DKuug nr. 177

NYT

April-Maj 2015

Recent kernel history

Vers	Date	Csets	Devs	Days
3.13	Jan 19	12,127	1,362	77
3.14	Mar 20	12,311	1,306	70
3.15	Jun 8	13,722	1,492	70
3.16	Aug 3	12,804	1,478	56
3.17	Oct 5	12,354	1,433	63
3.18	Dec 7	11,379	1,458	63
3.19	(February)	11,822*	1,308*	

(*so far)

Dansk Forum for Åbne Systemer

DKUUG - Unix Brugere (Linux/BSD) system administratorer

Community for IT-specialister og IT-interesserede.

- og mere

DKUUG

DKUUG i dag

Nye muligheder

DKuug er en forening og foreninger kommer og går.

Det er vores opgave at fortælle, hvorfor Unix er godt for produktiviteten, for økonomien, at Unix er teknisk overlegent, og hvorfor det gerne må være morsomt at programmere, og det ville være godt, hvis vi også kunne fortælle om hvordan Unix har været med til at få store projekter til at lykkes.

Det kan vi faktisk godt begynde på nu. Unix - rettere Linux - har nemlig også skabt en moderne samarbejdsform.

Aldrig før har der været et software projekt med så meget kode og så mange bidragsydere, og aldrig før har der været så mange ændringer og releases pr. år. Aldrig før har et system kunnet håndtere så mange forskellige hardware-enheder.

Det er store perspektiver. Det viser, at teknisk samarbejde i stor skala må være baseret på en åben samarbejdsform - andre principper end de, der anvendes til projekter som fx. brobyggeri, selv om der også ved bygge- og anlæg er et kreativt element, som skal beskyttes.

Sagen er, at Unix ikke længere kun er Unix.

Først lige et par ord om varemærket Unix som nu - efter en omskiftelig tilværelse - ejes af *The Open Group* som udgiver en Single Unix Specification. Mange leverandører betaler for en compliance test og kan derefter kalde deres OS et *Unix System.* The Open Group forlanger, at Unix altid bruges som adjektiv til en generisk term (fx system). Men Linux - det mest udbredte Unix-system - er ikke compliance testet og behøver det ikke.

Operativsystemer har konvergeret imod Unix siden dengang i 70'erne, da Unix greb om sig med forrygende hast fordi IT-producenter og brugere så det indlysende i standardiserede interfaces til IO, tidshåndtering, proceshåndtering, memory-management og så videre. Poul Henning Kamp skrev for et par år siden:

> *"UNIX viste sig at være det sidste ord i operativsystemer. Fra pSOS til VxWorks, fra VM til zOS fra MS-DOS til Windows, har de resterende operativsystemer allesammen konvergeret imod UNIX. [...]*
>
> *UNIX var godt tænkt, Ken, Dennis og Brian ramte plet og lige fra starten er historien fyldt med førstehåndsberetninger om hvor smitsomt UNIX' tankegods var og hvor underlige ting folk gjorde for at få adgang til UNIX, fra illegale downloads til piratkopiering af Lions berømte kommenterede udgave.*
>
> *Ikke at vejen fra 35 år siden til idag er en lige linie, det er den langt fra, men trods fjendtlige angreb, fra DEC til SCO, og inkompetence, fra USL til OSF, gik det fremad, i store og små ryk og UNIX blev, trods rygter og forudsigelser om det modsatte, hele tiden stærkere."*

(23. juli 2013 på Version2.dk) Poul Henning Kamp sluttede med ordene "Verden vil have Unix". Underforstået: Der er ikke mere så meget brug for interessegrupper som fortæller om Unix, det vi kalder Unix-advocacy med et godt dansk udtryk.

Linux ændrer sig drastisk og i en forrygende fart. Alene størrelsen af kernen kan blive et problem for små embedded ting som i "Internet of Things" IoT - smart belysning og lignende. Linux er kernen i Android, bruges i routere, mediacenters osv. og dér er til gengæld brug for flere og flere faciliteter.

Linux kernen er en *posix-compliant kernel.* Og Posix er som bekendt en formaliseret standard på basis af Unix og BSD.

Kan Posix standarden eller Single Unix hjælpe med at holde sammen på videreudvikling af Linux? Eller er Linux kommet så langt væk fra den teknik, de maskinmæssige muligheder, hvorpå Unix opstod?

Redaktionen har samlet nogle observationer fra de konferencer, som blev afholdt af The Linux Foundation i det forløbne år i artiklen om "Linux Topmøde". Mange af de nye API'er står ikke i standard-dokumenterne.

Linux er det største Open Source projekt - nogensinde - og derfor er det bemærkelsesværdigt at udviklingen går hurtigt, er stabil (så stabil som software kan være).

Det betyder, at Open Source ledelsesformen har vist at den er bedre end andre arbejdsformer. DKuug bør følge med på det område og må gøre opmærksom på fordelene ved Open Source samarbejde.

Det samarbejde findes i mange afskygninger. Det mindste, man kan gøre, er at kortlægge arbejdsformerne, der hænger sammen med rationalet bag de valgte løsningsmodeller.

Dette nummer ...

I dette nummer har vi en artikel om et **micro-python** board på størrelse med en Lego-duplo klods. Der er mange ting, den kan bruges til, og den slags tiltag er en modvægt til fremmedgørelsen overfor digitale enheder.

En af de nyskabelser, der tales meget om for tiden, er Apple Watch, som bl.a. kan monitorere sundhed, og som med en bevægelses-sensor kan registrere voldsomme bevægelser, som fx. hvis en person falder omkuld enten det nu er på grund af en ulykke eller en sygdom. Afhængig af andre omstændigheder tilkaldes hjælp eller der alarmeres. Sådanne hjælpmidler er allerede udviklet, men "for en gør det selv" indstillet person kunne udvikling af nye features ske på basis af fx. sådan en micropython.

David Askirk arbejder som kursusinstruktør og har givet os nogle små udfordringer i dette nummer. Koden til en server skrevet i Lisp kan findes på www.dkuug.dk og så kan man experimentere med den og se, om man kan bruge den som et sikkerhedshul. Den øvelse hedder ***Capture The Flag*** (CTF) og man kan rundt om finde events, hvor man dyrker den sport. Det er ikke sikkert at alle funktioner i Lisp-serveren fungerer, men vi overlader til læseren at finde ud af hvilke og hvorfor.

Som sædvanlig har vi også lidt kommando-linie gymnastik på næstsidste side.

Der var denne gang ikke plads til en sikkerhedsartikel.

