

Ukesoppgaver 11 – Scheduling, trap, systemkall og ticks, Dockerfiles

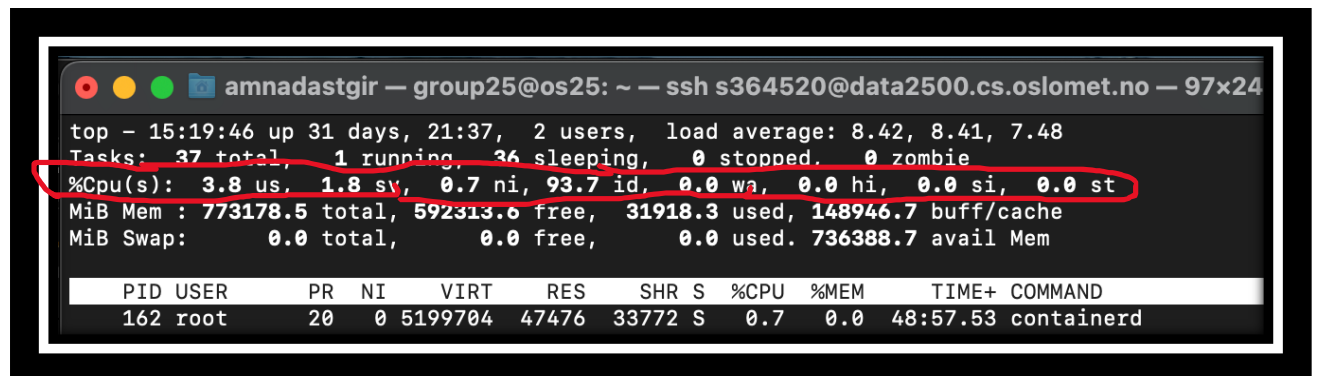
RØD - Obligoppgaver

GUL – Ikke obligoppgaver

TURKIS – Ukens nøtt og utfordringer

1. (Oblig)

Under tastet jeg «top».



```
amnadastgir — group25@os25: ~ — ssh s364520@data2500.cs.oslomet.no — 97x24
top - 15:19:46 up 31 days, 21:37, 2 users, load average: 8.42, 8.41, 7.48
Tasks: 37 total, 1 running, 36 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.8 us, 1.8 sy, 0.7 ni, 93.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 773178.5 total, 592313.0 free, 31918.3 used, 148946.7 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used, 736388.7 avail Mem

  PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
  162 root        20   0 5199704 47476 33772 S   0.7   0.0 48:57.53 containerd
```

Som outputen fra «top» viser er det en oversikt over systemprosessene og systemressursbruk.

Her er hva de forskjellige kolonnene i CPU-seksjonen betyr (markert i rødt):

- %Cpu(s): er en oversikt over CPU-bruk i prosent
 - Us (user): % av CPU-bruk brukt av brukerprosesser
 - Sy (system): % av CPU-bruk brukt av systemprosesser
 - Ni (nice): % av CPU-bruk brukt av prosesser med en endret prioritet (nice-verdi)
 - Id (idle): % av CPU-tid som ikke brukes, dvs. at ingen prosesser er aktivt i bruk på CPU
 - Wa (wait): % av CPU-tid brukt på å vente på I/O-operasjoner (som diskoperasjoner)
 - Hi (hardware interrupts): % av CPU-tid brukt på å håndtere maskinvare avbrytelser (som fra harddiskkontroller eller nettverkskort)
 - Si (software interrupts): % av CPU-tid brukt på å håndtere programvare avbrytelser (som fra tastatur eller mus)
 - St (steal time): % av CPU-tid “stjålet» av en annen virtuell maskin som deler CPU med denne

Man kan se at 94% av CPU-tiden er ledig, som betyr at CPU-ressursene ikke utnyttes fullt ut, og at det er potensial for å kjøre flere prosesser eller applikasjoner uten å belaste systemet for mye. Det er viktig å merke seg at å utnytte CPU-ressurser bedre ikke nødvendigvis betyr å bruke dem fullt, men man må finne en balanse mellom ytelse, stabilitet og pålitelighet for å optimalisere systemet på en effektiv måte.

