# Vranglås

1. i.

**-Gjensidig utelukking**

En ressurs er enten ledig eller allokert til en prosess. Prosessen som har ressursen har eksklusiv aksess til ressursen. Alle andre prosesser som prøver å få tak i ressursen blir utelukket.

**-No preemption**

Ressursen kan bare frigis av prosessen selv. Ingen andre kan kreve ressursene som en prosess har tak i. Den eneste måten å frigi en ressurs er at prosessen selv frigjør den.

**-Hold på ressursen og vent**

En prosess kan holde på en ressurs mens den spør etter nye ressurser.

**-Sirkulær venting**

Dette er en sirkulær kjede der hver prosess holder en ressurs som er ønsket av en annen.

Det trenger ikke alltid å oppstå en deadlock når alle disse er tilstede, men deadlock er umulig dersom en av dem ikke er tilstede.

1. Forklar hvordan denne følgende situasjonen fører til vranglås, og hvordan CPU scheduleren kunne oppdage og unngå det.

Ressurser: R1, R2[2] (R2 består av to kopier)

Prosesser: P1 (høy prioritet), P2 (høy prioritet), P3 (lav prioritet)

P1 eier R1, og ber om én av R2

P2 eier R2[0], og ber om R1

P3 eier R2[1]

-Her ser vi at prosesser holder på ressurser mens de spør om nye. Vi får en sirkulær venting siden hver ressurs holder en ressurs som er ønsket av en annen. P3 blir stående siden de høyt prioriterte prosessene ikke får kjørt. For å løse denne vranglåsen kan CPU-en endre prioriteringen til P3 slik at den kan kjøres først og frigi ressursen R2 som P1 trenger. Dette gjør at P1 får kjørt seg ferdig og kan frigi ressurser til P2.

**Maks poeng: 12**

# Round-robin

Forklar hvordan round-robin scheduling foregår. Hvilken betingelse er nødvendig for denne algoritmen?

Round-robin scheduling er den vanligste metoden for å tildele CPU-tid til prosesser. Her fordeles CPU-tiden likt for alle prosessene som befinner seg i systemet. Hver prosess får tildelt et tidskvant som den har lov til å kjøre på CPU-en. Når denne utløper vil neste prosess i CPU-køen kjøres og prosessen som kjørte blir lagt bakerst i CPU-køen. Dette forutsetter pre emtive CPU scheduling, dvs at en kjørende prosess kan bli avbrutt og en ny kan allokeres til CPU-en.

**Maks poeng: 5**

# Sikkerhet

Angi de tre OS sikkerhetsmålene og beskriv kort hensikten med hver. Hvilke trusler prøver hver enkelt å unngå?

**Konfidensialitet:**

Brukeren av OS vil opprettholde konfidensialitet av infoen som er lagret i datasystemet. Det vil si at informasjon ikke bør være tilgjengelig til noen som ikke er autorisert for å mota det.

**Integritet:**

Brukeren vil opprettholde integriteten av informasjonen lagret på datasystemet. Det betyr at informasjonen ikke bør endres eller slettes av personer som ikke er autorisert til å gjøre slike endringer.

**Tilgjengelighet:**

Brukeren vil ivareta tilgjengeligheten av tjenesten som er tilbydd av datasystemet. Det betyr at personene som er autorisert til å bruke systemet burde få gjøre det uten nekting av tjeneste.

**Ansvarlighet:**

Brukeren ønsker å sikre ansvarlighet. Det betyr at det burde være mulig å bestemme hvordan brukere har valgt å bruke deres autoritet, så de kan holdes ansvarlig for valg de har tatt innenfor grensene som er satt av sikkerhet kontroller.

**Maks poeng: 5**

# Sidedelt minne

Anta at sidestørrelsen er på 100 og minnet har 200 adresser tilgjengelig. Gitt den følgende

minnereferansesekvensen av et program som består av 400 adresser:

33, 34, 35, 36, 100, 101, 102, 37, 38, 388, 389, 390, 388, 280, 106

Hvor mange sidefeil blir det for hver av de to sideutskiftningsalgoritmene:

Siden en side har en sidestørrelse på 100 vil vi ha 4 sider (et program består av 200 adresser). Minnet har bare plass til 2 sider på samme tid.