Næste nummer: Det har været fremme, at DNyt burde vise nogle gode features i nyeste gadgets, men det må blive i næste nummer. Det vil være rart med flere konkrete forslag.

Donald Axel

DKuug-NYT er medlemsblad for DKuug, foreningen for Åbne Systemer og Internet
Nr. 177 - April-Maj 2014

Udgiver:
DKUUG
Fruebjergvej 3
2100 København Ø
Tlf. 39 17 99 44
email: blad@dkuug.dk
Redaktion:
Donald Axel (ansvarshavende)

Forsidecredits:
redaktionen og Jonathan Corbet

Design og layout:
DKUUG/Donald Axel med LibreOffice
Annoncer:
pr@dkuug.dk
Tryk:
Lasertryk i Aarhus
Oplag:
300 eksemplarer

Artikler og inlæg i DKUUG-Nyt er ikke nødvendigvis i overensstemmelse med redaktionens eller DKUUGs bestyrelses synspunkter.

Eftertryk i uddrag med kildeangivelse er tilladt.

Deadline for nr. 178: ons. 27. Maj 2015

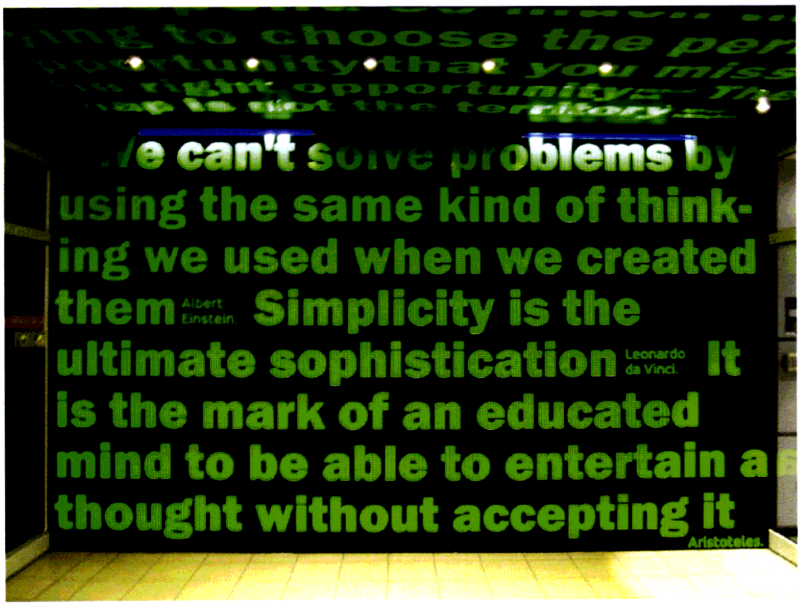
Medlem af Dansk Fagpresse
DKUUG-Nyt
ISSN-1395-1440

Vores møder og **foredrag** holdes - med mindre andet udtrykkeligt angives - på vores adresse:

DKUUG
SYMBION
Fruebjergvej 3
2100 København Ø

Hvis man kommer lidt før, er der tid til en snak på kontoret. DKUUG bor i en virksomhedsfarm, Symbion, hvor der er åbne døre indtil kl.18 eller 19 (afhængig af mødetidspunkt). Efter den tid har vi på foredragsaftener en vagt ved døren.

INDHOLD:



Arrangementer:

Dato ikke fastsat: Kenneth Geisshirt foredrag m.titlen:

Is the database a solved problem? Surveying mobile database solutions .

A decade ago, the database was assumed to be a solved problem. Relational databases (PostgreSQL, MySQL, SQLite to name a few) were dominating the database market and hierarchical databases (LDAP, DNS) where regarded as niche solutions. The NoSQL revolution surely changed the concept of what a database can be. At the same time, the popularity of mobile devices exploded. This talk will dive into how data structures are persisted and queried on mobile devices today, and try to revive the old question: is the database really a solved problem?

Kennet Geisshirt vil fortælle om en tur til Lego World Festival i St. Louis (US)

Desuden vil der komme et foredrag om "*Mobile Databases*" Andre arrangementer vil blive annonceret på web og via mail.

Onsdag : møde på kontoret fra kl.18 ca. - somme tider 17 eller før.

Gør-det-selv foredrag:

Vores kontor i Symbion giver mulighed for hands-on workshops, både dag og aften. Skriv til pr@dkuug.dk eller til bestyrelsen i DKUUG, bestyr@dkuug.dk, og hør om lokalet er ledigt den dag du vil arrangere et møde. Der er hurtig internetforbindelse, både wired og wireless. Er der større tilmelding, kan vi leje mødelokaler i Symbions mødested. Spørg os!

Blandede programmeringsbolsjer

Vi udforsker uendeligt store tal og andre spidsfindigheder i implementeringen af programmeringssprog - med hands-on

Af David Askirk

I denne artikel ser vi på de ubetrådte hjørner i nogle af de kendte programmeringssprog - træd varsomt! - Der lidt om php arrays, lidt om uendelighed og en write-up af en service fra Prosas CTF konkurrence 2012, nemlig en service skrevet i lisp.

PHP

Arrays i php er associative. Dette er vigtigt at huske på, ellers kan man godt falde i nogle fælder.

Vi tager udgangspunkt i følgende stump kode: (https://gist.github.com/davsebamse/b0e73cf1b84c55426f92):

```
<?php

$arr = array(
    1 => "hej",
    2 => "med",
    0 => "dig"
);

print array_values($arr)[0]."\n";

print $arr[0]."\n";

?>
```

Her bliver der udskrevet to forskellige værdier. Som en lille opgave kan der lige gættes på hvad der udskrives ***inden der læses videre.***

Har du gættet? Godt. Det rigtige svar er: "hej" og "dig". Et array i php fungere ikke som et array i andre sprog. Det er et associative array, og fungerer mere som et hash end et array. I php 5.4 er der en funktion, som hedder ***array_values()***, der finder værdierne i et array og returnerer dem. Dette giver svaret *"hej"*, mens det som vi ville antage ville give os den første værdi i array, nemlig $arr[0] giver *"dig"* som er den sidste værdi i arrayet. En vigtig pointe, der viser at det altid er vigtigt at vide hvad man beder om, samt hvad det er man beder om at få en værdi.

Du kan køre eksemplet med php <filnavn>- hvis du bruger Debian eller derivat, er det **`php5 <filnavn>`**.