2. (Ikke oblig)

Trap funksjonens forklaring fra mitt eget notat for uke 11 forelesninger:

TRAP

En «trap» i programmering referer til en hendelse eller et signal som kan avbryte kjøringen av en kode eller programvare. Traps brukes vanligvis til feilhåndtering og debugging, og kan brukes til å fange opp feil og avvik som oppstår under kjøring av programvaren. I tillegg kan traps brukes til å håndtere spesielle situasjoner som krever at programvaren stopper eller utfører spesielle handlinger. I operativsystemer brukes traps også til å benytte mellom user mode og kernel mode. Trap switcher til kernel mode og hopper til kode for systemkall i en operasjon.

Side 4 av 23

3. (Ikke oblig)

Etter research er noen **fordeler** med kortest mulig jiffie/tick lengde:

1. bedre responsivitet: kortere tick lengde vil øke nøyaktigheten og responsiviteten i systemet. Viktig i sanntidsapplikasjoner der små forsinkelser kan være kritiske
2. bedre utnyttelse av CPU-ressurser: kortere tick lengde bidrar til å utnytte CPU-ressurser bedre, spesielt på systemer med mange prosessorer. Kan føre til høyere systemytelse og bedre utnyttelse av tilgjengelige ressurser

Noen **ulemp**er med kortest mulig jiffie/tick lengde:

1. høyere CPU-bruk: kortere tick lengde vil øke antall interrupts som sendes til CPU-en, noe som kan føre til høyere CPU-bruk og økt strømbruk. Kan påvirke systemets ytelse og batterilevetid på mobile enheter

2. potensielle konflikter med annen maskinvare: en kortere tick kan føre til konflikter med annen maskinvare som bruker interrupts, f eks nettverkskort eller grafikkort. Kan føre til økt systembelastning og lavere ytelse
3. potensielle problemer med eldre programvare: noen eldre programvare kan ikke være kompatibel med en kortere tick lengde, som kan føre til stabilitetsproblemer eller feil i systemet

Vi kan med få ord si at kortere tick lengde gir bedre systemrespons og utnyttelse av ressurser, men kan føre til høyere CPU-bruk.

4. (Oblig)

Som lært i forelesning bruker Linux-kjernen et timer-interrupt for å fordele CPU-tid mellom alle prosesser på en rettferdig måte og unngår at en enkelt prosess kan ta over styringen av systemet. Timer-interruptet fungerer ved at hardware-timeren utløses hvert hundredels sekund, og når den utløses, bytter prosessoren til kjernemodus.

I kjernemodus har Linux-kjernen full kontroll over systemressursene og kan ta beslutninger om hvilken prosess som skal få tilgang til CPU-en og hvor lenge. Timer-interruptet gir kjernen muligheten til å fordele CPU-tid mellom alle prosessene og sikrer at ingen enkelt prosess monopoliserer CPU-en.

Uten timer-interruptet kan en prosess i teorien holde CPU-en opptatt hele tiden, og andre prosesser vil ikke få tilgang til ressursene de trenger. Dette kan føre til lav systemytelse, uforutsigbare responstider og en ustabil systemytelse. Ved å bruke timer-interruptet kan Linux-kjernen sørge for at ressursene fordeles rettferdig mellom alle prosesser og systemet kan kjøres jevnt og stabilt.

5. (Ikke oblig)

Operativsystemet kan prioritere prosesser forskjellig ved hjelp av tidsinndelingen som hardware-timeren gir gjennom jiffies eller ticks. Under er det to måter som jeg har funnet os kan bruke tidsinndelingen til for å prioritere prosesser:

1. **Round-robin scheduling**: vanlig prosessplanleggingsalgoritme som bruker ticks til å rotere gjennom alle prosessene og tildele hver prosess en lik andel av CPU-tiden.

Dette sikrer at alle prosessene får en rettferdig del av CPU-ressursene, og ingen prosess monopoliserer CPU-en.

