Løsningsforslag Eksamen vår 2019 Operativsystemer

Datamaskinarkitektur

1) AND

2) En OR-port

3) $A \cdot B + \overline{A} \cdot B$

4) $A \cdot B + \overline{A} \cdot B = (A + \overline{A}) \cdot B = 1 \cdot B = B$

Sannhetstabellen vil se slik ut:

A	B	F
0	0	0
0	1	1
1	0	0
1	1	1

slik at man kan se at F = B.

5) Prosesseringen av en krets av porter går veldig raskt, men det tar litt tid før resultatet er klart og så må mellomlagres før neste operasjon. Resultatet kan være input til neste beregning, men før det lagres må man være sikker på at beregningen er helt ferdig. Uten en klokke som synkroniserer dette, vil resultatene ukontrollert flyte over i neste beregning. Klokken styrer vipper (som er satt sammen av en master og en slave lås) slik at de venter med å gi resultatet videre til man er helt sikker på at kretsen har regnet seg ferdig. I eksempelet med studenter som var låser (latches), så man at informasjonen fløt ukontrollert og usynkront om man ikke organiserte dem to og to med en klokke som sørget for overføring av informasjon med faste intervaller.

6) Maskinkoden for `add %rax, %rbx` er en binær streng som inneholder et nummer som sier hvilken instruksjon som skal utføres (add) og hvilke to registere som skal inngå i operasjonen. Instruksjonsdekoderen tolker alle bit i instruksjonen og setter så alle nødvendige kontroll-bit i ALUen på en slik måte at verdien i registerne rax og rbx sendes inn til ALUen, den legger sammen disse to verdiene og resulatet sendes til register rbx.

Linux kommandolinje

7) mkdir

8) Mappen over den du står i

9) hei du der

10) `find /home/hh/div -type d` betyr finn alle mapper under `/home/hh/div` inkludert undermapper. Fra manualsiden:

```
-newerXY reference
          Succeeds  if  timestamp X of the file being considered is newer than timestamp Y of the
          file reference. The letters X and Y can be any of the following letters:

          a   The access time of the file reference
          B   The birth time of the file reference
          c   The inode status change time of reference
          m   The modification time of the file reference
          t   reference is interpreted directly as a time
```

slik at -newermt betyr at den slår til hvis modifisering av mappen er nyere enn det angitte tidspunktet. `-ls` er en action som `list current file in ls -dils format on standard output` Kommandoen gir derfor en listing av alle mapper som har blitt endret etter tidspunktet `21 Apr 2019 00:00`.

Bash-scripting

11)

```
#! /bin/bash

for fil in *.cnf
do
   python runML.py $fil &
done
```

12) Siden serveren har 32 uavhengige CPUer, vil de 30 jobbene fordeles på 30 CPUer, hver enkelt jobb og dermed hele jobben tar da ca en time siden de kjører i paralell.

13) Hun kan oppnå dette ved å starte en screen-session på research3, slik som dette:

```
research3:~$ screen -S run
```

deretter kan hun starte scriptet i dette vinduet. Nå kan hun logge ut og screen vil fortsette i bakgrunnen (kan også gå ut av screen med CTRL-A CTRL-D, men det er ikke nødvendig). Så kan hun hjemmefra koble seg til research3 med ssh og åpne sin screen-session med

```
research3:~$ screen -r run
```

Om hun ikke ga sin screen-session et navn med -S når hun startet den, kan hun ved hjelp av

```
research3:~$ screen -ls
```

finne sin session og så bruke det navnet med -r (eller bare bruke -r om det kun finnes en eneste screen-session).

Et alternativ er å starte scriptet på research3 med

```
research3:~$ run.sh  > run.log 2>&1 &
```

som legger både output og stderr i filen run.log. Deretter kan hun logge seg på med ssh hjemmefra og se output i filen `run.log`.

