

IS	105
Gruppe 3 NoName	ICA02

Gruppemedlemmer:

Ali Al Musawi
Tor Borgen
Ann Margrethe Ly Pedersen
Brage Fosso
Adrian Lorentzen
Arne Bastian Wiik
Morten Schibbye

Alle kode henvisninger ligger i README.md

<https://github.com/GB-Noname/is105-ica02>

Oppgave 1

Formål: bli kjent med datasystemet sitt.

All data er notert nedenfor i tabellen

Kan man gi et nøyaktig antall på prosesser som kjører? Begrunn.

NEI. Prosesser kommer og går hele tiden, fra oppstart av maskinen og ut ifra hva som blir gjort på datamaskinen til en hver tid.

Hvis de ikke kjører, hvilke tilstander befinner de seg da?

De befinner seg i en "Sleeping" tilstand, det vil si at de venter på et kall fra en annen applikasjon, når de får det kallet så blir de aktive.

Finn ut hvilken prosess i ditt system bruker mest minne. Beskrive denne prosessen kort.

Vi har alle funnet ut at Google Chrome er den prosessen som bruker mest minne. Grunnen kan være at chrome dupliserer alle kjørende instances i hver fane, det betyr at alle tillegg extensions eller scripts som kjører blir duplisert (adblock, privacy badger, lastpass etc.). Det gjør at den bruker veldig mye RAM i forhold til de andre.

Teamarbeid: Oppsummer alle data i en tabell i deres team-besvarelse. Sammenlign deres plattformer og diskuter forskjeller.

	Bastian	Ali	Brage	Morten	Adrian	Ann Margrethe	Tor
Prosseser	142	79	102	157	111	136	95
Prosessortype	Intel Core i3	Intel Core i3-2350M	Intel Celeron CPU B810	Intel Pentium CPU 4405U	Intel Core i5-3230M	Intel Core i5-6267U	Intel Core i5 4210M
Prosesorarkitektur	Intel	Intel	Intel	Intel	Intel	Intel	Intel 8051
Klokkefrekvens	2.3 GHz	2.3GHz	1.6 GHz	2.1GHz	2.6 GHz	2.9 GHz	2.6 GHz
Info om primært minne	4096 MB	4096 MB RAM	4096 MB	4096 MB	4096 MB	8192 MB	4096MB
Cache: L1	128 kB	128 kB	128 kB	128kB	128 kB	128 kB	128 kB
L2	512 kB	512 kB	512 kb	512kB	512 kB	512 kB	512 kB
L3	3 MB	3 MB	2 MB	2MB	3 MB	4 MB	3 MB
Hvilken prosess bruker mest minne	Chrome	Chrome	Google Chrome	Google Chrome	Chrome.exe	Google Chrome	Chrome
CPU-cores på dataen	2	4	2	4	2	4	4
CPU-cores på virtuell server	1	1	1	1	1	1	1
VM Processer	118	118	118	118	117	118	171

Vi ser at det er forskjeller på hvor mange prosesser som kjører på datamaskinene våre. En forklaring på dette kan være at vi hadde ulikt antall programmer kjørende på datamaskinen. Jo flere programmer som kjører, jo flere prosesser må til for å kjøre programmene.

Vi ser at alle har samme prosessorarkitektur av typen Intel. Mens ingen har samme modell.

Klokkefrekvensen varierer noe. Den laveste klokkefrekvensen finner vi på Brage sin datamaskin, som er 1.6 GHz. Den raskeste finner vi Ann-Margrethe sin datamaskin som er på 2.9 GHz. Dette betyr at Ann-Margrethe sin prosessor er raskere enn Brage sin. Klokkefrekvens (Hz) er hvor mange kalkulasjoner prosessoren klarer å utføre per klokkepulss.

Alle på gruppen har samme RAM, bortsett fra Ann-Margrethe som har 8 GB ram, mens resten har 4 GB. Hennes PC kan da kjøre flere programmer som krever mye minne, enn det vi andre kan.

Cachen er lik på level 1 og 2. Mens Ann-Margrethe sin PC skiller seg ut nok en gang hvor hun har 4 MB L3 cache, mens Brage og Morten har 2 MB L3 cache, mens resten av gruppen har 3 MB L3 cache. L1 cache er direkte på CPU-en. Jo større

cache, jo mer har du mulighet til å prosessere flere sykluser. Det er litt likt som RAM, men det som skiller det er at cache ligger enten i kjernen eller i prosessoren, noe som vil hjelpe Pcen å utføre flere prosesser.