2. **Priority scheduling**: annen prosessplanleggingsalgoritme som bruker ticks til å tildele prosesser med høyere prioritet mer CPU-tid enn prosesser med lavere prioritet. Os kan justere prioritetene til prosessene basert på faktorer som viktighet, interaktivitet eller andre kriterier. Dette sikrer at viktige eller interaktive prosesser får mer CPU-tid og kan gi bedre systemrespons og ytelse.

I begge tilfeller kan tidsinndelingen som hardware-timeren gir, brukes til å fordele CPU-ressurser på en effektiv måte og prioritere prosesser på en rettferdig og effektiv måte.

6. (Oblig)

Det er først og fremst viktig å bemerke når en trap og når et interrupt oppstår og hvordan de håndteres av systemet. En trap utløses av programvare og krever at os griper inn, mens et interrupt oppstår som følge av en ekstern hendelse og håndteres av maskinvaren.

En trap er en type instruksjon som er ment å bli utført i programvarekoden og krever privilegier som bare os kan gi. Når en trap utløses, vil professoren bytte til kjernemodis og utføre instruksjonen i os for å håndtere forespørselen. Traps brukes vanligvis til å implementere systemkall og håndtere feil i applikasjoner.

Et interrupt, derimot, oppstår når en ekstern hendelse (for eksempel en I/O-avbrudd eller en maskinvarefeil) inntreffer mens prosessoren utfører en instruksjon. Interrupts håndteres av maskinvaren, som avbryter den nåværende instruksjonen og bytter til en annen prosess eller prosedyre som kan håndtere hendelsen. Dette gjør at operativsystemet kan håndtere eksterne hendelser på en effektiv og pålitelig måte.

7. (Ikke oblig)

Mekanisme som os bruker til å tillate applikasjoner for å få tilgang til systemressurser og kjernemodus-funksjoner som ikke er tilgjengelige i brukermodus. Systemkall gir et grensesnitt mellom applikasjoner og os-kjernen, og tillater applikasjoner å be om tjenester som å lese eller skrive filer, kommunisere med enheter, opprette prosesser, administrere minne og utføre andre operasjoner som krever privilegier.

Hensikten med et systemkall er å gi applikasjoner en standardisert og kontrollert måte å be om tjenester og få tilgang til systemressurser i operativsystemet på. Systemkall gir en sikker og abstrahert tilgang til systemet, uavhengig av maskinvare eller arkitektur, og gjør det enklere for utviklere å skrive bærbar applikasjoner.

8. (Oblig)

Her vil hver prosess kjøre henholdsvis til sin utdelte timeslice. Fordelingen av CPU-tid er:

1. prosessen med det lengste timeslice (A) vil kjøre i 30 ticks, som er 300 millisekunder. Når tiden er ute, vil prosessen legges tilbake i ready-list og neste prosess velges.
2. prosessen med det nest lengste timeslice (B) vil kjøre etter i 20 ticks, som er 200 millisekunder. Slik som A vil denne også legges tilbake i ready-lista og neste prosess velges.
3. da vil prosessen med det korteste tildelte timeslice (C) kjøres i 10 ticks, som er 100 millisekunder. Når tiden er ute vil prosessen også legges tilbake i ready-list.

Dette skjer til alle prosessene har fått kjøre en gang hver i løpet av epoken. Hvis det er flere i ready-listen, vil de bli valgt i henhold til deres tildelte timeslice også kjøres i det angitte tidsrommet før de legges tilbake i ready-listen og neste prosess blir valgt.

9. (Oblig)

Antall timer-interrupts som vil skje i denne epoken vil være lik antall prosesser som er 3. antall context switches vil være det samme som ganger en prosess blir bytta ut med en annen, som vil være to ganger hver epoke. Den første context switchen er fra prosess A til B, og den andre er fra B til C.

10. (Oblig)

Når epoken er over, vil tildelingen av CPU-tid til hver prosess bestemmes av os igjen, og det er ikke fastbestemt at hver prosess får like mange ticks som før. Dette er fordi os tar hensyn til en rekke faktorer når CPU-tiden skal bestemmes, slik som prioritet og antall andre prosesser som også venter på å kjøre.