Side 0: 0-99

Side 1: 100-199

Side 2: 200-299

Side 3: 300-399

Siden vi begynner med ingen sider i minnet må vi hente inn dette, noe som vil føre til en sidefeil. Ser at første side som må hentes inn er 0.

|  |
| --- |
| 0 |
|  |

Sidefeil: 1

Når vi vil ha tak adresse 100 må vi hente inn en ny side med adressen som vil være side 1. Dette fører til en til sidefeil.

|  |  |
| --- | --- |
| 0 | 0 |
|  | 1 |

Sidefeil: 2

Når vi når adresse 388 må vi ta inn side 3, men minnet rommer bare 200 adresser så nå må vi droppe en av sidene. Om vi hadde brukt optimal ville vi forutsett hvilke sider som ikke hadde blitt brukt på nytt, så da ville vi ha erstattet 0, med LRU (least recently used) ville vi ha byttet ut 1. Da får vi en sidefeil til.

|  |  |  |  |
| --- | --- | --- | --- |
| LRU | | | |
| 0 | 0 | 0 | 0 |
|  | 1 | 1 | 3 |
| Optimal | | | |
| 0 | 0 | 0 | 3 |
|  | 1 | 1 | 1 |

Sidefeil: 3

Når vi når adresse 280 må vi erstatte på nytt. Med optimal ville vi ha erstattet 3 men og med LRU ville vi ha byttet ut 0. Da får vi enda en sidefeil.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| LRU | | | | | | | |
| 0 | 0 | | 0 | | 0 | | 2 |
|  | 1 | | 1 | | 3 | | 3 |
| Optimal | | | | | | | |
| 0 | | 0 | | 0 | | 3 | 0 |
|  | | 1 | | 1 | | 1 | 2 |

Sidefeil: 4

Når vi skal ha adressen 106 ligger ikke den lenger i minnet og vi må hente den ut på nytt, med LRU må vi hente inn 2 igjen mens med optimal vil 2 ligge i minnet så vi vil få en mindre sidefeil.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LRU | | | | | |
| 0 | 0 | 0 | 0 | 2 | 2 |
|  | 1 | 1 | 3 | 3 | 1 |
| Optimal | | | | | |
| 0 | 0 | 0 | 3 | 0 | 0 |
|  | 1 | 1 | 1 | 2 | 2 |

Sidefeil: 5 (optimal 4.)

1. Optimal

4 sidefeil

2. LRU

5 sidefeil.

**Maks poeng: 13**

# Sidetabell

Et innslag i sidetabellen har flere felter, blant annet referanse-bit, dirty-bit, og tilstede bit. Forklar hensikten med hver av disse bitene.

- Referanse bit - used for page utskift algorithms like LRU

- Dirty bit - data in page has not yet been written back to disk or memory

- tilstede bit - data is currently valid in the page

**Maks poeng: 5**

*Innledning*

På en disk lagres alltid informasjon i form av bitmønstre. Disker er delt opp i to deler: et stort dataområde og et ganske lite systemområde. Dataområdet inneholder selve

dataene (bit-mønstre organisert som filer). Systemområdet inneholder informasjon om hva som finnes på dataområdet (filnavn, hvor begynner filene, hvor lange er de osv.)

NB: Du trenger ikke vite noe om operativsystemer og filsystemer for å svare på denne oppgaven. Du skal kun forholde deg til bitmønstre, tallsystemer og datarepresentasjon!

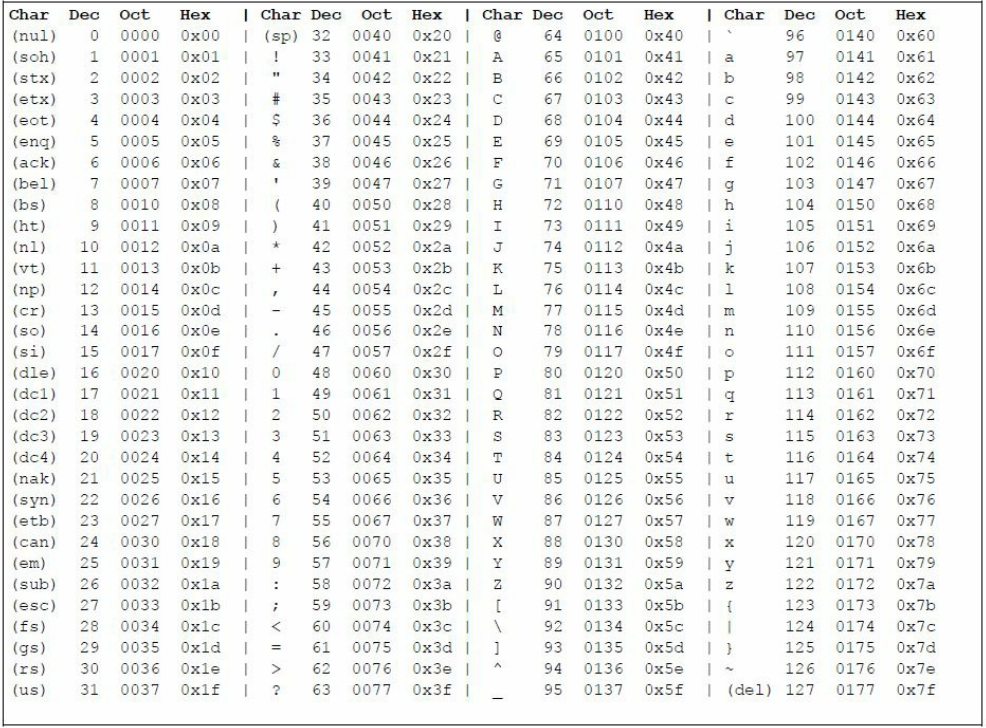
Hvis en diskfeil ødelegger systemområdet vil dataområdet likevel være uskadd. Problemet er at informasjonen på dataområdet utelukkende er bitmønstre, og nå vet du ingen ting om hva disse bitmønstrene representerer, hvilken fil de tilhører, hva slags fil det er, eller noen annet. Du har altså bare en lang rekke med bits liggende på hver sektor. Anta at du har en disk med ødelagt systemområde. Fra dataområdet på disken har du hentet ut bitmønstre fra starten av to ulike sektorer.