Antall CPU-cores varierer på datamaskinen, enten har vi på gruppen 2 eller 4 cores. Kort fortalt vil det si at jo flere cores prosessoren har, jo fortere kan den utføre mer krevende oppgaver på datamaskinen. Dette på grunn av utnyttelse av multi-threading eller "rutiner"/concurrency som kan kjøres parallelt, det er da bedre å kunne kjøre flere prosesser på en CPU med lavere klokkefrekvens om man behøver concurrency. Skal man kun kjøre en thread er det bedre å ha en større klokkefrekvens med færre prosessorer.

Alle på gruppen har tildelt den virtuelle serveren 1 CPU-core. Hvis vi hadde tildelt flere cores, ville dette ha påvirket hastigheten på selve Pcen.

Når det kommer til antall prosessorer på den virtuelle serveren, er det Tor som skiller seg ut. Resten ligger på 117 eller 118 prosesser, mens Tor har 171 prosesser. Dette er på grunn av at han har lagt til mer funksjonalitet på sin server enn det vi andre har gjort (For eksempel, kjører tilleggsprogrammer, tjenester og lignende)

Hvilke komponenter (både fysiske og abstrakte) i deres datasystemer er involvert i oppstart, administrasjon og avslutning av prosesser? Definer komponentene du nevner.

CPU- Central processing unit. Utfører instruksjonene til programmene

RAM- Random access memory- minne som gir tilgang til lagrede data i tilfeldig rekkefølge(Hovedminnet)

Harddisker- lagringsplasser for binært kodet info.

HDD - for lagring eller aksessering av data

Motherboard - Hovedkort som binder alt sammen i PC

I/O enhet - En form for input

Oppgave 2

Formål: begynne å forstå hvordan programmer utføres i et datasystem.

Aktivitet: Kompilere kode for forskjellige plattformer og bytte kode med hverandre. For eksempel, kompiler kode for din MS Windows- eller Mac OS X-maskin på din virtuelle server i skyen. Dere må selv finne ut hvordan dere skal overføre filen fra deres virtuelle server til deres lokale datamaskin (det blir plattform- og applikasjons-avhengig).

Link til video med lyd: <https://www.youtube.com/watch?v=0Y-7mLzbX0Y&feature=youtu.be>

Oppgave 3

Formål: begynne å forstå testing og datatyper.

Når vi forsøker å kjøre filene som de ble lagt ut, så feiler testen. Dette grunnet tallene overgår mulige verdier for int8, 128 er ikke mulig så vi endret til 127 slik at vi kan kjøre programmet. Det samme gjorde vi følgende tester, "FAIL" blir dermed kalkuleringsfeil ved "off by one" for å få feil.

Uint32 returnerer 0 fordi rollover fra max uint32 til 0 når den overstiger max verdi.

Int32 returnerer minus fordi rollover fra max int32 blir minus som neste tall.

Det samme gjelder int64.

