Løsningsforslag Eksamen 2020 Operativsystemer

Datamaskinarkitektur

1) 00

2) Grunnen til dette er relatert til pipelining og branch predicition. Hver instruksjon blir delt opp i flere deler og kjøres delvis i parallell. Men en if-test er en forgrening (branch) og dermed kan man ikke vite hva som er neste instruksjon. Derfor gjetter CPU'en basert på erfaring hvilken branch som vil bli fulgt og legger tilsvarende instruksjoner inn i pipe'en. Med et usortert array vil det hele tiden variere hvilken branch som følges og dermed vil gjettingene ofte være feil. Men når arrayet er sortert vil i første halvdel av programmet samme forgreining følges og i andre halvdel den andre forgreiningen følges. Dermed vil branch-prediction nesten alltid lykkes og programmet går mer enn dobbelt så fort som når den feiler omtrent halvparten av gangene med et usortert array. (Om man fjerner if-testen helt fra programmet, går det omtrent like fort som kjøringen med et sortert array)

Threads og prosesser

3) Ressursdeling: flere prosesser deler på kode, data og delvis PCB innenfor samme tråd. (Skal være "flere tråder ... innenfor samme prosess")

4) Bruker en hardware timer til å gi begrensede tidsintervall til brukerprosessene

5) user mode

Serialisering

6) blockRaceCondition()

7) At en prosess venter i en while-løkke til en hendelse intreffer

Internminne

8) 256 (1 MiB = 2^{20}, 4096 = 2^{12}, 1 MiB/4096 = 2^8 = 256)

9) En side som har blitt endret slik at den må skrives til disk om den må ut av minnet

10) Fordi minne-aksess da ville ha tatt alt for lang tid

Docker og prosesser

11)

```
gcc -O prime.c
time ./a.out
```

Docker og prosesser

12)

```
FROM ubuntu # Bruk siste versjon av Ubuntu
COPY prime.c prime.c # Kopier filen prime.c fra mappen du kompilerer i til en fil med samme navn i imaget
RUN gcc -O prime.c   # Kompiler filen i dette imaget
RUN time ./a.out > res.txt # Kjør programmet og ta tiden på det (i imaget som bygges, før containeren har startet)
                           # og skriv resultatet til res.txt
```

Bygges med

```
docker build -t prime .
```

13) Dette skyldes at gcc ikke er installert i et default Ubuntu-image. Det må installeres først, noe som ordenes om man legger til følgende linjer etter FROM ubuntu:

```
RUN apt-get update -y
RUN apt-get install -y gcc # Eventuelt build-essential
```

14) Av feilmeldingen kan man se at Dockerfile-kommandoene kjøres av /bin/sh og av siste kommando at det er dash-shellet som kjøres. Første kommando som kjøres viser at a.out kan kjøres i dette shellet, men andre kommando viser at kommandoen 'time' ikke finnes i dette shellet og dette er grunnen til feilmeldingen. En annen strategi kan være å legge time-kommandoen inn i et bash-script og så kjøre dette scriptet under byggingen med RUN.

15)

```
root@osG70:~/prime# time ./a.out
25997 primes below 300000
Real:14,314 User:14,284 System:0,032 100,01%
----- All info, STDOUT fra a.out og STDERR fra time, skrives til terminal-vinduet

root@osG70:~/prime# time ./a.out > res.txt
Real:14,190 User:14,174 System:0,021 100,04%
----- STDOUT legges i res.txt, STDERR til terminal

root@osG70:~/prime# time ./a.out &> res.txt
Real:14,173 User:14,146 System:0,032 100,04%
----- STDOUT og STDERR fra a.out legges i res.txt, STDERR fra time går fortsatt til terminal

root@osG70:~/prime# (time ./a.out) > res.txt
Real:14,308 User:14,281 System:0,029 100,01%
----- Lager et sub-shell slik at output fra time og a.out slås sammen,
----- men bare STDOUT sendes til res.txt

root@osG70:~/prime# (time ./a.out) &> res.txt
----- Lager et sub-shell slik at output fra time og a.out slås sammen,
----- både STDOUT og STDERR sendes til res.txt

root@osG70:~/prime# cat res.txt
25997 primes below 300000
Real:14,275 User:14,235 System:0,038 99,99%
----- Viser effekten av forrige kommando
```