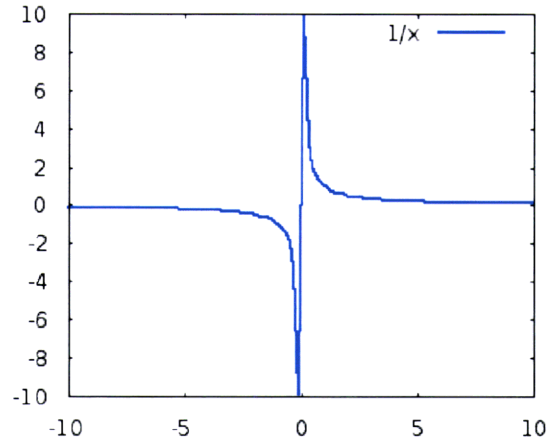
Uendelighed

Fra matematikken kendes begrebet uendelighed eller uendelig.

Hvordan er uendelig defineret? Hvis man kigger på funktionen

```
f = 1/x
```

er den interresant at kigge på når man nærmer sig 0.



Det spændende sted er ved 0. Godt nok ser det ud som om at funktionen er defineret i 0, men det er den ikke. I matematikken er det defineret at 1/0 giver uendelig.

Hvis man kigger på grafen kan man se at den går både mod plus og minus uendelig. Faktisk er det sådan at hvis man tager formelen:

$$\lim_{x \to 0+} \frac{1}{x} \Rightarrow \infty$$

Altså vi nærmer os 1/x fra plus siden så giver det uendelig, hvorimod hvis man nærmer hos fra den negative side, bliver det negativ uendelig.

$$\lim_{x \to 0-} \frac{1}{x} \Rightarrow -\infty$$

Uendelighed er implementeret i nogle programmeringssprog:

Javascript:

```
> 1/0
Infinity
```

.Net (F#):

```
> 1.0/0.0;;
val it : float = infinity
```

Ruby:

```
2.2.0 :001 > 1/0.0
 => Infinity
```

Nu er det lige F#, der er som et eksempel på et **.Net** sprog, men det gælder for alle sprog der kører på **.Net** at der er uendelig, dog kun defineret på double. **Ruby** kræver også noget, nemlig at det skal være en float, hvorimod **javascript** er ligeglad med om man skriver det som en float eller ej.

Når man først står med uendelig og kan regne med den er der altid en sjov beregning at lave:

Uendelig - Uendelig

Dette er ikke defineret i matematikken.

Uendelig + 0 = Uendelig => Uendelig - Uendelig = 0, men samtidig gælder det at

Uendelig + 5 = Uendelig => Uendelig - Uendelig = 5.

Man kan ikke have at den samme beregning giver to forskellige resultater, og derfor er det ikke defineret i matematikken, hvordan man subtraherer og dividerer uendelig med sig selv.

Det man kan se, at de sprog hvor man støder på uendelig, overholder at man ikke kan foretage beregninger med uendelig, så de returnerer NaN (Not a Number).

Her eksempel i F#:

```
> let a = 1.0/0.0;;
val a : float = infinity

> a - a;;
val it : float = nan
```

Det, der så er spændende er jo hvor stort tal kan man få, førend det for stort?

I et sprog som F# er det det defineret i Double.MaxValue.

```
> let a = System.Double.MaxValue;;
val a : float = 1.797693135e+308
```

Nu bliver det spændende:

```
> a = (a-1.0);;
val it : bool = true
```

Her sammenligner vi **a**, som er det største tal en double kan indeholde i F#, og **a - 1.0** og så siger systemet til os at de to tal er ens! Det er mærkeligt! Dette skyldes at flydende tal er er lidt upræcist implementeret, også i .Net. Læs mere om flydende tal og deres implementering her http://en.wikipedia.org/wiki/IEEE_floating_point

Javascript

I **javascript** er det største tal defineret i ECMA standarden (http://ecma262-5.com/ELS5_HTML.htm#Section_8.5) Her kan det ses at det største tal i javascript er 18437736874454810627, men i Chrome vil den ikke gå over 18437736874454810000. Når man kommer derop af sker der sjove ting i chrome.

Ved tallet 2^53-1 sker dette:

```
a     = 9007199254740991
a + 1 = 9007199254740992
a + 2 = 9007199254740992
a + 3 = 9007199254740994
a + 4 = 9007199254740996
a + 5 = 9007199254740996
```

Læg mærke til at den ikke regner rigtigt!!

Hvis man bevæger sig op omkring den maks værdi, som chrome har, sker der også sjove ting.

```
a          = 18437736874454810000-1        = 18437736874454810000
a          = 18437736874454810000-100      = 18437736874454810000
a          = 18437736874454810000-1000000  = 18437736874453811000
a + 1      = 18437736874453811000
a + 2      = 18437736874453811000
a + 3      = 18437736874453811000
a + 1000   = 18437736874453811000
```

Så selvom det ikke er uendeligt, regner browseren alligevel ikke rigtigt! Dette er igen på grund af flydende tal. I Javascript findes der ikke heltal, men kun en tal type kaldet number. Denne type dækker både heltal og flydende tal.

Writeup

Hvert år (næsten) holder prosa en CTF Capture The Flag, som er en IT-sikkerhedskonkurence.

En capture the flag konkurence går ud på at beskytte sin egen server, mens man angriber modstandernes.

I 2012 havde jeg lavet en service i **lisp**.

Se vedlagte kode for den samlede kode. Koden virker dog kun i SBCL (Steel Bank Common Lisp) som findes i din yndlings pakke manager (i det mindste findes den i Debian!)

Servicen er en meget simpel service, hvor man kan gemme noget under en nøgle og hente det frem igen.

Udover funktionalitet til at hente data frem og gemme data, var der en ekstra funktion til at vælge data ud på en anden måde. Ideen med denne var at det skulle være muligt at hente data ud på andre måder end via nøgle.

Sårbarheden ligger netop i denne funktion.

Her er linjen:

```
(eval (read-from-string (subseq line 7))))
```

Læg mærke til det **`eval`** der står i linjen. Dette gør at folk kan køre det lisp kode de ønsker og derved kan overtage maskinen, eller læse alt data ud på en gang.

Lisp er ikke det eneste sprog der har eval, så husk altid at være opmærksom når man støder på eval.