11. UKENS NØTT 1

Skjerm bilde fra egne notater: svaret i blått:

NEED RESCHED

- Hvis det i løpet av epoken kommer et interrupt, vil et flagg `need_resched` bli satt
- Scheduler vil på grunn av dette kjøres etter neste timer-tick.
- Interaktive prosessen gis dynamisk høy prioritet og får dermed bedre responstid
- Ellers måtte de vente helt til en kjørende CPU-intensiv prosess var ferdig med alle sine ticks i en epoke

I Linux-kjernen blir flagget «`need_resched`» satt hvis det oppstår et interrupt mens en prosess er i kjøring. Dette indikerer at kjernen må planlegge omfordeling av CPU-ressurser for å gi den nye oppgaven en sjanse til å kjøre umiddelbart. Ved neste mulighet vil kjernen planlegge omfordeling av CPU-tiden for å sikre at alle oppgavene får en rettferdig andel av CPU-ressursene. Dette er en mekanisme som sikrer effektiv multitasking og ressursstyring i operativsystemet.

Når flagget «`need_resched`» settes på grunn av et interrupt i en epoke, vil scheduleren starte på nytt etter timer-tick. Dette gir muligheten til å gi interaktive prosesser, som trenger rask responstid, en høyere prioritet og dermed få en raskere respons fra systemet. Uten denne mekanismen ville interaktive prosesser måtte vente til CPU-intensive prosesser var ferdige med sin tid, noe som kunne føre til lang ventetid og en mindre responsiv brukeropplevelse.

12. UKENS NØTT 2

Tiden det tar fra en bruker taster et tegn på tastaturet til det syntes på skjermen i en teksteditor under Linux 2.6 vil avhenge av flere faktorer, inkludert responstiden til tastaturet, hastigheten på datamaskinen og antall prosesser som kjører samtidig. Men ettersom flagget `need_resched` prioriterer input-hendelser, kan responstiden for tasturinndata være ganske rask.

Hvis lengden av en tick endres, vil tiden det tar fra en bruker taster et tegn på tastaturet til det vises på skjermen også endres. En kortere tick vil gi mer presis tidsinndeling og kan dermed øke responstiden for input hendelser, men det kan også føre til mer overhead og økt CPU-belastning på systemet. En lengre tick kan gi mindre overhead og mindre CPU-belastning, men kan også redusere responstiden for inputhendelser. Derfor er det viktig å finne en passende lengde på tick for å balansere mellom presisjon og overhead.