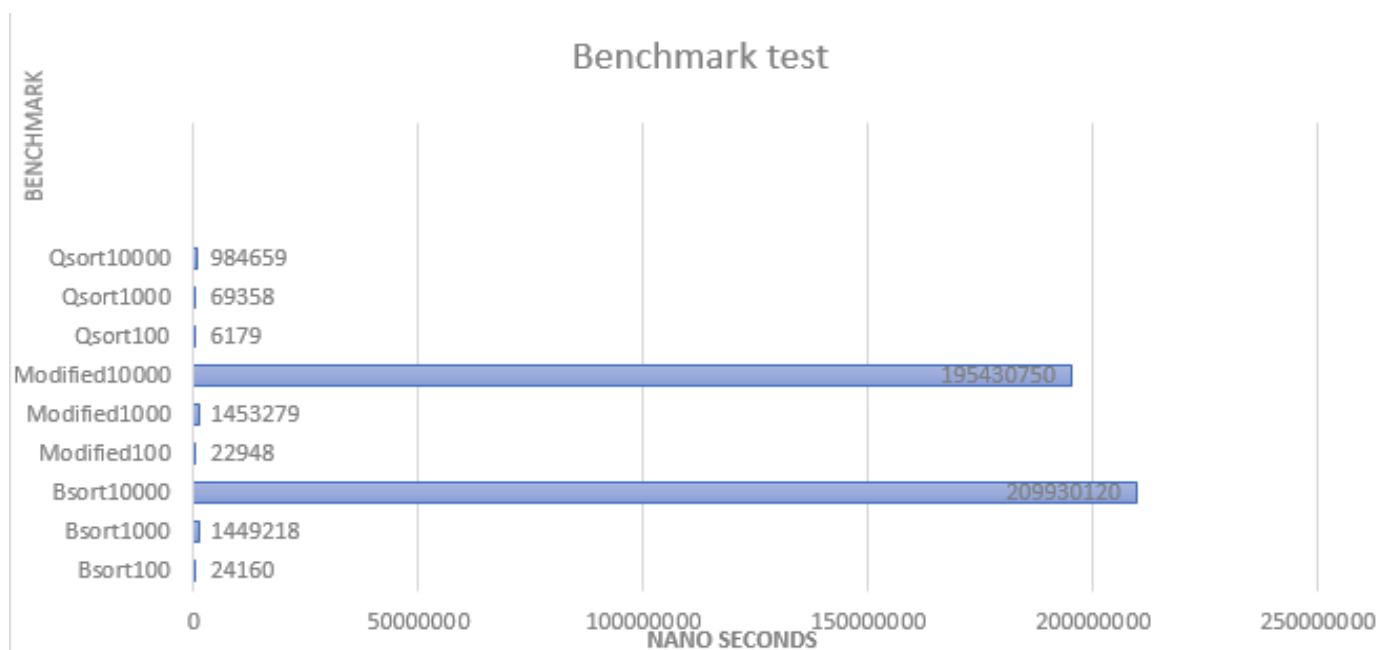
Float64 returnerer feil verdi fordi utregningen med float er vanskelig å teste ut til maks siden float64 har utrolig stor bredde av ulike nøyaktighet.

Vi ser at jo lengre opp i datatype tabellen man kommer jo mer krevende blir testene for større tall.

Det blir også mer krevende å teste grenseverdier og "rollover" fra grenseverdi max til minus.

Oppgave 4

Formål: begynne å forstå algoritmer og utføre "benchmark"-tester på koden.



Hva kan dere si om big-O for alle 3 algoritmene, som dere har testet?

Quick Sort bruker gjennomsnittlig tiden $n \log(n)$ som er den teoretiske grensen QuickSorts algoritme sorterer verdiene i en tabell (Array) gjennom tre trinn.

1. Først velges en tilfeldig verdi i tabellen kalt dreietapp (Pivot)
2. Så deles alle verdiene i tabellen i større og mindre verdier enn Pivot (det tilfeldige tallet)

3. Samme metode gjentaes rekursivt først på de mindre verdiene enn pivot inntil alle er sortert, så på de større verdiene enn originale pivot med samme medtode.

QuickSort er en metode som er raskere enn Bubblesort fordi Bubblesort krever mange flere operasjoner med å flytte et lavt tall nedover i tabellen.

Bubble Sort bruker svært lang tid og må gjennomføre n^2 operasjoner for å kunne sikre sorteringen.

Bubblesort starter med å sammenligne de to laveste verdiene og sortere de etter størrelse. Prosessen gjentaes så med alle andre par i tabellen. Deretter begynner Bubblesort fra starten av med de to første elementene og gjentar prosessen helt til alle verdiene er i riktig rekkefølge.

Bubble Sort Modified bruker marginalt mindre tid. Så i større skala vil det kunne ha en effekt.

Oppgave 5

Formål: begynne å forstå prosessadministrasjon på et platform

Hva kan du si noe om antall prosesser og tråder, som programmet bruker på ditt system?