Selve koden:

```
(require :sb-bsd-sockets)

(defpackage kv-server
  (:use :cl :sb-bsd-sockets))

(in-package kv-server)

(defvar *port* 7000)

(defvar *data* nil)

(defun get-data ()
       *data*)

(defun split (string)
       (let ((space-position (position #\: string)))
              (list
                     (subseq string 0 space-position)
                     (subseq string (+ space-position 1)))))

(defun find-data (key)
    (find-if
           #'(lambda (X)
                        (equalp (first X) key))
                     (get-data)))

(defun save-data (line)
       (setf *data* (append *data* (list (split (subseq line 5 ))))))

(defun load-data (line)
       (find-data (subseq line 5)))

(defun custom-selector (line)
       (eval (read-from-string (subseq line 7))))

(defun make-echoer (stream id disconnector)
  (lambda (_)
    (declare (ignore _))
    (handler-case
        (let ((line (read-line stream)))
          (setf line (subseq line 0 (1- (length line))))
          (cond ((string= line "quit")
```

```
                (funcall disconnector stream))
                        ((and (not (null (search "save" line))) (= 0 (search "save"
line)))
                         (save-data line)
                         (funcall disconnector stream))
                        ((and (not (null (search "load" line))) (= 0 (search "load"
line)))
                         (format stream "~a~%" (load-data line))
                         (force-output stream)
                         (funcall disconnector stream))
                        ((and (not (null (search "custom" line))) (= 0 (search
"custom" line)))
                         (format stream "~a~%" (custom-selector line))
                         (force-output stream)
                         (funcall disconnector stream))
               (t
                (format t "~a: ~a~%" id line)
                ;;(format stream "~a: ~a~%" id line)
                ;;(force-output stream)
                        (funcall disconnector stream))))
      (error (e)
             (progn
                    (print "ERROR START")
                    (print e)
             (funcall disconnector stream)
                    (print "ERRor end"))))))

(defun make-disconnector (socket id)
  (lambda (client-stream)
    (let ((fd (socket-file-descriptor socket)))
      (format t "~a: closing~%" id)
      (sb-impl::invalidate-descriptor fd)
        (sb-impl::ignore-interrupt sb-unix:sigpipe)
        (ignore-errors (socket-close socket))
        (ignore-errors (close client-stream))
        )))

(defun serve (socket id)
  (let ((stream (socket-make-stream socket :output t :input t))
        (fd (socket-file-descriptor socket)))
    (sb-impl::add-fd-handler fd
                             :input
                             (make-echoer stream
                                          id
                                          (make-disconnector socket id)))))

(defun kv-server (&optional (port *port*))
  (let ((socket (make-instance 'inet-socket :type :stream :protocol :tcp))
        (counter 0))
    (socket-bind socket #(0 0 0 0) port)
    (socket-listen socket 5)
    (sb-impl::add-fd-handler (socket-file-descriptor socket)
                             :input
                             (lambda (_)
                               (declare (ignore _))
                               (incf counter)
                               (format t "Accepted client ~A~%" counter)
                               (serve (socket-accept socket) counter)))))

#+sb-thread
(sb-thread:make-thread (lambda ()
                         (kv-server)
                         (loop
                            (sb-impl::serve-all-events))))
#-sb-thread
(kv-server)
```

Forklaringer til service implementeret i LISP:

Installation, kørsel og hints til "Capture"

Steel Bank Common Lisp er en afledning af Carnegie Mellon University Common Lisp (CMUCL). Det var et venskabeligt fork i 1999. Forskellen bestod bl.a. i at CMUCL kræver en binær CMUCL for at man kan bootstrappe en ny version, dvs. kompilere fra source, og få en ny version til at fungere, mens SBCL kan bruge en hvilken som helst ANSI compliant Lisp-binary for at komme op at køre. (Til sammenligning kan Gnu C compileren også bootstrappes med en ældre ISO 90 C-compiler - og de ældre Gnu C kunne bootstrappes med en old-gammel C-compiler.)

Det meste af SBCL er struktureret som CMUCL, men der er tilstrækkeligt mange forskelle til at det ville være forvirrende at bruge samme navn+fork-betegnelse. Derfor er systemet navngivet efter Steel Bank efter den industrigren, som gjorde at Andrew Carnegie og Andrew Mellon til mangemillionærer.

Steel Bank Common Lisp hedder sbcl i Debian pakkesystemet, installation som sædvanligt (enten med softwaremanager eller fra kommandolinien:)

```
# apt-get install sbcl sbcl-doc
```

Eller gå til **http://www.sbcl.org/** og hent source - compile, installer. Det er absolut ikke garanteret at source kompilerer uden problemer eftersom den bruger thread-libraries og de er skiftet lidt gennem de seneste år (især header filer mv), så det nemmeste er absolut at finde en Debian eller Fedora pakke.

Programmet kan kopieres fra DKuug.dk (http://www.dkuug.dk/david-askirks-ctf-program-til-download-dnyt-177/ (eller se evt. på annoncering af blad177) eller det kan downloades fra Google-drive:

https://drive.google.com/file/d/0BxtWx2h12yVPRWF0VE45YWMtLWc/view?usp=sharing

Gem lisp-programmet som **davservice.lisp**

Kommentarer til programmet:

Når man ikke arbejder dagligt med Lisp, er det nødvendigt at vide noget om sprogets opbygning - vi havde artikler om Lisp i blad166, men her er de vigtigste ting. Programmet får sit eget namespace gennem **`(defpackage [...])`** statement, (en lisper ville sikkert sige at det er en liste, men lad det nu ligge!) Vi kan bruge simple navne til variable uden frygt og bæven.

Det fremgår af allerførste linie, at et modul med BSD sockets (d.v.s. Internet library) skal være til rådighed. Desuden er der nogle globale variable, *port* og *data* - de har scope i vores egen pakke (skabt med *defpackage*). Identifiers kan bruge bindestreg (som også er minus-tegnet).

Derefter defineres en række funktioner: split(string), find-data(key), save-data(line) osv.

Den næst-nederste linie med hash-

```
#+sb-thread
(sb-thread:make-thread (lambda ()
    (kv-server)[osv...]
#-sb-thread
(kv-server)
```

betyder at hvis vi har sb-threads, så skal der startes en letvægtstråd her med argumentet (kv-server) osv. ellers - hvis det inkludering af sb-thread ikke opfylder alle krav - skal den sidste linie (kv-server) køres.