16)

```
#! /bin/bash

TIMEFORMAT="Real:%R User:%U System:%S %P%%"
(time ./a.out) > res.txt 2> tid.txt
```

17)

```
FROM ubuntu
RUN apt-get -y update
RUN apt-get -y install gcc
```

```
COPY prime.c prime.c
RUN gcc -O prime.c
COPY run.sh run.sh
RUN chmod 755 run.sh # Behøves ikke om allerede kjørerettigheter i mappen
CMD ["./run.sh"]     # Kjøres etter oppstart av containeren, ikke under bygging
```

18)

```
docker container run  -d prime
```

Problemet er at containeren avslutter når kommandoen som kjøres med CMD avsluttes. Dette kan løses ved å legge til en sleep-kommando på slutten av run.sh, for eksempel

```
sleep 600
```

som gjør at det går 10 minutter før containeren avsluttes.

19)

```
root@osG70:~/prime# docker container ps
CONTAINER ID        IMAGE      COMMAND      CREATED                 STATUS                NAMES
31c5c5fbde8b        prime        "./run.sh"         46 seconds ago    Up 44 seconds   goofy_stallman
----- Viser at containeren fortsatt kjører

root@osG70:~/prime# docker container exec  31c5 cat res.txt
25997 primes below 300000
----- Skriver ut innholdet av res.txt (31c5 er entydig og derfor nok) ved å kjøre cat-kommandoen

root@osG70:~/prime# docker container exec  31c5 cat tid.txt
Real:14.226 User:14.202 System:0.032 100.05%
----- Skriver ut innholdet av tid.txt

root@osG70:~/prime# time ./a.out
25997 primes below 300000
Real:14,290 User:14,259 System:0,027 99,98%
----- Kjører a.out på serveren og det tar 14.29 sekunder.
```

Dette viser at det går like fort og er like effektivt å kjøre i en container som direkte på serveren (og i praksis kjøres prosessen direkte på serveren når den startes i en container).

20)

```
#! /bin/bash

TIMEFORMAT="Real:%R User:%U System:%S %P%%"
for i in {1..4}
do
   (time ./a.out) > res$i.txt 2> time$i.txt &
done

sleep 600
```

21)

```
for i in {1..4}; do docker container exec 4d89 cat time$i.txt; done
```

Alternativt:

```
docker container exec 4d89 cat time?.txt
```

22) Scriptet starter fire uavhengige prosesser og vi kan se av output fra time at i begge tilfeller kjører de like fort som om en prosess kjører alene. Av prosent-tallet på slutten av linjene ser vi også at i begge tilfeller får hver prosess ca 100% av hver sin CPU. Slik at vi kan konkludere med at prosesser som kjører i en Docker-container kan utnytte alle CPUene. (Dette er ikke overraskende siden prosessene kjører direkte på serveren inne i containeren, kun med diverse beskyttelsesmekanismer som isolerer dem noe mot andre prosesser. Men om man kjører 'docker container run' med opsjonen –cpus 1, vil docker begrense containteren til å kun bruke en enkelt CPU og i prinsippet kunne dette vært en default konfigurasjon for Docker)

Bash-script

23)

```
#! /bin/bash

echo "" > tesla.log # Tømmer eventuelt innhold i filen
vid=29583758204
token=30ab56c57d37f244db35a23c4b
url=https://owner-api.teslamotors.com/api/1/vehicles/$vid/data_request/drive_state
while [ 1 ]
do
   speed=$(curl -s -X GET $url -H "Authorization: Bearer $token"  | jq '.response | .speed')
   echo -n "$speed " >> tesla.log # -n unngår linjeskift
   echo $(date) >> tesla.log
done
```

24)

```
sort -rn tesla.log # n gir numerisk sort, -r gir reverse sort, største tall først
```

PowerShell

25)

```
 $time1 = Measure-Command{.\run1.ps1}
 $time2 = Measure-Command{.\run2.ps1}
 $diff = $time1 - $time2
 $sec = $diff.TotalSeconds

"run1.ps1 bruker $sec sekunder mer enn run2.ps1"
```