Windows PS>

5-8 tråder

```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\Tor> ps
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
360	19	9256	21864	0,44	11676	6	ApplicationFrameHost
255	15	2680	9536		2848	0	armsvc
208	15	4640	14680	0,20	7084	6	backgroundTaskHost
175	9	6876	7568	7,88	7332	6	bash
113	5	6216	6036	0,03	14788	6	bash
134	8	3152	6324	0,03	10368	6	boring_main
140	10	1852	1040		2812	0	BtwRSupportService
2035	77	238964	138272	209,88	1084	6	chrome
290	37	92760	42044	4,98	3408	6	chrome
276	35	60244	45224	9,69	7112	6	chrome
382	60	407132	101160	94,78	8756	6	chrome
320	30	54560	32348	1,61	10032	6	chrome
127	11	1976	8212	0,03	10376	6	chrome

Med boring_main kjørende er varierende mellom 5-8 aktive tråder på prosessoren

Hvilken tilstand befinner prosessen seg i?

Til tross for at både golang og bash kjører så sier top at de er i sleeping

Hvordan kan du stoppe prosessen?

CTRL+ C Stopper ikke prosess, men den kansellerer eller dreper den. Teknisk forårsaker det et avbruddssignal som sendes til programmet som forteller det å avbryte hva det gjør og avslutte umiddelbart.

CTRL+Z hvor kommandoen "stopper" en jobb. Igjen er dette gjort med et signal, men denne gangen er det et "stopp" i stedet for et "avbrudd" signal. Det setter programmet på vent og gir kontroll tilbake til bash

Starte samme programmet på den virtuelle serveren og sammenligne måten å få tilgang til prosessinformasjon på og detaljer man får se om prosessen

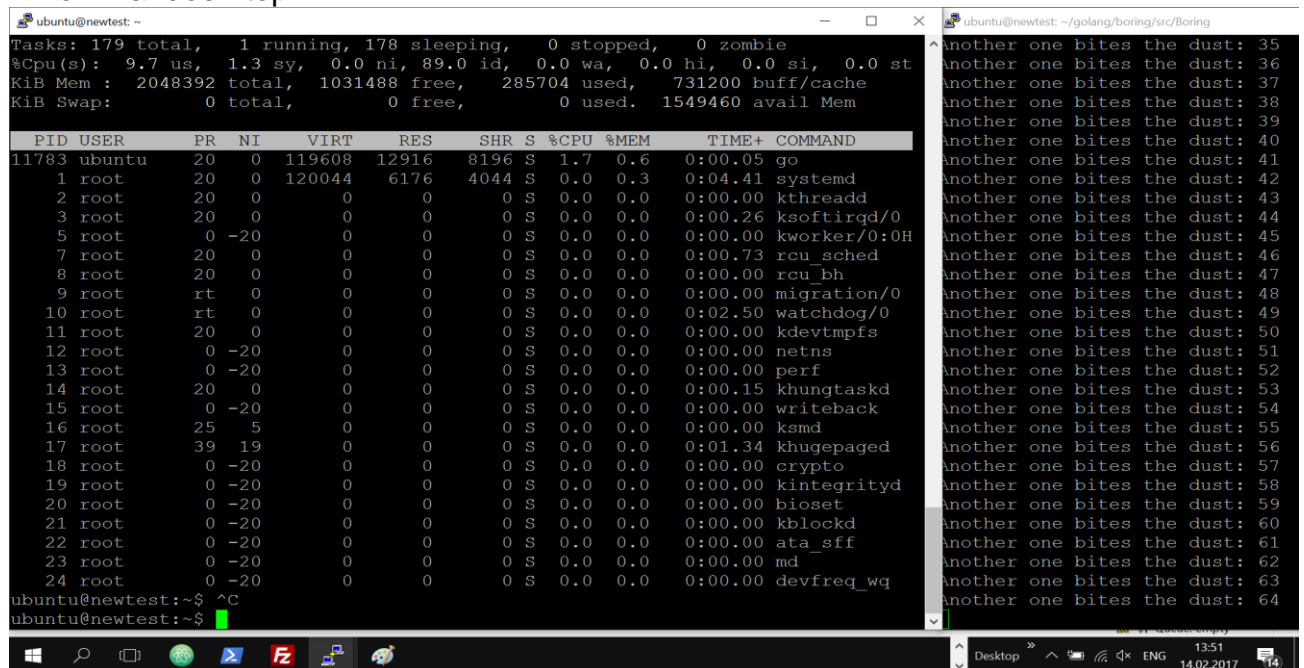
Linux TOP command>

```
PID    USER    PR NI   VIRT  RES   SHR S %CPU %MEM  TIME+
COMMAND
11783 ubuntu  20  0 119608 12916 8196 S 1.7 0.6 0:00.05 go
```

Bilder i bedre kvalitet: <https://github.com/GB-Noname/is105-ica02/tree/master/pics>

BoringLinux

Her kjører vi programmet boring_main.go på linux serveren. Mens programmet kjører vi kommandoen top.