Programmet begynder eksekvering, når det når frem til en linie, som ikke er definition af pakke, variabel eller funktion. Det er kun linierne efter #+sb-thread, som opfylder det krav. Dér bliver kv-serveren startet, og ved hjælp af diverse sort magi åbner den en socket - lytter - og hvis den får en forbindelse (som når vi bruger telnet til port 7000) så skrives **Accepted client ~A~%.**

Programmet startes sådan:

```
# sbcl --load davservice.lisp
```

i et andet xterm vindue prøves om port 7000 er åben

```
# nmap localhost
Not shown: 995 closed ports
PORT     STATE SERVICE
53/tcp   open  domain
111/tcp  open  rpcbind
631/tcp  open  ipp
6000/tcp open  X11
7000/tcp open  afs3-fileserver

Nmap done: 1 IP address (1 host up)
scanned in 0.10 seconds
```

Man bemærker at port 7000 normalt hører til Andrew Filesystem (AFS) - et alternativ til Net File System (NFS)

Man kan stoppe programmet (men vent lige lidt med det!) med kommandoen:

```
# killall -9 sbcl
```

eller - lidt pænere - find PID med ps kommandoen og slå sbcl ned med **`kill -9 <pid>`**

Linux (og andre Unix) sørger for at lukke alting pænt ned (men hvis der var åbne filer og data, som skulle skrives, bliver de naturligvis ikke skrevet).

Fra den anden Xterm køres:

```
# telnet localhost 7000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
save bogstavA
Connection closed by foreign host.
```

etc.

Prøv også kommandoerne *save* og *load.*

```
sat2:/wb5/arb177 #telnet localhost 7000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
save Hallo1
Connection closed by foreign host.
```

Nøglen til at ramme *eval()* statementet, og dermed at få erobret flaget *(Capture the flag),* er at bruge kommandoen *custom (...)*

Prøv fx. **`custom (get-data).`**

■

Topmøde om Linux

En tilbagevendende begivenhed, hvor The Linux Foundation samler udviklere til foredrag og diskussion om, hvad der sker indenfor udvikling og anvendelse af Linux

Af Donald Axel

I Santa Rosa, en smuk by lidt nord for San Francisco-området, afholdt The Linux Foundation den 18. - 20. februar en konference under overskriften **Collaboration Summit**. Der var foredrag om store og større projekter, som bruger Open Source, og en af de mest spændende foredrag var ***Rapport om Linux Kernen*** af Jonathan Corbet. Han fortalte om arbejdet med at holde styr på udviklere, kvaliteten af kildetekst til forbedringer og rettelser, sikkerhedsproblematikker og godkendelsesprocedure, testprocedure, og release politik. Foredraget kan ses og høres i en rimelig lyd og billedkvalitet på Youtube, og slides kan hentes fra TheLinuxFoundation.org websitet, men vi har valgt at plukke fra slides og foredrag, fordi vi mener at den samarbejdsprocedure, som Corbet beskriver, er central for dette og for alle fremtidige projekter, projekter, som vi herhjemme har set give problemer gang på gang, når de udvikles i kommercielle miljøer ud fra princippet at den der betaler, også bestemmer musikken.

Om foredragsholderen

Hvert år udgiver Jonathan Corbet - Jon for kortheds skyld - sammen med andre programmører og forfattere *Hvem skriver Linux? (Who Writes Linux)* - og holder foredrag om videreudvikling af Linux kernen. Corbet er også en af de mest produktive bidragsydere til kerneudviklingen. Han begyndte allerede som førsteårsstuderende at lappe på memory management på en VAX som kørte BSD. Kode, som forlængst er glemt - heldigvis! siger Corbet selv.

Gennem 80'erne kom Corbet i kontakt med både kerne-programmering og netværksprogrammering. Han har blandt andet arbejdet med netfilter (firewall) på BSD og Sun i slutfirserne. I begyndelsen af 90'erne kom de første spæde Linux-kerner på nettet, som bekendt. De kunne være på en enkelt floppy diskette.

I 1997 (Linux-1.3 eller deromkring) så det ud som om Linus' spøgefulde mål om verdensherredømme så ud til at være tættere på sandheden end man ellers skulle tro, og derfor valgte Corbet at forlade sin fast-løns tilværelse. Jobbet var ved at blive for "pointy-haired", et udtryk fra Dilbert tegneserien, navnet på den dumme og uforstående boss.

Corbet har siden dengang tjent til livets ophold ved at udgive LWN, Linux World News, som blev betalings-website, - Open Source er som bekendt ikke gratis, om så også Linus Torvalds spøger med at sige at Linux er "free as in free beer". Men huslejen skal jo komme et eller andet sted fra.

Hvem skriver Linux?

Allerførst en opsummering af kerneversioner i 2014: der kom version 3.13 til 3.18 og man ser at der til hver release har været ca. 1300 bidrags-ydere - til den sidste 1458.

Der er så mange ændringer, at Corbet (og andre) har fokus på om distro'erne kan følge med.

Den vigtigste parameter for brug af Linux-kernen var allerede fra starten at Linux havde de nyeste faciliteter og kunne håndtere nye enheder. Hvis nu distributionerne frøs kerneversion - d.v.s. holdt fast på den version, som de har brugt til deres distribution - og hvis denne distribution skal være stabil og skal kunne bruges i længere tid, (som fx. Red Hat Enterprise har gjort) så har firmaerne ofte taget de vigtigste rettelser og back-portet dem, d.v.s. ændret den den gamle kerne med de nyeste fejlrettelser - men ikke med skift til ny kerne.

Release-processen blev løseligt fastlagt i 2005. Målet var at få nye features ind i kernen så hurtigt som muligt og ud til brugerne (herunder mobil-telefon producenter) så hurtigt som muligt. Som en følge af det har de fleste distributør-kerner forholdsvis få leverandør-specifikke patches, det giver højere kvalitet og færre forskelle mellem Linux-kerner.

Der stræbes efter en større stabil release hver 2-3 måned. Efter hver større release har kerne-udviklernes "stabilitets-team", som for tiden ledes af Greg Kroah-Hartman, en kort periode med vedligehold, d.v.s. fejlretning, kun vigtigste fixes, så snart de er færdige og klar til udgivelse. Denne ***stabititets-proces*** sikrer at vigtige rettelser kommer til distributørerne samt at de er med i fremtidige releases af kernen.

Man opnår derved, at distributørerne ikke har behov for selv at rette eventuelle fejl eller uhensigtsmæssigheder i kernen og drivere, og derved får man et mere stabilt produkt.

Målsætningen for release-periode er 8-12 uger (2-3 måneder, opnået ved konsensus) og det passer akkurat med at man kan afprøve og finde problemer med nye kerner inden de udsendes.

Who is Doing the Work

The number of different developers who are doing Linux kernel development and the identifiable companies who are sponsoring this work have been increasing over the different kernel versions, as can be seen in the following table.

