UIA IS-105 UTKAST ICA02
Sist oppdater: 28. Januar 2017

ICA02

Besvarelsen skal evalueres med karakter (A-F). Begrunn alle svar!

Formål

Hva er nyttig å forstå om temaet?
Forståelse av begrepene “program”, “kode”, “prosess” og “tråd”.
Forståelse av kompilert, bygd og lenket kode.
Forståelse av begrepet algoritme og hvorfor er algoritmer viktig i databehandling.

Bestemme akseptabel bevis? Hva som kan telle som bevis at studentene faktisk forstår det som har vært vår intensjon at de skal forstå?
Vise at man kan analysere data fra POSIX-programmer “ps”, “top”, “pstree” (beskrive hva hver linje betyr og endre tilstanden av systemet ved å utføre spesifikk kode og forklare endringene i prosesstabellen)

Vise at man kan skrive flere tester i golang (må ha skrevet og kommentert spesifiserte (3-5) enhetstester og benchmarkstester)
Vise at man kan utføre tester i golang og analysere resultater på en oversiktlig måte.

Vise at man forstår oversettelser av kode til den er platformspesifikk (maskinkode?) og kan utføres.

I denne oppgaven skal dere skrive og utføre en del kode i golang. Eksemplerfiler, som dere kan ta som utgangspunkt finnes i denne repository-en (referert videre som **REP_ICA02**):

https://github.com/uia-worker/is105-ica02

Oppgave 1

Formål: bli kjent med datasystemet sitt.

Hvor mange prosesser som kjører på din datamaskin?
Hvor mange prosesser som kjører på din virtuelle server i nettskyen?
Kan man gi et nøyaktig antall? Begrunn.
Hvor mange av prosessene som “kjører”?
Hvis de ikke kjører, hvilke tilstander befinner de seg da?
Hva er maskinvarespesifikasjon til din datamaskin (noter prosessortype, prosessorarkitektur, klokkefrekvens, informasjon om primært minne, størrelse på cache (både L1, L2 og L3 er ønskelig))?
Hvor mange CPU-“cores” har du tilgjengelig på din maskin? Noter.
Hvor mange CPU-”cores” har du tilgjengelig på din virtuelle server? Noter.
Finn ut hvilken prosess i ditt system bruker mest minne. Beskrive denne prosessen kort.

Teamarbeid: Oppsummer alle data i en tabell i deres team-besvarelsen. Sammenlign deres platformer og diskuter forkjeller.

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

Tips: studer og bruk POSIX-kommando “ps” på Mac OS X og Linux og taskmgr, “ps” e.l. på MS Windows for å besvare spørsmålene. På Linux og Mac OS X finnes det også kommandoer “top” og “pstree”, som kan brukes for å få oversikt over prosess-tilstander på et datasystem.

Oppgave 2

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

Aktivitet: Kompilere kode for forskjellige platformer 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 platform- og applikasjons-avhengig).

Skriv følgende kode i en fil **hello.go**:

```
package main

import "fmt"

func main() {
	fmt.Println("Hello, 世界")

}
```

Tips: "på-tvers"-kompilering i golang gjøres ved å spesifisere operativsystemtype og arkitekur foran "`go build`" kommandoet (for Mac OS X):

```
GOOS=darwin GOARCH=386 go build <filnavn>.go
```

Hvis man ønsker å kompilere kode for MS Windows platformen, bruk GOOS=windows.

Prosessen skal dokumenteres med "screenshots" og tekst eller video og lyd.

Oppgave 3

Formål: begynne å forstå testing og datatyper.

Programmeringsoppgave:

- ta utgangspunkt i pakken **`sum`** i REP_ICA02
- skriv sum_tests_<type> og tilsvarende test-funksjoner TestSum<Type> for følgende typer i golang: uint32, int32, int64, float64 (følg eksemplene i filene)
- lag tester, som produserer "FAIL" og forklar
- informasjon om golang's datatyper finner dere her https://golang.org/ref/spec#Numeric_types
- informasjon om formattering av utskrift til "standard-out" finner dere her ("fmt" pakken) https://golang.org/pkg/fmt/
- tester i golang kan utføres med kommanodet "**`go test`**" og det må finnes filer i den aktuelle mappen med filnavn "`*_test.go`" (for eksempel, `sum_test.go`)
- skriv et program **`main_sum.go`**, basert på pakken "**`sum`**" i REP_ICA02 (som dere har modifisert), som tar inn to parametre fra kommandolinjen og skriver ut summen til "standard-out" strømmen
- prøv ut verdier (for hver av typene uint32, int32, int64, float64), som ikke passerte test-rutinene deres og forklar resultater

- foreslå (og gjerne implementer) en løsning, som gjør bruken av deres pakke “sum” trygg for brukeren

Oppgave 4

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

Programmeringsoppgave:

- ta utgangspunkt i pakken **`algorithms`** i REP_ICA02
- basert på eksempel-funksjon **`Bubble_sort`** i filen **`sorting.go`**, skriv en modifisert bubble-sort funksjon **`benchmarkBSortModified`** (se https://en.wikipedia.org/wiki/Bubble_sort for tips)
- basert på eksempel-funksjon **`benchmarkBSort`** i filen **`sorting_test.go`**, skriv "benchmark"-tester for benchmarkBSortModified funksjonen
- det finnes også en implementasjon av Quicksort algoritme i sorting.go og tilsvarende implementasjon av tester i sorting_test.go; utfør alle benchmark-testene med kommando “go test -bench=.” og presenter resultatene grafisk
- hva kan dere si om big-O for alle 3 algoritmene, som dere har testet? (se http://bigocheatsheet.com/)

Oppgave 5

Formål: begynne å forstå prosessadministrajon på et platform

Programmerings- og overvåkningsoppgaver:

- ta utgangspunkt i pakken **`boring`** i REP_ICA02
- skriv et program i en fil med navn boring_main.go, som kun kaller opp funksjonen **Boring01** fra pakken “**boring**”
- starte programmet på ditt system
- starte et annet terminalvindu og utførsk prosessinformasjon til programmet boring_main.go (bruk “ps”, “top” osv.)
- Hva kan du si noe om antall prosesser og tråder, som programmet bruker på ditt system?
- Hvilken tilstand befinner prosessen seg i?
- Hvordan kan du stoppe prosessen?
- Starte samme programmet på den virtuelle serveren og sammenligne måten å få tilgang til prosessinformasjon på og detaljer man får se om prosessen
- Gjør det samme med funksjonen **Boring10** (kildekoden til main-funksjonen ligger i **main_boring_goroutine.go**)

- Dokumenter med “screenshots” eller video i tillegg til koden i gruppe-repository og beskrivelse i ICA02-rapporten

Referanser

De fleste refferansene er gitt i oppgaver. Dette er tillegslitteratur, som kan være interessant.

Ahmed W. (2014), “Getting started with Go and Test Driven Development”, https://www.binpress.com/tutorial/getting-started-with-go-and-test-driven-development/160, sist sett 2017-02-02

V. Driessen. (2010). A successful Git branching model. Retrieved from ULR http://nvie.com/posts/a-successful-git-branching-model/, sist sett 2017-01

GitHub Help, https://help.github.com/, sist sett 2017-01-24

P. Krill. (2017). “Go tops Java, C, Python for programming language of the year”, http://www.infoworld.com/article/3155924/application-development/go-tops-java-c-python-for-programming-language-of-the-year.html?imm_mid=0ec6d0&cmp=em-prog-na-na-newsltr_20170121, sist sett 2017-01

SLUTT.

JG/2017