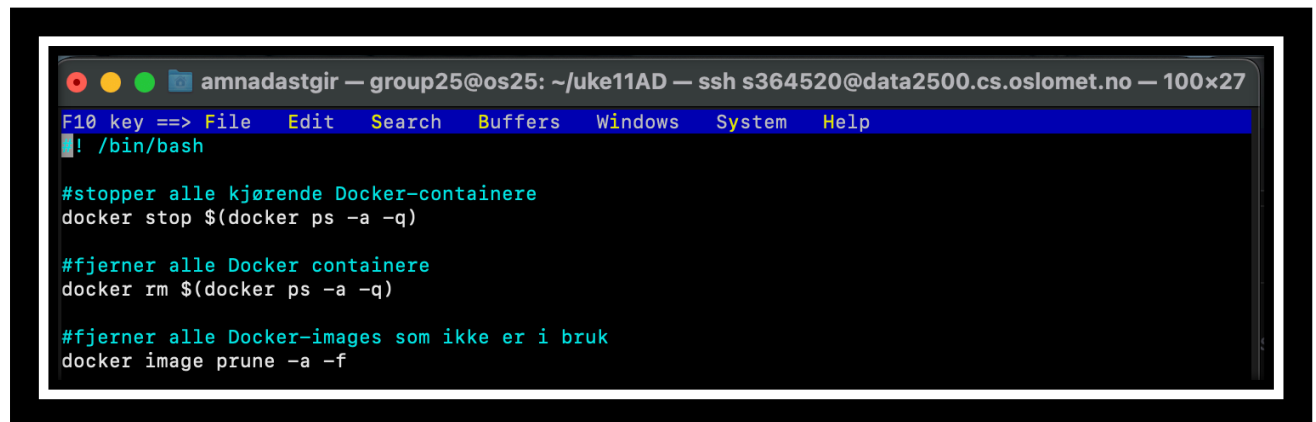
13. UKENS NØTT 3

Dersom 2.6-kjernen ikke bruker `need_resched`, vil det ta ca 100 ms for at teksteditoren får tildelt CPU-tid etter at regnejobben er ferdig med sin timeslice (på 10 ticks). Dette er fordi at os ikke da vil klare å prioritere teksteditoren før alle andre prosesser med høyere prioritet har fått sin CPU-tid, som kan ta opp til 100 ms (i dette tilfellet). Så hvis man taster et tegn på tasturet, vil det ta ca 100 ms før tegnet vises på skjermen i teksteditoren.

Hvis `need_resched` brukes, på den andre siden, vil det tvinge scheduleren til å kjøre med engang etter at interruptet fra tastaturet har registrert seg. Det betyr at teksteditoren vil få tildelt CPU-tid med engang, selv om regnejobben ikke har brukt om sin timeslice.

14. (Oblig)

Under er et bilde av skriptet:




```
amnadastgir — group25@os25: ~/uke11AD — ssh s364520@data2500.cs.oslomet.no — 100x27
F10 key ==> File Edit Search Buffers Windows System Help
!! /bin/bash

#stopper alle kjørende Docker-containerere
docker stop $(docker ps -a -q)

#fjerner alle Docker containere
docker rm $(docker ps -a -q)

#fjerner alle Docker-images som ikke er i bruk
docker image prune -a -f
```

Endret rettigheter



```
group25@os25:~/uke11AD$ chmod 700 oppg14.sh
group25@os25:~/uke11AD$ ls -l
```

Kjøring:



```
group25@os25:~/uke11AD$ sudo ./oppg14.sh
"docker stop" requires at least 1 argument.
See 'docker stop --help'.

Usage:  docker stop [OPTIONS] CONTAINER [CONTAINER...]

Stop one or more running containers
"docker rm" requires at least 1 argument.
See 'docker rm --help'.

Usage:  docker rm [OPTIONS] CONTAINER [CONTAINER...]

Remove one or more containers
Total reclaimed space: 0B
```

15. (Oblig)

Koden til Dockerfile: Filen «Dockerfile» er en oppskrift for å bygge et Docker image. Docker images er «templates» eller «mal» for å lage Docker-containerer.

I denne Dockerfilen bruker vi ubuntu som base image, og deretter utfører vi to kommandoer for å installere Apache webserver og kopiere filen «index.html» fra vår lokale maskin til Docker-containeren.

```
amnadastgir — group25@os25: ~/uke11AD/oppg15 — ssh s364520@data2500.cs.oslomet.no — 146x25
F10 key ==> File Edit Search Buffers Windows System Help
FROM ubuntu:latest

ENV TZ=Europe/Oslo
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

RUN apt-get update && apt-get install -y apache2

COPY index.html /var/www/html/

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Her bygger jeg docker hvor -t opsjonen (tag) legger til navn og . refererer til gjeldende katalog som inneholder Dockerfile

```
group25@os25:~/uke11AD/oppg15$ sudo docker build -t webapache .
[+] Building 0.8s (9/9) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 287B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 28 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 0.8s
=> [1/4] FROM docker.io/library/ubuntu:latest@sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 31B 0.0s
=> CACHED [2/4] RUN ln -snf /usr/share/zoneinfo/Europe/Oslo /etc/localtime && echo Europe/Oslo > /etc/timezone 0.0s
=> CACHED [3/4] RUN apt-get update && apt-get install -y apache2 0.0s
=> CACHED [4/4] COPY index.html /var/www/html/ 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:64f41950eda9e0cf5ae3ebb639e01a2f5526d414e7d6b16ba209c6c36271436b 0.0s
=> => naming to docker.io/library/webapache 0.0s
group25@os25:~/uke11AD/oppg15$
```

Lager index.html