```
Tasks: 179 total, 1 running, 178 sleeping, 0 stopped, 0 zombie
%Cpu(s):  9.7 us,  1.3 sy,  0.0 ni, 89.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 2048392 total, 1031488 free, 285704 used, 731200 buff/cache
KiB Swap:  0 total,  0 free,  0 used. 1549460 avail Mem

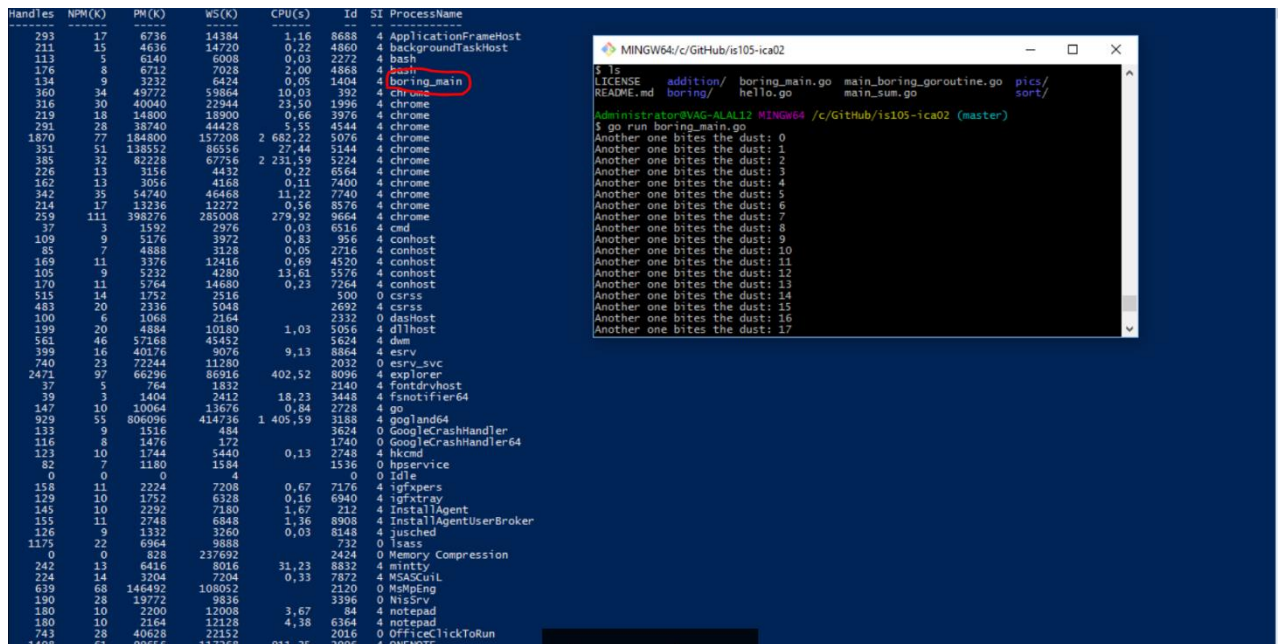
  PID USER    PR  NI   VIRT  RES   SHR S %CPU %MEM    TIME+  COMMAND
11783 ubuntu  20   0 119608 12916 8196 S  1.7  0.6   0:00.05   go
   1 root    20   0 120044  6176 4044 S  0.0  0.3   0:04.41 systemd
   2 root    20   0      0     0     0 S  0.0  0.0   0:00.00 kthreadd
   3 root    20   0      0     0     0 S  0.0  0.0   0:00.26 ksoftirqd/0
   5 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 kworker/0:0H
   7 root    20   0      0     0     0 S  0.0  0.0   0:00.73 rcu_sched
   8 root    20   0      0     0     0 S  0.0  0.0   0:00.00 rcu_bh
   9 root    rt   0      0     0     0 S  0.0  0.0   0:00.00 migration/0
  10 root    rt   0      0     0     0 S  0.0  0.0   0:02.50 watchdog/0
  11 root    20   0      0     0     0 S  0.0  0.0   0:00.00 kdevtmpfs
  12 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 netns
  13 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 perf
  14 root    20   0      0     0     0 S  0.0  0.0   0:00.15 khungtaskd
  15 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 writeback
  16 root    25   5      0     0     0 S  0.0  0.0   0:00.00 ksmd
  17 root    39  19      0     0     0 S  0.0  0.0   0:01.34 khugepaged
  18 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 crypto
  19 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 kintegrityd
  20 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 bioset
  21 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 kblockd
  22 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 ata_sff
  23 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 md
  24 root    0 -20      0     0     0 S  0.0  0.0   0:00.00 devfreq_wq

