Løsningsforslag Eksamen våren 2017 Operativsystemer

Oppgave 1 - Prosesser (20%)

a) PID betyr prosess-ID. Hver prosess får en unik ID. Denne ID'en kan brukes til å sende signaler til en prosess, f.eks ved hjelp av kill kommandoen.

b) Et systemkall er et kall til en metode i OS-kjernen. Et program som kjører i user mode kan bare utføre et begrenset antall instruksjoner og aksessere en berenset del av RAM. Hvis det ønsker å gjøre en oppgave som involverer noe av det som ikke kan gjøres fra user mode, må det gjøre et systemkall og be OS-kjernen om å utføre dette for seg.

c) Kommandoen ønsker å liste opp filen /etc/passwd med ls kommandoen. For at dette er mulig må kommandoen ls kunne lese mappe-informasjonen til /etc. Dette får den ikke lov til å gjøre selv, siden man må aksessere en ekstern enhet, så den må kjøre systemkall som open(), readdir() og flere andre, hvoretter operativsystemet returnerer informasjonen tilbake til prosessen.

d) Nei, selvom de underliggende instruksjonene er de samme, vil all kommunikasjon programmet gjør med operativsystemet være forskjellig, inkludert systemkall.

e) Ja, generelt vil et slikt program kunne kjøre, siden det vil måtte kjøres i en JVM(Java Virtual Machine) som gjør at class-filen er plattformuavhengig. Men det forutsetter at JVM'en på Windows 10 har samme hovedversjonsnummer av Java, for class-filer er stort sett ikke kompatible hvis JVM har forskjellig hovedversjonsnummer.

f) Ca 16 minutter. En regneprosess trenger CPU hele tiden og de to prosessene må da bytte på å bruke CPU'en og det tar omtrent dobbelt så lang tid.

g) Fordi OS ikke kjenner den indre logikken i prosessene og må la instruksjonene kjøres sekvensielt. Instruksjonen kan ikke kjøres parallelt, for det kan være at de avhenger av hverandre.

h) Ca 21 minutter. OS vil fordele CPU jevnt mellom de 7, de får da 2/7 deler CPU-tid hver og 2/7 deler av 21 minutter er 6 minutter. Alternativt: det er $7 \cdot 6 = 42$ minutter CPU-tid som trengs for å bli ferdig. Hver CPU bidrar like mye og det tar $42/2 = 21$ minutter.

Oppgave 2 - Linux kommandolinje (20%)

I denne oppgaven skal vi først studere noen Linux-kommandoer som til slutt skal settes sammen til en lang Linux-kommando som viser hvilke fem ord Shakespear brukte oftest i sine samlede verker.

a) Den skriver ut de to første linjene av filen `num.txt`.

b) Opsjonen `n` betyr sorter numerisk (default er alfabetisk) og `r` betyr 'reverse sort' eller å sortere slik at de største tallene kommer øverst. Default er de minste først.

c) `uniq -c test.txt`

d) `sort | uniq -c | sort -rn`

e) `cat */* >> /tmp/alle.txt` Resultatet blir det samme med enkel `>` . Andre altertativer:

```
for file in $(find . -type f); do cat "$file" >> /tmp/alle.txt ; done;
find . -type f -name "*" -exec cat {} >> /tmp/alle.txt \;
```

f) `tr -cs A-Za-z '\n' | tr A-Z a-z` eller
`tr A-Z a-z | tr -cs a-z '\n'`

g) `cat /tmp/alle.txt | tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | head -n 5`

Oppgave 3 - Bash-scripting (15%)

a)

```
#! /bin/bash
fra=$1
til=$2

for fil in *.$fra
do
   if [ -f $fil ]
   then
      name=$(basename $fil .$fra)
      mv $fil $name.$til
      echo "Endrer $fil til $name.$til"
   fi
done
```

Oppgave 4 - Virtualisering (10%)

a)

- **Isolasjon**: Tjenester og programmer kan kjøre på hver sin dedikerte server. Unngår at de forskjellige tjenestene ødelegger for hverandre, gir ryddigere drift og økt sikkerhet: hvis en tjeneste blir hacket, vil det ikke påvirke de andre tjenestene.
- **Ressusrssparing**: Virtuelle maskiner (VMer) som for eksempel bruker lite CPU kan settes på samme fysiske server. VMer kan enkelt flyttes til og fra fysiske servere og man kan dermed spare hardware og strøm.
- **Fleksibilitet**: Kapasiteten kan enkelt skaleres ved å legge til eller fjerne VMer, lastbalansering blir enklere. Elastisitet: Man kan dynamisk tildele CPUer og internminne til VMer. Har en VM blitt ødelagt eller kompromittert kan man enkelt starte opp en ny kopi.
- **Programvare-utvikling**: Man kan raskt teste ut programvare på forskjellige operativsytemer, Window, Linux, Mac, etc. ved å kjøre VMer med en rekke forskjellige OS
- **Skytjenester**: Virtualisering er grunnlaget for fleksible skytjenester. Kunder kan gis egne VMer med et antall CPUer, disk og minne. Disse kundene kan dele fysiske servere, noe som gir store besparelser av hardware