Hvis hun lokalt (fra en annen host på jobben) logger seg på med ssh, kan det være nødvendig å legge til `</dev/null`, ellers kan i noen tilfeller bakgrunnsjobben bli stoppet når ssh-tilkoblingen tas ned:

```
research3:~$ run.sh  </dev/null > run.log 2>&1 &
```

14)

```
#! /bin/bash
(( dtot=0 ))
(( ftot=0 ))
```

```
(( ltot=0 ))
for fil in $(find .)
do
    if [ -L "$fil" ]; then
           (( ltot += 1 ))
    elif [ -f "$fil" ]; then
           (( ftot += 1 ))
    elif [ -d "$fil" ]; then
           (( dtot += 1 ))
    fi
done
echo Filer: $ftot
echo Kataloger: $dtot
echo Linker: $ltot
```

Det er viktig å ha link-testen -L først, ellers vil linker bli talt opp som fil eller mappe etter hva de peker på. Litt pirk: Et problem med denne løsningen er at den ikke fungerer for filer som har mellomrom i navnet. Følgende løsning:

```
#! /bin/bash

ltot=$(find . -type l | wc -l)
dtot=$(find . -type d | wc -l)
ftot=$(find . -type f | wc -l)

echo Filer: $ftot
echo Kataloger: $dtot
echo Linker: $ltot
```

fungerer for filnavn med mellomrom også, men går igjennom filsystemet tre ganger (og find -type f teller ikke linker som peker til filer). En annen korrekt løsning som går igjennom filsystemet kun en gang er:

```
#! /bin/bash
(( dtot=0 ))
(( ftot=0 ))
(( ltot=0 ))

while read fil
do
    if [ -L "$fil" ]; then
           (( ltot += 1 ))
    elif [ -f "$fil" ]; then
           (( ftot += 1 ))
    elif [ -d "$fil" ]; then
           (( dtot += 1 ))
    fi
done < <(find .)

echo Filer: $ftot
echo Kataloger: $dtot
echo Linker: $ltot
```

hvor den spesielle konstruksjonen `<(` fungerer som en slags baklengs pipe. Det kalles process substitution og får en prosess sin output til å opptre som et filnavn. (Om man pipe't find-kommandoen inn til while, ville det samme skjedd, bortsett fra at variablene da ikke ville bli bevart etter løkken.)

Prosesser

15) Et operativsystem får det til å se ut som om mange prosesser kan kjøre samtidig ved å dele opp tiden i små biter og la hver prosess som kjører få en bit CPU-tid (størrelsesorden et hundredels sekund) av gangen i en Round Robin kø. Prosesser som trenger rask respons (som teksteditorer og web-browsere) vil gjennom tastatur og muse-interrupts sende beskjed, slik at de raskt blir behandlet. En hardware timer sender jevnlig et interrupt-signal som gjør at OS kan ta over kontrollen over CPUen og umiddelbart sørge for å gi respons til prosesser som krever rask respons. Dermed unngås det også at vanlige brukerprosesse tar over styringen selvom de kjører instruksjoner direkte på CPUen.

16) Et systemkall utføres når en prosess i user-mode ber OS-kjernen om en tjeneste. Det kan være tjenester som å skrive til disk eller å opprette en ny prosess, alle operasjoner som er kontrollert av OS-kjernen som dermed i høyeste grad er involvert. Totalt sett utgjør alle systemkall et API mot OS-kjernen som en bruker-prosess kan benytte seg av. Hvis det å utføre et systemkall tar litt tid, som for eksempel å lese en fil fra disken, vil prosessen bli satt på vent til systemkallet er ferdig. Dette er såkalte blokkerende systemkall. I eksempelet med vaffel-prosessen var det å hente melk et systemkall som tok relativt lang tid, dermed ble vaffel-prosessen satt på vent og forelesnings-prosessen tok over. Knusing av egg var derimot et ikke-blokkerende systemkall og ble utført med en gang. I begge tilfeller var det OS-kjernen som utførte operasjonen som bruker-prosessen (vaffellaging) ba om.