```
group25@os25:~/uke11AD/oppg15$ ls -l
total 12
-rwx----- 1 group25 group25 248 Mar 21 23:04 Dockerfile
-rwx----- 1 group25 group25 248 Mar 21 23:02 Dockerfile~
-rw-rw-r-- 1 group25 group25 38 Mar 21 23:07 index.html
```

Også bygger jeg. Under ser man at etter bygginga så finnes imaget:


```
group25@os25:~/uke11AD/oppg15$ sudo docker image ls
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
webapache_amna      latest         64f41950eda9   13 minutes ago 228MB
```

Går inn i imaget

```
group25@os25:~/uke11AD/oppg15$ sudo docker container run -it webapache_amna bash
root@d721b8f59774:/#
```

Det er flere måter å starte apache på:

```
root@d721b8f59774:/# /etc/init.d/apache2
Usage: apache2 {start|stop|graceful-stop|restart|reload|force-reload}
```

```
root@d721b8f59774:/# /etc/init.d/apache2 start
* Starting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
*
root@d721b8f59774:/#
```

Går ut med control p control q og lister docker:

```
group25@os25:~/uke11AD/oppg15$ sudo docker container ps
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS          NAMES
d721b8f59774   webapache_amna "bash"         2 minutes ago Up 2 minutes   quirky_moore
```

Vi ser at containeren ikke har noe port. Da stopper jeg den

```
group25@os25:~/uke11AD/oppg15$ sudo docker container ps
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS          NAMES
d721b8f59774   webapache_amna "bash"         3 minutes ago Up 3 minutes   quirky_moore
group25@os25:~/uke11AD/oppg15$ sudo docker container stop d
d
group25@os25:~/uke11AD/oppg15$ sudo docker container ps
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS          NAMES
```

Nå prøver vi å kjøre igjen med opsjonen -p som sier at all trafikk skal videresendes til 80:

```
group25@os25:~/uke11AD/oppg15$ sudo docker container run -p 8081:80 -it webapache_amna bash
root@622427fec49f:/#
```

Starter apache igjen:

```
root@622427fec49f:/# ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   4616   3712 pts/0    Ss   22:27   0:00 bash
root          63  0.0  0.0   7052   1608 pts/0    R+   22:28   0:00 ps aux
root@622427fec49f:/# /etc/init.d/apache2
Usage: apache2 {start|stop|graceful-stop|restart|reload|force-reload}
```

```
root@622427fec49f:/# /etc/init.d/apache2 start
* Starting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
*
```

Kan se at jeg har docker image her

```
group25@os25:~/uke11AD/opp15$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
webapache_amna latest    64f41950eda9   25 minutes ago 228MB
```

Runner og bruker curl for å se om trafikken korrigeres riktig:

```
group25@os25:~/uke11AD/opp15$ sudo docker run -d -p 8081:80 webapache_amna
23a37eae46d20bee0852125c64a604b6686ec1ac4e8f342942deb1c818f046ab
group25@os25:~/uke11AD/opp15$ sudo docker run -d -p 8082:80 webapache_amna
3dec70e567e34c6fcd6e01eb9a57a83f57f83e264c2f573f29517c95909baf9
```

```
group25@os25:~/uke11AD/opp15$ curl localhost:8081
Amna hilser fra Dockerfile container!
group25@os25:~/uke11AD/opp15$ curl localhost:8082
Amna hilser fra Dockerfile container!
```

16. UKENS NØTT 4

Skriptet mitt:

```
amnadastgir — group25@os25: ~/uke11AD/opp15 — ssh s364520@data2500.cs.oslomet...
F10 key ==> File Edit Search Buffers Windows System Help
#!/bin/bash

#bygger docker image
docker build -t apache-amna .

for i in {1..9}
do
    docker volume create volume$i
    docker run -d -p 808$i:80 -v volume$i:/var/www/html apache-amna
done
```

Endret rettigheter og kjøring: jeg får feilmelding om at to av portene er i bruk, men det vet jeg, orket ikke fikse det ☺