b) Gjeste-OS kjører i user mode og vil ikke fungere som på vanlig hardware om en slik instruksjon droppes. Det kan føre til at OS crasher i verste fall. Disse sensitive instruksjonene ble endret slik at de trap'er til kernel mode når de utføres i user mode, istedet for å ikke gjøre noe.

c) En Unikernel er en VM som bare har en funksjon eller tjeneste(single purpose) og som bare har linket inn de operativsystemmodulene som er nødvendig for å tilby denne tjenesten.

d) Et generelt operativsystem er laget for å kunne kjøre på mange forskjellige typer hardware og har et komplekst grensesnitt mot denne hardwaren. Et OS i en virtuell maskin opererer mot et vesentlig enklere grenesesnitt, hypervisors virtualisering av hardware. Dermed er det mye kode i et generelt OS som er overflødig. I tillegg har hypervisor allerede en form for OS-kjerne som for eksempel sørger for scheduling og virtuelt minne. Et OS for en virtuell maskin kan dermed klare seg uten mye av koden et generelt OS trenger og det er nettopp dette som utnyttes i en Unikernel. Kun det som er nødvendig for at tjenesten skal virke er tatt med.

Oppgave 5 - PowerShell (20%)

a) Det betyr at `ps` er et alias for CommandLet'en Get-Process og at det er denne som utføres når man bruker kommandoen `ps`.

b) Get-Member gir informasjon om alle alias, metoder og variabler(egenskaper/properties) som er definert for ps-objektene som returneres når kommandoen `ps` kjøres.

c)

```
ps | Select-Object id, name, @{name="ProsentWS";expression={100*$_.workingSet64/$_.peakWorkingSet64}}
```

d) Working set er størrelsen på hvor mye RAM prosessen aktivt bruker. `Prosent WS` viser hvor stor del som nå er i bruk i forhold til det prosessen brukte når minnebruken var på sitt høyeste. Dette kan være nyttig informasjon om man ønsker å vite hvor mye minnebruken potensielt kan øke til hvis prosessen bruker opp mot det den historisk sett maksimalt har brukt. Og det kan si noe om hvilken tilstand en prosess er inne i, spesielt om man kjenner detaljer om hva prosessen gjør.

e)

```
foreach ($ps in ps){
        $W += $ps.WorkingSet64
        $P += $ps.PeakWorkingSet64
}
$prosent = 100*$W/$P
"Prosent Workingset av maksimalt: $prosent"
```

Oppgave 6 - Kode og maskinkode (15%)

a)

- C-kode er tekst-kode som er skrevet i C. Denne koden må kompileres til maskinkode for at den ska kunne kjøres.
- assembly-kode er tekst-kode som ligger svært nær maskinkode. En assembly-instruksjon tilsvarer en maskin-instruksjon. Men den må kompileres(eller assembles) til maskinkode.
- maskinkode er binær kode som vanligvis er resultatet av et kompilert høynivåprogram eller assemblykode. Det er instruksjoner i et program som kjøres etterhverandre en for en av CPU'en.

b) Et CPU-register er en minneenheten i CPU'en som brukes direkte i CPU-instruksjoner som ADD, SUB og CMP. De brukes til å lagre data CPU'en jobber med og også selve instruksjonene. CPU-registere er den hurtigste og minste minneenheten i en datamaskin. Det finnes noen titalls slike registere i en CPU. I moderne CPUer er et register vanligvis på 64 bit.

c) Med 32 bit kan man lagre heltall fra 0 til $2^{32} - 1$. (som er 4.294.967.295 om man regner det ut)

d) Først legges verdien 0 i rbx og rax. Deretter kjøres en løkke hvor rbx økes med 1 for hver runde i løkken og avslutter når $rbx = 3$. For hver runde i løkken utføres $rax = rax + rbx$. Dermed blir til slutt $rax = 1 + 2 + 3 = 6$ og denne verdien returneres dermed av metoden.

e) Hvis man endrer $3 til $4 i cmp-linjen vil løkken gå en runde til og $rax = 1 + 2 + 3 + 4 = 10$ returneres i rax. En annen mulighet er å endre $0 til $4 i initialiseringen av rax `mov $0, %rax`, slik at sluttresultatet blir 4 større og dermed 10.