Løsningsforslag Eksamen høst 2019 Operativsystemer

Datamaskinarkitektur

1) En transistor

2) 100 (11 + 01 = 100, 3 + 1 = 4)

3) Heltall blir lagret og behandlet binært i en datamaskin og lagret som nuller og enere. For å legge sammen to store tall trenger man først to registere A og B for å lagre tallene (med 17 bit eller mer for å kunne lagre femsifrede tall). Tallet 0 representeres ved null spenning og tallet en ved en positiv spenning. Man lager så et logisk men svært komplekst nettverk av nanometertykke ledninger der strømmen styres ved hjelp av transistorer, som er ekstremt små elektriske brytere. Dette mikro-ledningsnettet kobles til bit'ene i de to registerne A og B i den ene enden og et register C som lagrer svaret i den andre enden. Ved hjelp av binær logikk er nettverket av ledninger og transistorene (som avgjør om ledningene leder strøm eller ikke) koblet sammen på en slik måte at uansett hvilke to tall som er lagret i A og B så vil resultatet etter at kretsen er skrudd på bli at tallet C = A + B lagres i resultat-registeret C. Algoritmen som blir brukt for å legge sammen tallene binært er en binær versjon av metoden alle barn lærer på skolen for å legge sammen desimaltall, ved å legge sammen tall for tall fra høyre og føre mente videre. Binær logikk brukes til å lage fysiske nettverk som utfører en slik operasjon og så blir dette utført for hvert bit i registerne. Den binære logikken implementeres med logiske porter som er laget av transistorer.

4) I begge tilfeller må koden kompileres før den kan kjøres på en x86-prosessor.

- Kompileringen av C-kode gir en fil med x86 maskinkode direkte. Disse instruksjonene utføres en for en direkte av x86-prosessoren.
- Kompileringen av Java-kode gir Java bytecode og denne koden inneholder instruksjoner laget for en virtuell maskin, JVM (Java Virtual Machine). Denne virtuelle maskinen, JVM, er et kompilert program som utfører x86-instruksjoner når det kjører for å utføre Java bytecode.

Generelt vil det gå litt raskere å kjøre maskinkode direkte på prosessoren, siden JVM er et ekstra lag med logikk. Men blant annet på grunn av av JIT- (Just In Time) kompilering, er Java nesten like raskt (og uansett vil en så kort algoritme ikke gi noen merkbar forskjell).

Prosesser

5) En context switch er det som skjer når OS stopper en kjørende proses, skifter den ut og setter en annen prosess til å kjøre. I korte trekk:

- OS lagrer all informasjon om den kjørende prosessen i PCB (Process Control Block), inkludert alle verdier i CPU-registre. Også pagetabellen i MMU lagres.
- Den gamle prosessen settes i riktig kø; i ready list om den var klar til å kjøre flere instruksjoner og ikke venter på noe.
- All den tilsvarende informasjonen lastes inn for den nye prosessen, inkludert alle registerverdier slik de var etter at den nye prosessen utførte sin siste instruksjon før denne ble svitsjet ut.
- OS sørger for at den nye prosessen fortsetter med sin neste instruksjon.

6) Multithreading i betydning at flere tråder kjøres samtidig i samme prosess betyr at flere eksekveringer av samme kode kjøres samtidig, enten ved å multitaske på samme CPU med context switching eller ved å kjøre samtidig i reell tid på hver sin CPU-core. I begge tilfeller er det OS som styrer tildeling av CPU-tid

for hver tråd, på samme måte som OS gjør det for uavhengise prosesser (trådene blir schedulert av OS som uavhengige enheter).

Hyperthreading styres derimot av CPU'en selv. Om en CPU har kapasitet for to tråder, scheduleres begge av OS til denne CPU'en. Deretter switcher CPU'en selv mellom de to trådene. Hvis den ene må vente på RAM, kan den andre tråden utføre instruksjoner i ALU, ofte er denne delt mellom de to trådene. Men de har hvert sitt uavhengige sett med registere, slik at det ikke skjer noen context switch når de byttes. Et slikt bytte skjer på nanosekund-nivå og ikke i løpet av mikrosekunder som ved ordinær multithreading (hyperthreading er et spesifikt begrep for Intel-prosessorer. Det generelle begrepet er SMT, Simultaneous Multithreading, og dette må ikke forveksles med vanlig OS-styrt multithreading).

Ja, hvis to tråder fra samme prosess blir schedulert av OS til en CPU med hyperthreading, kan CPU'en på samme måte som forklart over switche mellom de to trådene og dermed utnyttes effekten av hyperthreading.