Kernel Release	Developers	Companies
3.11	1,266	225
3.12	1,332	244
3.13	1,361	228
3.14	1,446	240
3.15	1,492	237
3.16	1,477	234
3.17	1,433	241
3.18	1,458	239

These numbers show a continuation of the steady increase in the number of developers contributing to each kernel release—we have nearly 200 more developers participating in each development cycle at the end of the study period than the beginning.

Since the beginning of the Git era (the 2.6.11 release in 2005), a total of 11,695 developers have contributed to the Linux kernel; those developers worked for a minimum of 1,230 companies. Interestingly, the number of companies supporting work on the kernel appears to be declining slowly, suggesting that developers are consolidating under a (slightly) smaller number of employers.

Bidragsydere

Man skulle tro, at Linus Torvalds var blandt top-bidragsydere af tilføjelser og rettelser, kaldet forandring-sæt, change-sets (også kaldet patches, lapper - se patch(1)) til Linux-kernen, men sådan er det ikke. Det meste af hans og andre top-udvikleres tid går med at gennemse patches. Allerede da Linus i 1991 spurgte på Internettet om der var interesserede, som ville bidrage til hans første, primitive 386-kerne, valgte han en Open Source model for udvikling.

Corbet og hans medskribenter har samlet noget statistik om hvor mange ændringer der er kommet i løbet af 2014:

konkurrerer på andre parametre - og brugerne er glade for stabile systemer.

Antallet af bidrag fra ubetalte udviklere (private/hjemmegående) er faldet en anelse. Man kan kun håbe, at det er tegn på at firmaerne, som bruger Linux, indser betydningen af at sponsorere udviklingen. Det kan også betyde, at de, der har vist evne til at deltage i kerne-udvikling, lettere kan få job og dermed få betaling for arbejdet.

Samarbejdsformen er grundlag for succesen

Der skelnes mellem indkommet kode (patch, changeset), accepteret patch, godkendt patch, committed patch og release.

Name	Changes	Percent
H Hartley Sweeten	2,089	2.2%
Sachin Kamat	1,374	1.4%
Jingoo Han	1,230	1.3%
Laurent Pinchart	953	1.0%
Jes Sorensen	772	0.8%
Daniel Vetter	764	0.8%
Malcolm Priestley	745	0.8%
Alex Deucher	727	0.8%
Lars-Peter Clausen	697	0.7%
Geert Uytterhoeven	685	0.7%
Ville Syrjälä	669	0.7%
Mark Brown	653	0.7%
Takashi Iwai	601	0.6%
Tejun Heo	594	0.6%
Joe Perches	581	0.6%
Dan Carpenter	538	0.6%
Axel Lin	526	0.5%
Al Viro	524	0.5%
Russell King	517	0.5%

Der er mange flere udviklere end de 19 viste

Siden begyndelsen af ***Git-æraen*** *(fra kerne 2.6.11 i 2005,* ***git*** *er det versions-arkiv system, som Torvalds udviklede nogle år før pga. utilfredshed med de eksisterende systemer)* - siden dengang har 11,695 udviklere bidraget til kernen. De udviklere arbejdede for 1230 firmaer. Antallet af firmaer, som sponsorerer Linux-udvikling, ser ud til at falde en anelse.

Selv om der er mange udviklere, er det et forholdsvis lille antal, som udfører hovedparten af arbejdet. Mange udviklere har bidraget med et enkelt patch. Top 30 af udviklerne bidrog ialt med 17 procent af total antal ændringer.

Indsendte patches ryger ind i et af ca. 100 subsystemer, hver med fokus på en speciel del af kernen (fx. netværk) og kontrolleres af en dedikeret udvikler; godkenes patch, bliver det "signed-off" af ham. Flere sign-offs betyder flere har gennemgået koden inden den endelig godtages (commit).

Hvem sponsorerer?

Linux kernen bruges af et stort antal firmaer - både til serversystemer, embedded systemer i alt fra routere og TV-apparater til proces-kontrol og mobiltelefoner - og de firmaers succes er afhængig af at kerne-udviklingsprocessen giver stabile resultater. Det er en væsentlig besparelse at man ikke skal begynde forfra på kerne-udvikling og så videre; firmaerne

På antallet af endelige godkendelser fra Linus Torvalds og andre betroede udviklere kan man se, at mere og mere af arbejdet uddelegeres, bl.a. til Greg Kroah-Hartman, David S. Miller, Mark Brown og Andrew Morton - de 4 i toppen. Deres navne støder man også på andre steder.

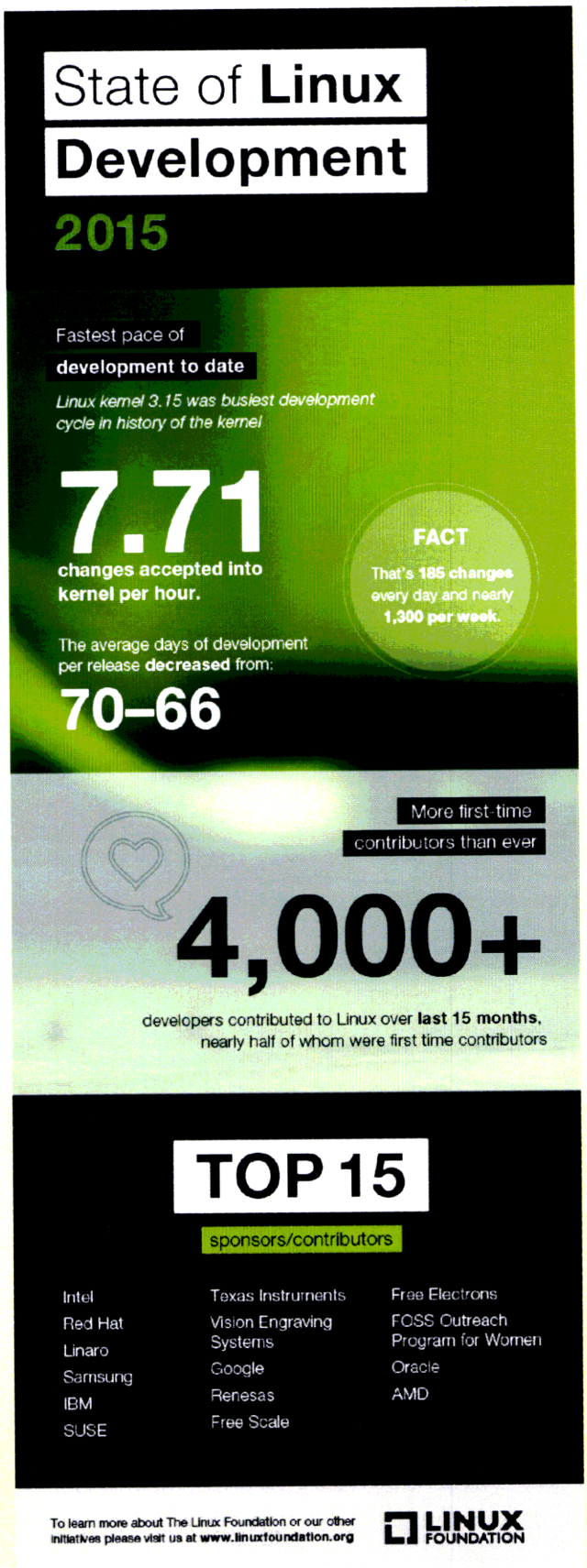
Har Unix fundet sin endelige form?