17) Ja, etter en context switch starter prosessen opp igjen på en annen CPU istedet for den samme. Registerverdier etc legges da over på den nye CPUen istedet for på den gamle. Dette skjer ofte hvis det er flere CPU-intensive prosesser enn CPUer; da bytter prosessene på å kjøre og blir ikke startet opp på samme CPU hver gang (men pga cache, velges om mulig samme CPU når en prosess skal startes på nytt). Sidene i RAM må ikke flyttes, RAM deles av alle CPUene i systemet og sidene blir liggende på samme sted.

18) En tråd er den sammenhengende rekken av hendelser/instruksjoner som utføres når et program kjøres. I en prosess kan flere slike rekker av instruksjoner utføres samtidig, de kan være på forskjellige steder i samme felles kode og endre på både egne og felles variabler. Har man flere uavhengige CPU-cores kan trådene utføres samtidig innenfor den samme prosessen; med kun en core bytter de på å kjøre.

PowerShell

19)

```
PS C:\Users\os> $now = Get-Date                    # legger nåværende dag og tid i variabelen $now
PS C:\Users\os> $then = Get-Date "20 Januar 2019" # Legger dette tidspunktet i $then
PS C:\Users\os> $then                              # Skriver ut dette tidspunktet (tilsvarer echo $then)

søndag 20. januar 2019 00.00.00

PS C:\Users\os> $then -lt $now         # Tester om $then er mindre enn $now og
True                                   # testen er sann siden $then skjedde tidligere enn $now
```

20)

```
(ls fil.txt).CreationTime.Year
```

21)

```
ls -r C:\Users\os | Where-Object {$_.LastWriteTime -gt (Get-Date "20 januar 2019") -and $_.LastWriteTime -lt (Get-Date "21 januar 2019")}
```

22) PowerShell-kommandoen returnerer et helt objekt med properties og en av disse er LastAccessTime, dermed kan man også få ut dette tidspunktet med kommandoen:

```
(ls fil.txt).LastAccessTime
```

Linux-kommandoen gir kun tekststrengen med datoen som vist over.

Plattformuavhengighet

23) Ubuntu 16.04 med AMD-prosessor (Kun et Linux-OS kan kjøre denne koden; både AMD og Intel er x86-basert)

24) Ja, dette vil virke fordi class-koden kjøres i en JVM (Java Virtual Machine) som er kompilert for hver av plattformene og class-koden kjøres og tolkes da på samme måte. Men svaret vil avhenge av at de to JVM'ene har samme hovedversjon, ikke alle hovedversjoner av JVM er kompatible med hverandre.

Internminne

25) 4 byte*40 000*40 000/4 000 byte = 4*40 000*10 = 1 600 000. (det eksakte tallet, som det ikke blir spurt etter, er 1 562 500) og man kan se at det er ca en minor fault pr side. En minor page fault vil skje når det ikke er laget en entry for den i MMU og det vil lages en for hver side som er med i programmet.

26) Den store forskjellen skyldes at cache utnyttes mye bedre i det første tilfellet. Dette igjen skyldes hvordan et array er lagret i RAM. array[0][0], array[0][1], array[0][2], etc ligger etterhverandre i minnet. Når variabelen b endres i den indre løkken, vil derfor sammenhengende deler av RAM skrives til fortløpende. Dermed blir bruken av cahce veldig effektiv. Når man bytter om på [a][b] til [b][a], vil det skrives til helt forskjellige steder i RAM hver gang (4*40 000 bytes fra hverandre) og cahce-bruken vil bli veldig ineffektiv.

Output fra perf viser at det er veldig mange flere cache-referanser og misses for den andre versjonen, spesielt for data og TLB, slik at det i siste linje blir 2500 Mega bus-cycles i motsetning til 200 Mega bus-cycles (men minor og major page-faults er det samme, så det skyldes ikke paging). Skriving til RAM står trolig for det meste av tidsbruken og det andre programmet bruker da 12.5 ganger så lang tid.