7) OS schedulerer de 6 prosessene slik at det til enhver tid er 4 av dem som kjører på hver av CPUene. De som venter swithces i løpet av brøkdelen av et sekund fortløpende inn slik at alle de 6 prosessene får omtrent like mye CPU-tid, det vil si 4/6 eller ca 67% CPU-tid.

8) Det er totalt 6x30 = 180 CPU-minutter som må prosesseres. Jevnt fordelt på 4 CPUer vil dette utføres på 180/4 = 45 minutter og etter denne tiden er alle de 6 prosessene ferdig samtidig.

Alternativt kan man si at tilgang til 4/6 CPU i 45 minutter gjør at den enkelte kan utføre (4/6)x45 = 30 CPU-minutter med arbeid.

Synkronisering

9) Hvis to eller flere prosesser samtidig bruker en felles ressurs som kan endre verdi, er det generelt nødvendig å serialisere prosessene slik at de ikke samtidig prøver å endre verdien på denne ressursen.

Et eksempel på dette er om to Java-tråder som kjøres samtidig har kode som endrer på en felles variabel i RAM. Java-koden vil da hente inn verdien fra RAM til CPU (i dette tilfellet på JVM-stack'en) og etter at prosesseringen er ferdig lagre den nye verdien i RAM. Om en annen prosess samtidig prøver å gjøre en endring, enten på en annen CPU eller etter en context switch, vil de kunne overskrive hverandres endringer om de ikke serialiseres slik at en prosess av gangen leser, endrer og lagrer en ny verdi.

10) Et kritisk avsnitt er en del av et program hvor det er nødvendig for en prosess at ingen andre prosesser har tilgang til felles data mens den oppdaterer en felles ressurs for å unngå problemene nevnt i forrige delspørsmål.

11) Scriptet leser inn et tall fra en fil (gitt at det ligger et tall i denne filen), øker verdien med en, venter i 3 sekunder og skriver så den nye verdien ut igjen til filen. Når scriptet kjøres, legges først tallet 0 i filen med en echo-kommando. Deretter økes tallet som er lagret i filen med en hver gang scriptet kjøres.

12) Følgende skjer:

1. Tallet 0 skrives til filen
2. Script nr 1 leser verdien 0 og øker den til 1
3. Etter 1 sekund: Script nr 2 starter, leser verdien 0 og øker den til 1
4. Etter 3 sekunder: Script nr 1 skriver ute verdien 1 og avslutter
5. Etter 4 sekunder: Script nr 2 skriver ute verdien 1 og avslutter
6. Verdien i filen skrives ut med cat og den er da 1

Verdien blir ikke 2 som sist, fordi script nr 2 overskriver script nr 1 sin økning (behandlingen av den felles verdien må serialiseres for å unngå dette).

13) Denne versjonen av scriptet bruker en lock-fil og konvensjonen er at om denne filen eksisterer så skal andre script vente med å lese og endre verdien til lock-filen er fjernet av scriptet som er inne i kritisk avsnitt. Dette sørger while-løkken for, scriptet vil gå i en løkke så lenge filen eksisterer. Følgende skjer:

1. Tallet 0 skrives til filen
2. Script nr 1 starter, lock-filen finnes ikke og det går ikke inn i while-løkken. En tom lock-fil lages. Deretter leser den verdien 0 og øker den til 1 og begynner så å vente med sleep.
3. Etter 1 sekund: Script nr 2 starter, men blir stående å loope i while-løkken siden lock-filen finnes.
4. Etter 3 sekunder: Script nr 1 skriver ute verdien 1, fjerner lock-filen og avslutter. Dermed avslutter script nr 2 while-løkken, leser inn verdien 1 fra filen og øker verdien til 2.
5. Etter 6 sekunder: Script nr 2 skriver ut verdien 2 og fjerner lock-filen.

While-løkken med lås-filen lock sørger for at kun ett script slipper til av gangen i det kritiske avsnittet.

14) For-løkken gjør at de to scriptene starter nesten helt samtidig, deretter vil de multitaskes på samme core eller kjøres samtidig i reell tid på hver sin core. Følgende må ha skjedd:

1. Det første scriptet som kommer til while-løkken tester om lock-filen finnes med -f $lock og det gjør den ikke.
2. Neste script gjør den samme sjekken før det første scriptet har laget en lock-fil, dermed passerer begge script while-løkken og går samtidig inn i kritisk avsnitt og begge leser inn verdien 0.
3. Det første scriptet som blir ferdig med sleep skriver ut verdien 1 og sletter lock-filen.
4. Det andre scriptet som blir ferdig med sleep skriver ut verdien 1 og sletter lock-filen. Men den er allered slettet og feilmeldingen 'No such file or directory' skrives ut.
5. Når teller skrives ut med cat vil verdien dermed være 1.