Med det antal nye drivere og nye features, som kommer ind i kernen, fordi det er den mest hensigtsmæssige måde at implementere dem, må man konkludere, at Unix ikke har fundet sin endelige form. I næste DKuug-NYT vil vi se på de principper, som gælder for kerne-udvikling - hvad der kan implementeres i user-space må blive i user-space, mens ting som fx. device håndtering må være en kerne-opgave.

Et eksempel:

En af de nyskabelser, som Corbet nævnte ved konferencen 18.feb. var control-group omarbejdelse. En control-group, (kontrol gruppe) er en Linux kerne facilitet, som implementerer en måde at måle og kontrollere en proces' forbrug af ressourcer, mere fleksibelt end **`nice`** og **`/etc/security/limits.conf`**

Man har så mange brugere af den eksisterende control-group struktur, at man også er nødt til at supportere den i kommende kerner, men det forhindrer ikke at man laver en ny, bedre.



Er Unix stadig Unix?

På den baggrund kan man spørge, om Unix stadig er Unix eller rettere, om Linux stadig er Unix?

Ja og nej - de grundlæggende design-principper bliver stadig fulgt, og Linux har som grundprincip at være en posix-compliant kerne. Men ud over de gamle egenskaber og API funktioner er der kommet så mange nye gennem de senere år, at man må se i øjnene at Linux processen og rationale må tages op til revision.

Vækst i størrelse

Kernen er så stor og vokser med en sådan hast, at selve arbejdsprocessen kan blive et problem, og dermed trues stabilitet og sikkerhed.

Derfor er test vigtigt - men det er også vanskeligt og der er uløste problemer, bl.a. fordi ikke alle hardware devices kan være til rådighed i én test.

Testmetoder kan være utilstrækkelige

I øjeblikket bruges tre værktøjer til test af kernen (regressionstest). Man prøver hver nat at kompilere kernen automatisk i et sæt grundlæggende konfigurationer, og det fejler, informeres de pågældende udviklere automatisk.

Men selv om kernen kompilerer kan der være fejl.

Derfor bruges også andre værktøjer: *Coverity,* statisk kode-check, *Trinity,* et fuzz-test system for nyligt tilføjede API til user-space, *Smatch,* source match, rensning/forbedring af kode, og *Coccinelle,* transformation og syntaktisk kontrolleret ændring af navne så man overholder navnekonventioner.

Hver af disse systemer fortjener en artikel, men i første omgang må vi henvise til LWN, søg fx. på ***LCA: The Trinity fuzz tester.***

Sikkerhed

Desværre var der en stribe sikkerhedsproblemer i 2014 - men på den positive side står, at man rapporterer og kan finde løsninger.

Der var 115 kerne-CVE (Common Vulnerabilities and Exposures).

Der er meget gammel kode, som ikke bliver vedligeholdt (testet imod nye versioner).

Der er alt for mange motiverede angribere.

Der er for få, som arbejder på problemerne.

Minimale embedded systemer

Man har indtil videre kunnet konfigurere en minimal-Linux, enten det nu er til en lille controller eller en server uden særlige features. Den var oprindelig meget lille, denne minimal kerne, men alene memoryforbrug er i de nyeste versioner på 2MB.

Det går ikke, hvis man vil lave et *Internet af ting,* kaffemaskiner og elektrisk belysning, som ikke må koste noget og fremstilles i store mængder - så er selv en lille besparelse et konkurrenceparameter.

Nogle leverandører løser dette problem ved at gå tilbage til fx. kerne 2.4, men det er jo ikke en måde, der holder i længden, hvis man skal minimere arbejdet med kernel-maintenance. Imidlertid kan forskellen mellem embedded systemer og fuldskala workstations mv. blive så stor at man er nødt til at lave et split.

Man kan følge Jonathan Corbet på Google+

■

MicroPython board

Python er blevet mere og mere populært - både til system administration og blandt udviklere

Af David Askirk, instruktør hos SuperUsers

Python er gennem flere år blevet kåret som det bedste scripting sprog og bedste shellscripting sprog i flere linux blade.

Mini computeren Raspberry Pi benytter python som det primære programmeringssprog.

Python er også brugt i undervisning, da det er meget læsevenligt. Python bliver også kaldt Executable Pseudo kode.

Pythons store brug skyldes at det er simpelt, og samtidig stærkt, og har et stort standard bibliotek, hvor man kan næsten alt.

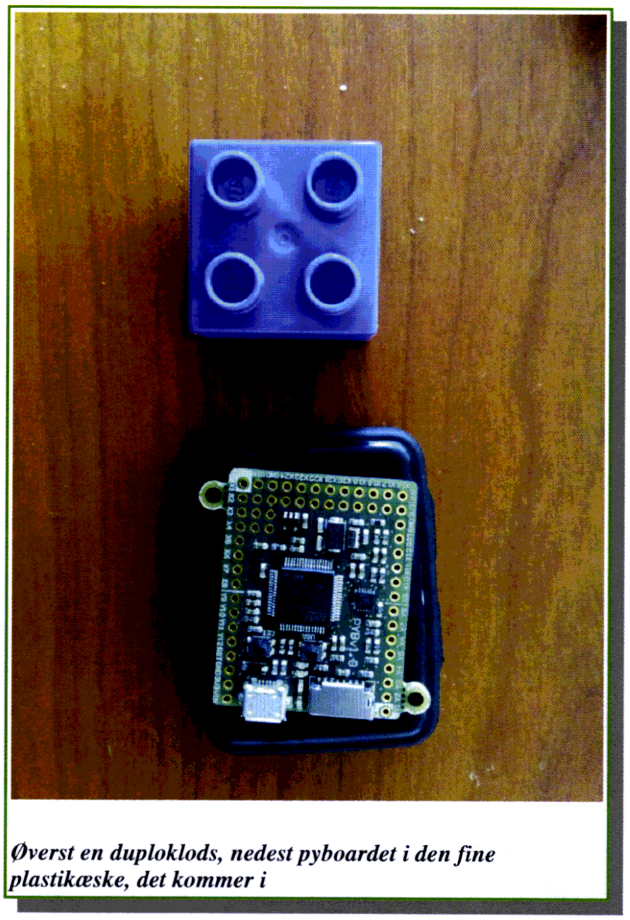
Ind til nu har python været forbeholdt computere med et styresystem. Denne tid er forbi nu!