```
group25@os25:~/uke11AD/oppg15$ sudo ./oppg16.sh
[+] Building 0.9s (9/9) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 283B                                              0.0s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest                0.9s
=> [1/4] FROM docker.io/library/ubuntu:latest@sha256:67211c14fa74f070d27cc59d69a7fa9aef 0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 31B                                                0.0s
=> CACHED [2/4] RUN ln -snf /usr/share/zoneinfo/Europe/Oslo /etc/localtime && echo Euro 0.0s
=> CACHED [3/4] RUN apt-get update && apt-get install -y apache2               0.0s
=> CACHED [4/4] COPY index.html /var/www/html/                                0.0s
=> exporting to image                                                          0.0s
=> => exporting layers                                                         0.0s
=> => writing image sha256:64f41950eda9e0cf5ae3ebb639e01a2f5526d414e7d6b16ba209c6c36271 0.0s
=> => naming to docker.io/library/apache-amna                                0.0s
volume1
7a7b68efbc7feeab40753cea4d84a5c7826517adbd45d163d6c5e6021d209145
docker: Error response from daemon: driver failed programming external connectivity on endpoint
xenodochial_kare (2a0916227198c41a824e87e5acececab776893ff325c4c720c235be747b0bc70): Bind for
0.0.0.0:8081 failed: port is already allocated.
volume2
6a1ed73774c86890878feb36434f957564b2e470149c66e9287f058c6d66e04b
docker: Error response from daemon: driver failed programming external connectivity on endpoint
vibrant_elgamal (96adf40111488f4d1b09816d76e9fc0e735ba4eb3c1fcffe3aa3a4b851a2d641): Bind for 0
.0.0.0:8082 failed: port is already allocated.
volume3
a133a3e917193cc383239758d93f144ec872ab785638291652488fa81d5f1f8a
volume4
0eff1675c8366d67cd1358f57bb706e070fcdf336781857a5d44bbc0a5143ab0
volume5
735fd38ec1e903b9cc462783871b3b3c9f76c8d162acbd96a093c7b65e6f0c33
volume6
0653702aaa94d463d15978f8bb9922c58ae69e43b9c53ed8854dc5b0b99f5922
volume7
503e1f85716721e5376ef5b24afd5260e7bd807d496ee51cabcc880447785368a
volume8
e00985cc1bb8e8d2122fd914f468f5a9bd9aa08ee3ae5f39f972e01322f69393
volume9
25ac52528eaba83689c6a9a3e77c775f71e7f79592f0565cda160cd36e3916c9
-----
```

17. (EFFICODE)

Logger inn i min s-VM

```
group25@os25:~$ ssh -p 5264 s264@intel1.vlab.cs.oslomet.no
The authenticity of host '[intel1.vlab.cs.oslomet.no]:5264 ([10.196.10.11]:5264)' can't be established.
ECDSA key fingerprint is SHA256:vXaCT0GMRZkKe7y4VW8hTZKqxxfQVXdsRxIhVV9sZCY.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[intel1.vlab.cs.oslomet.no]:5264,[10.196.10.11]:5264' (ECDSA) to the list of known hosts.
s264@intel1.vlab.cs.oslomet.no's password:
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.4.0-144-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Mar  6 12:02:03 2023 from 10.47.144.207
```

Finner docker id:

```
s264@os5264:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3da1f0c1b6b6	public.ecr.aws/lts/ubuntu:latest	"tail -f /dev/null"	6 days ago	Up 6 days		suspicious_elgamal

Logger meg inn i docker med exec og finner koden:

```
s264@os5264:~$ sudo docker exec -it 3da1f0c1b6b6 /bin/bash
root@3da1f0c1b6b6:/# cat /etc/*-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
DISTRIB_DESCRIPTION="Ubuntu 22.04.2 LTS"
Pvu5vJS6xE
PRETTY_NAME="Ubuntu 22.04.2 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.2 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
root@3da1f0c1b6b6:/#
```