ubuntu@newtest:~$ ^C
ubuntu@newtest:~$
```

```
Another one bites the dust: 35
Another one bites the dust: 36
Another one bites the dust: 37
Another one bites the dust: 38
Another one bites the dust: 39
Another one bites the dust: 40
Another one bites the dust: 41
Another one bites the dust: 42
Another one bites the dust: 43
Another one bites the dust: 44
Another one bites the dust: 45
Another one bites the dust: 46
Another one bites the dust: 47
Another one bites the dust: 48
Another one bites the dust: 49
Another one bites the dust: 50
Another one bites the dust: 51
Another one bites the dust: 52
Another one bites the dust: 53
Another one bites the dust: 54
Another one bites the dust: 55
Another one bites the dust: 56
Another one bites the dust: 57
Another one bites the dust: 58
Another one bites the dust: 59
Another one bites the dust: 60
Another one bites the dust: 61
Another one bites the dust: 62
Another one bites the dust: 63
Another one bites the dust: 64
```

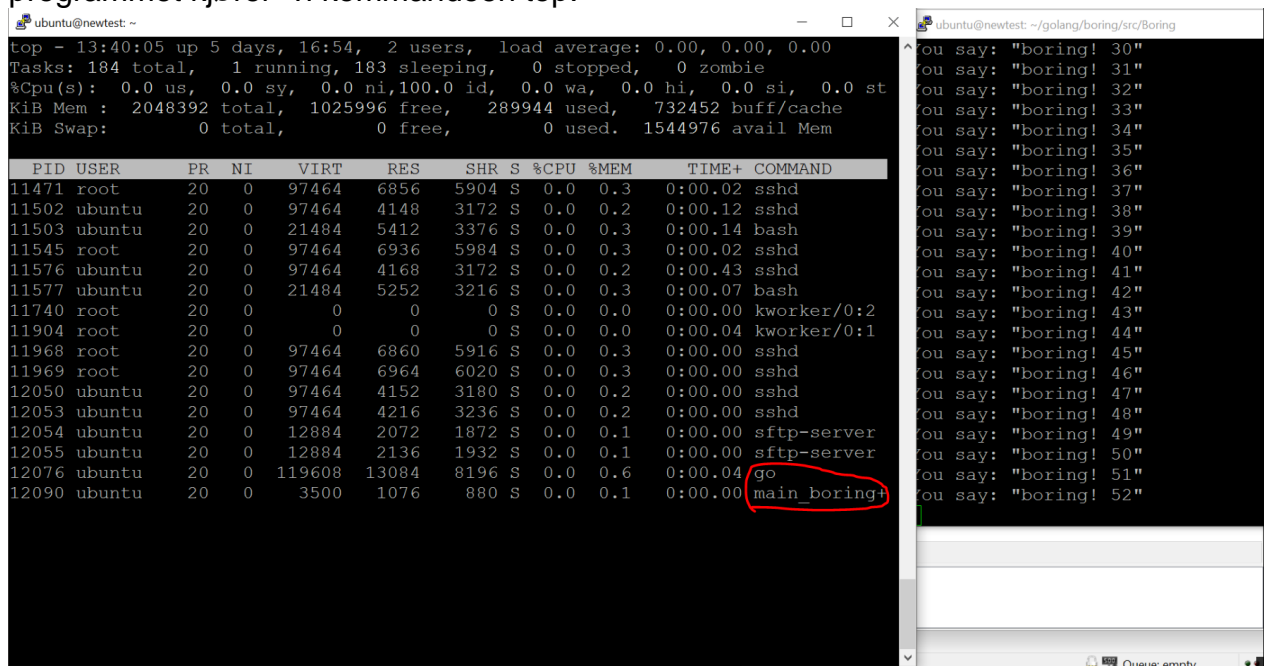
BoringWindows

Her kjører vi boring_main.go i GitBash. Mens programmet kjører utførte vi kommandoen ps i PowerShell.



Boring10Linux

Her kjører vi programmet `main_boring_goroutine.go` på linux serveren. Mens programmet kjører vi kommandoen `top`.



Boring10Windows

Vi bruker git bash for å kjøre `boring_main_goroutine`. Mens programmet kjører, åpner vi PowerShell, og kjører kommandoen `ps`.

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
293	17	6736	14384	1.16	8688	4	ApplicationFrameHost
176	8	6712	7024	2.00	4868	4	bash
113	5	6112	5988	0.03	5304	4	bash
360	34	49772	59876	10.03	392	4	chrome
316	30	40040	22492	23.50	1996	4	chrome
219	18	14800	18808	0.66	3976	4	chrome
291	28	18600	44352	5.55	4544	4	chrome
1871	77	184728	157152	2 682.28	5076	4	chrome
351	51	138600	86452	27.47	5144	4	chrome
385	32	82088	67648	2 231.59	5224	4	chrome
226	13	3156	4428	0.22	6564	4	chrome
162	13	3056	4148	0.13	7400	4	chrome
342	35	54740	48440	11.22	7740	4	chrome
214	17	13236	12256	0.56	8576	4	chrome
259	110	398068	284900	279.92	9664	4	chrome
