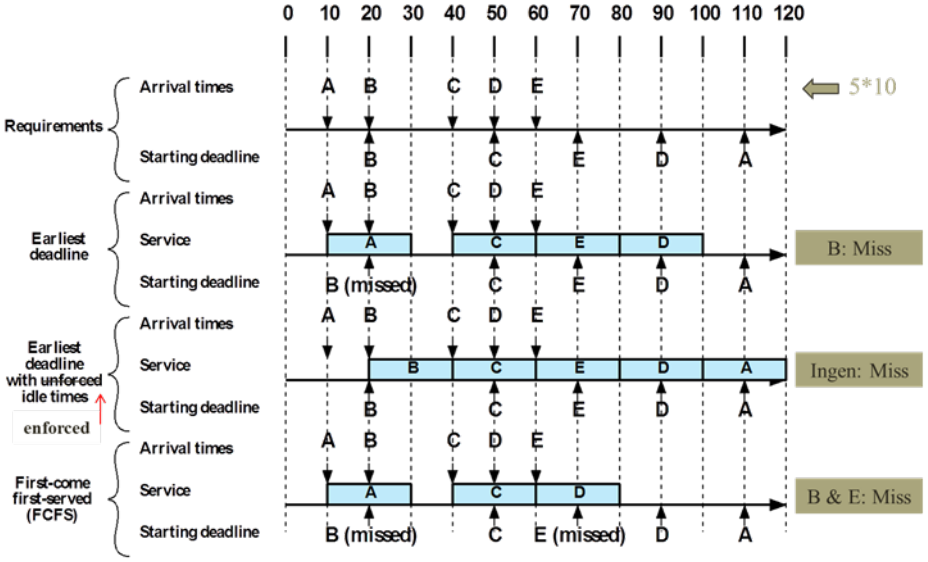


Figure 10.6 Scheduling of Aperiodic Real-time Tasks with Starting Deadlines

c) Sammenlign effekt og ytelse av tidligste-tidsfrist-først algoritmen med noen andre relevante algoritmer mht ulike former for prosesstidsstyring med tidsfrister

SVAR:

- Earliest Deadline First-algoritmen gir garanti for snitt, men gir ikke garanti for varians, og gir ikke garanti for hver enkelt prosess / tråd for perioderangering / slakkrangering
- Rate Monotonic Scheduling-algoritmen gir garantier for hver enkelt prosess / tråd gitt at et ganske tøft utnyttelseskrav holder for perioderangering
- Least Laxity Scheduling-algoritmen gir garantier for hver enkelt prosess / tråd gitt at et mindre tøft utnyttelseskrav holder for slakkrangering

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

a) Angi kort forskjellene mellom bufring (buffering) og caching (caching) i operativsystem-sammenheng

SVAR:

- Bufring er mellomlagring med fokus på en enkelt aksess -
  mellom CPU og I/O-enheter, både inn og ut via OS-område

- Caching er mellomlagring med fokus på flere aksesser -
  utnytter lokalitetsprinsippet for fil-data

b) Sammenlign effekt og ytelse av ulike varianter av bufring ifm I/O-håndtering – og sammenlign effekt og ytelse av ulike varianter av caching ifm I/O-håndtering

SVAR:

- Varianter av bufring
  - Uten bufring: Gir kjøre-blokkering og swapp-blokkering
  - Single bufring: Unngår kjøre-blokkering og swapp-blokkering
  - Dobbel bufring: Utjevner mindre hastighetsvariasjoner
  - Sirkulær bufring: Utjevner større hastighetsvariasjoner

- Varianter av caching
  - Minst nylig referert (LRU):
    Krever tidsmerke for hver side til en prosess / tråd
    Kan i praksis brukes i motsetning til for program-data
  - Minst ofte referert (LFU):
    Krever referanseteller for hver side til en prosess / tråd
    Kan i praksis brukes i motsetning til for program-data

- Frekvensbasert stakk (FBS):
  Krever seksjonert stakk for hver prosess / tråd
  Fungerer bedre enn både LRU & LFU ved å kombinere dem

c) Diskuter bruk av ulike former for bufring og / eller caching mht ulike typer I/O-enheter

SVAR:

- Uten bufring / caching: 1*DMA
  - Raskest (D->L / L->D)
  - Anvendelig for f.eks. disk / tape / skriver
- Med bufring: 2*DMA
  - Uten blokkering (“D”->L, L->L / L->L, L->“D”)
  - Anvendelig for f.eks. terminal / kommunikasjonslinje / skriver
- Med caching: 2*DMA
  - Lokalitetsutnyttelse (D->L, L->L / L->L, L->D)
  - Anvendelig for f.eks. disk / tape