Disse bitmønstrene er gitt nedenfor på heksadesimal form:

Sektor A: 546F2062 65206F72

Sektor B: 69662861 3E313029

Spørsmålene i oppgaven tar utgangspunkt i disse to bitmønstrene.



# Binær

Skriv bitmønsteret fra sektor A på binær form. Forklar fremgangsmåten tydelig.

|  |  |
| --- | --- |
| Hex | Binær |
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |
| A | 1010 |
| B | 1011 |
| C | 1100 |
| D | 1101 |
| E | 1110 |
| F | 1111 |

Bruker tabellen ovenfor for å omskrive tallet.

5 4 6 F 2 0 6 2 6 5 2 0 6

(0101) (0100) (0110) (1111) (0010) (0000) (0110) (0010) (0110) (0101) (0010) (0000) (0110)

F 7 2

(1111) (0111) (0010)

**Maks poeng: 5**

# Ascii

Du husker at du har to tekstfiler på disken. En fil med java kildekode, og en fil med et dikt av W. Shakespeare.

1. Vis hvordan du går frem for å sjekke om bitmønstrene fra sektor A og sektor B kan være deler av tekstfiler på 8-bits ascii-format? (Du ser Ascii-tabellen til venstre for denne teksten.)

Vi vet at sektor A og B begge består av 16 hexadesimaler og siden deler av tekstfiler kan være på 8-bits vet vi at to heksadesimal tall tilsvarer 8 bits. Deretter bruker vi tabellen vi får oppgitt for å sjekke hva tallene tilsvarer.

2. Virker det sannsynlig at dette kan være ascii-tekst? Begrunn svaret nøye.

Det virker meget sannsynlig at det kan være ascii-tekst siden oversettingen vi får ved hjelp av tabellen gir mening, men vi kan aldri si det med 100% sikkehet.

3. Hvilken sektor ser i så fall ut til å inneholde java kildekode, og hvilken inneholder et dikt?

Husk å begrunne alle svar nøye.

Begynner med å oversette ved hjelp av tabellen:

Sektor A: 54 6F 20 62 65 20 6F 72

Sektor B: 69 66 28 61 3E 31 30 29

Får ut disse oversettingene:

Sektor A: To\_be\_or

Sektor B: If(a>10)

Ser at sektor A ser ut til å inneholde diktet mens sektor B ser ut til å inneholde java kildekode.

**Maks poeng: 6**

# Toer-komplement heltall

Anta at bitmønstrene på Sektor A representerer 32-bits heltall med fortegn på toer-komplement form.

Sektor A: 546F2062 65206F72

Sektor B: 69662861 3E313029

1. Hvor mange slike heltall består bitmønstret fra sektor A av? Begrunn svaret.

Sektor A består av 2 heltall, dette vet vi siden et hexadesimal tall tilsvarer 4-bits, og sektor A består av 16 hexadesimal altså 16\*4= 64 bits. Vi vet at 32 bits var et heltall derfor vet vi at sektor A må ha to siden 64/2=32.

2. Er noen av heltallene negative? Begrunn svaret nøy

Siden vi vet at toer-komplementet for negative tall på hexadesimal vil være 7-F og ingen av hexadesimalene begynner med noe fra 7-f vet vi at alle heltallene vil være positiv.

**Maks poeng: 6**

# Parallellitet på instruksjonsnivå

1. Hva mener vi med parallellitet på instruksjonsnivå (engelsk: Instruction Level Parallelism)?

Med dette menes det at en og samme prosessor kan utføre flere instruksjoner parallelt.

2. Nevn eksempler på mekanismer i moderne prosessorarkitektur som implementer denne type parallellitet. Skisser hvordan parallellitet oppnås med hver av disse mekanismene.