109	9	5176	3932	0.83	956	4	conhost
85	7	4888	3112	0.05	2716	4	conhost
169	12	4012	13184	1.34	4520	4	conhost
105	9	5232	4252	13.64	5576	4	conhost
535	14	1752	2488		500	0	csrss
475	20	2280	5216		2692	4	csrss
100	6	1068	2164		2332	0	dasHost
165	9	2808	8284	1.06	5056	4	dllHost
560	46	64780	59716		3424	4	dwm
397	15	38128	8416	9.19	8864	4	esrv
740	23	72244	11240		2032	0	esrv_svc
2464	96	66208	86884	404.98	8096	4	explorer
37	5	764	1816		2140	4	Fontdrvhost
39	3	1404	2412	18.30	3448	4	fsnotifier64
144	10	10044	13892	0.44	5340	4	go
916	55	805868	401000	1 406.47	3188	4	googland64
133	9	1516	460		3624	0	GoogleCrashHandler
116	8	1476	172		1740	0	GoogleCrashHandler64
123	10	1744	5428	0.13	2748	4	hkcmd
82	7	1180	1576		1536	0	hpservice
0	0	0	4		0	0	Idle
149	11	2096	7156	0.67	7176	4	igfxpers
127	10	1720	6300	0.16	6940	4	igfxtray
145	10	2292	7180	1.67	212	4	InstallAgent
155	11	2748	6848	1.36	8908	4	InstallAgentUserBroker
126	9	1332	3248	0.03	8148	4	lsasched
1164	22	7016	9848		732	0	lsass
134	9	3228	6452	0.05	2244	4	main_boring_poroutine
0	0	0	828	242.80	2424	0	Memory Compression
242	13	6480	8188	32.02	8832	4	mintty
224	14	3204	7164	0.33	7872	4	MSAScuIL
623	68	146516	104452		2120	0	MsKmpeng
190	29	19772	9800		3396	0	NisSrv
180	10	2200	12000	3.67	84	4	notepad
180	10	2164	12104	4.38	6364	4	notepad
746	28	40680	21464		2016	0	OfficeClickToRun
1501	66	38500	99720	933.23	2996	4	ONENOTE
1153	62	45084	25964	7.33	1556	4	onenoteim
149	10	4460	1576	0.16	3804	4	ONENOTER
662	29	53316	63796	6.42	6676	4	powershell

```
MINGW64/c:/GitHub/is105-ica02
You say: "boring! 91"
You say: "boring! 92"
You say: "boring! 93"
You say: "boring! 94"
You say: "boring! 95"
You say: "boring! 96"
You say: "boring! 97"
You say: "boring! 98"
You say: "boring! 99"
You say: "boring! 100"
You say: "boring! 101"
You say: "boring! 102"
You say: "boring! 103"
You say: "boring! 104"
You say: "boring! 105"
You say: "boring! 106"
You say: "boring! 107"
You say: "boring! 108"
You say: "boring! 109"
You say: "boring! 110"
You say: "boring! 111"
You say: "boring! 112"
You say: "boring! 113"
```