En slik kjøring kunne ha endt med at tallet ble 2 hvis det første scriptet hadde rukket å lage lock-filen før neste script gikk inn i while-løkken. Men hvis de gjør det samtidig, kan man risikere at de ødelegger for hverandre. Det finnes ingen enkel løsning på dette. Peterson-algoritmen er en litt kompleks løsning som kun virker for to tråder. Scriptene trenger hjelp fra OS for å serialiseres, eventuelt kan atomiske operasjoner benyttes, som tester og endrer en verdi uten at andre prosesser kan få aksess til verdien samtidig.

Linux kommandolinje

15) Mappen over den du står i

16) `/usr/bin/diff` (eneste som starter med /)

17) `regn > res.txt`

18) `*`

19) Forsøkene feiler fordi:

1. $prefixfase blir tolket som en annen og ny variabel og den er tom
2. $(perfix)fase er et forsøk på å kjøre kommandoen 'prefix' og den finnes ikke
3. Siste forsøk virker nesten, men skriver ut et mellomrom

Tre måter for å få det til på:

1. `echo ${prefix}fase`
2. `echo "$prefix"fase`
3. `echo $prefix"fase"`

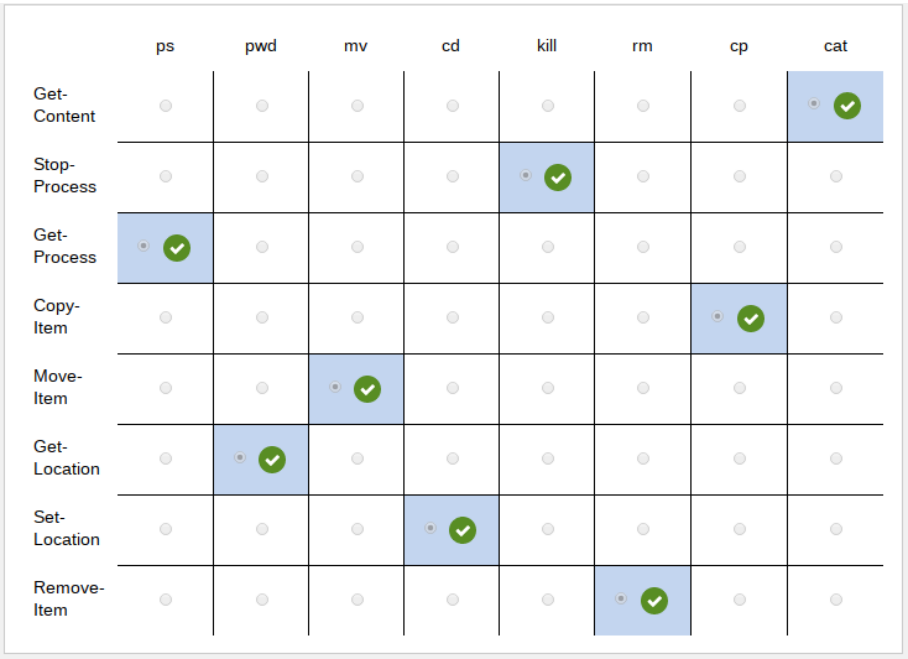
Bash-scripting

20)

```
#! /bin/bash

for i in a b c
do
   mkdir ${i}dir
   mv ${i}* ${i}dir
done
```

PowerShell



21)

22)

```
foreach ($ls in ls -r C:\ 2> $null){
    if($ls.Extension -eq ".ps1"){
    $sum += $ls.length
    }
}
echo "Antall bytes av filer med extension .ps1 under C: $sum"
```

Internminne

23) Betydningen av uttrykket random access er at det ikke er noen forskjell på tiden det skal ta å lese eller skrive ethvert byte i RAM. Dette vil vanligvis ikke være oppfylt da alle servere bruker cache og bytes som ligger i cache er raskere å hente enn om man må helt ut i RAM for å hente en gitt byte. Et annet tilfelle er servere med numa-noder; for disse vil noen deler av RAM ha lenger aksesstid en andre for den enkelte CPU.

24) Pages

25) Paritets bit