Vi har lært om pipelines, superskalare prosessorer og dynamisk utførelse for å innføre denne typen parallellitet.

**Pipelines:**

Med pipelines får vi delt opp utføringssyklusens i flere trinn slik at flere instruksjoner kan utføres parallelt. Med dette kan det oppstå situasjoner der instruksjoner som går parallelt trenger samme ressurs, eller er dataavhengige. Da oppstår det vi kaller hasarder. Mer utfyllende om hasarder står i neste oppgave.

**Superskalare prosessorer:**

Ved superskalare prosessorer får vi flere parallelle pipelines. Graden av superskalaritet forteller oss hvor mange parallelle pipelines det er i prosessoren. F.eks. en superskalar prosessor med en superskalaritet av 2. grad har to parallelle pipelines. Kompilatoren til en superskalar prosessor vil sørge for å bytte om på rekkefølgen av instruksjonene slik at man i størst grad unngår at en instruksjon er avhengig av resultater produsert av den foregående.

**Dynamisk utføring:**

Ved hjelp av dynamisk utføring vil prosessoren prøve å gjennomføre hoppene med bedre flyt ved hjelp av gode algoritmer som gjetter hopp. Dynamisk utføring betyr at prosessoren fortsetter med påfølgendeinstruksjoner dersom en instruksjon må vente på primærminnet. Vi skiller mellom statisk forutsigelse ved hopp og dynamisk forutsigelse ved hopp. Ved statisk forutsigelse gjetter prosessoren likt ved hvert hopp og ved dynamisk forutsigelse lager man en automatisk mekanisme som tar vare på hopphistorien til programmet, og på bakgrunn av dette forsøker man å gjette seg frem til utfallet av neste betingede hoppinstruksjon. Om prosessoren gjetter rett vil det være god flyt mens om det gjettes feil vil det virke som om prosessoren stopper opp litt. Nesten som den hikker. Prosessorer med dynamisk utføring vil være veldig komplisert siden man trenger flere mekanismer for å sikre riktig utføring.

3. Hvem er ansvarlig for å sikre at parallellitet på instruksjonsnivå ikke fører til utføringsfeil? Er det programmeren, kompilatoren, operativsystemet, brukeren, prosessorens maskinvare eller en kombinasjon av disse? Begrunn svaret nøye.

Hovedansvarlig for at parallellitet på instruksjonsnivå ikke fører til utføringsfeil er prosessorens maskinvare. Men flere faktorer kan være med å hjelpe som f.eks. kan programmereren kode fornuftig som bidrar til effektivisering i systemet. Kompilatoren kan også være med å effektivisere. F.eks. ved å sørge for å bytte om rekkefølgen slik at ikke dataavhengige prosessorer havner etter hverandre.

**Maks poeng: 16**

# Hasarder

1. I forbindelse med parallellitet på instruksjonsnivå snakker vi om hasard. Hva er hasard?

Når sitasjoner inntreffer der CPU-en må ta spesielle hensyn for å få rett utførelse kalles det for en hasard.

2. Diskuter de tre typene hasard vi har lært om i faget

1. Strukturell hasard

Denne hasarden oppstår når flere instruksjoner får bruk for samme ressurs samtidig. For å sikre korrekt utførelse vil det settes inn et ventetrinn under utføringen av den siste instruksjonen. Denne har ikke noen annen funksjon enn å gi en pause slik at ressurskonflikten blir avklart. Et slikt ventetrinn forplanter seg i de senere instruksjonene og det oppstår en liten forsinkelse, slik at man ikke lenger får ferdigstilt en instruksjon pr klokkeperiode.

2. Datahasard

Om to dataavhengige instruksjoner utføres parallelt oppstår en datahasard. At instruksjoner er dataavhengige betyr at en instruksjon lager et resultat som en annen skal bruke. For å løse en datahasard kan man unngå sitasjonen ved å flytte en uavhengig instruksjon imellom de to dataavhengige instruksjonene. Gode kompilatorer og programmerere vil prøve å unngå kode der datahasarder oppstår.

I en pipeline vil det legges inn et ventetrinn, slik som med strukturelle hasarder, disse vil alltid forplante seg videre å vi får igjen en forsinkelse der vi ikke får utført en instruksjon per klokkepuls.

3. Kontrollhasard

En kontrollhasard er når en hoppinstruksjon gjør at neste instruksjon ikke ligger i neste sekvensielle minnelokasjon. Før prosessoren vet om hoppe skal utføres eller ikke vil pipelinen fylles opp av andre instruksjoner, om det viser seg at det faktisk skal hoppes vil hele pipelinen tømmes (pipe flush) og prosessoren har jobbet forgjeves med instruksjonene som tømmes.

**Maks poeng: 17**