Micro Python er nemlig kommet. MicroPython er en python der er bygget til at køre på microprocessorere. Pt. kan man købe et board hvor man kan køre python på selve "jernet".

I denne artikel vil vi kigge på hvad et micropython board kan bruges til og besvare spørgsmålet: Kan et micropython erstatte en Arduino, som er standard i meget gør det selv elektronik.

MicroPython boardet kan fås herfra: https://micropython.org/

Boardet tilsluttes computeren med et micro-usb kabel. Det samme kabel som man bruger til en raspberry pi eller til at lade nogle telefoner.



Øverst en duploklods, nedest pyboardet i den fine plastikæske, det kommer i

Hardware

Boardet er 3 cm x 5 cm, så det er ikke stort. Der er et micro-usb stik samt en micro-SD kortlæser som man kender fra en telefon.

Der er to knapper, den ene kan man bruge i sine programmer, den anden bruges til at resette boardet. Der er også fire LEDs som man kan bruge i sine programmer. Udover dette er der 34 porte til at lodde sine eksterne komponenter på, som man kan bruge til sine elektronik projekter. Dette er en del mere end der er på en Arduino!

Der er også mulighed for UART, I2C, Timers, PWM (pulse width modulation), ADC, DAC og SPI, så dette åbner virkelig muligheden for at snakke med meget forskelligt elektronik.

Den første tilslutning

Når man tilslutter boardet bliver der mountet et drev, ligesom hvis man putter en usbstick i sin computer. På dette share ligger der fire filer:

README.txt en readme fil, der forklarer hvordan man får adgang til boardets indbyggede serielle konsol.

boot.py - Python kode der kører når boardet booter

main.py - Selve programmet som bliver kørt efter boot.py

pybcdc.inf - Windows driver til at få adgang til den serielle konsol fra windows.

Det er i main.py man skriver sit program, så det er også i denne fil vi kommer til at arbejde i denne artikel.

Software

Et micropython board bliver programmet i python.

Følgende biblioteker fra pythons standard bibliotek er til rådighed (se

http://docs.micropython.org/en/latest/library/cmath.html)

1. cmath – matematik for complekse tal
2. gc – Garbage collectoren og styring af denne
3. math – matematiske funktioner
4. os – få et kig ned i "styre systemet"
5. select – vente på events.
6. struct – pakning og udpakning af datatyper
7. sys – system funktioner
8. time – tidsfunktioner

Udover dette er følgende libraries specielt lavet til micropython til rådighed:

1. ubinascii – konvertering mellem binære og ascii
2. uctypes – få adgang til typerne fra C
3. uhashlib – hashing algoritme
4. uheapq – en hob kø
5. ujson – json værktøjer som encoding og decoding
6. ure – regexs
7. usocket – sockets
8. uzlib – zlib dekomprimering

Der er endda en feature mere: Hvis man har et script, der har import json, leder den først efter json.py, dernæst json/ altså et directory kaldet json, hvor den tilsidst vil importere ujson.

De specielle pyboard funktioner ligger i et bibliotek der hedder pyb. Det er igennem dette bibliotek man kan få fat på LED, switch, acceleraometer og andet hardware som boardet stiller til rådighed.

Det er dette bibliotek vi kommer til at bruge i resten af artikelen.

Det første program

Det første program som vi skal skrive til boardet er et program der får en lysdiode til at blinke. Vi vil arbejde videre med dette program så det bliver en ret avanceret blinker til sidst.

Først, tilslut microboardet hvis det ikke allerede er tilsluttet.

Åben main.py i din absolut yndlings teksteditor af dem du har til rådighed.

Udfyld main.py så der står følgende tekst:

```
import pyb

led = pyb.LED(1)

while True:
    led.toggle()
    pyb.delay(500)
```

Unmount boardet og genstart det, enten ved at pille kablet fra og sætte det til igen, eller endnu nemmere, at trykke på reset knappen. Den er markeret med RST.

Nu står der en rød diode og blinker med et halvt sekund mellem at den tænder og slukker.

Lad os lige skære python programmet op i bidder:

```
import pyb
```

Her importere vi pyb modulet i programmet og gør det tilgengængelig i programmet.

```
led = pyb.LED(1)
```

Lad variablen led pege på den første LED. Der er fire LED i alt på pyboardet.

```
while True:
```

Dette er en løkke som kører hele tiden. Den kører så længe True er sand, og det skulle gerne gælde til evig tid.

```
led.toggle()
```

Skift tilstand på led. Hvis den før var slukket skal den nu være tændt og omvendt.

```
pyb.delay(500)
```

Vent i 500 ms førend vi går videre.

Det var det første program. Det næste program vi skal skrive er en lommelygte, nemlig vi skal bruge knappen som der er adgang til, så når man trykker på knappen tænder LED.

Lommelygte program

Åben igen main.py og erstat indholdet med følgende:

```
import pyb

led = pyb.LED(1)
switch = pyb.Switch()

while True:
    if switch():
        led.toggle()
```

Lad os se på det som er nyt:

```
switch = pyb.Switch()
```

Her får vi en adgang til knappen. Senere kan switch() kaldes; den returnerer True eller False, der angiver om knappen er trykket ned eller ej.

```
if switch():
```

Her bliver der undersøgt om switchen er trykket ned. Hvis det er tilfældet, skifter vi LED tilstand, så den skifter fra tændt til slukket -- eller omvendt.

Hvis man kører det her program, vil man opdage at det ikke er altid at knappen fanger resultatet. Det skyldes at den er så hurtig til at se om vi har trykket, og kommer rundt i loopet igen inden vi når at slippe knappen.

Hvis man gerne vil forbedre den lidt, kan man putte et delay ind således:

```
import pyb

led = pyb.LED(1)
switch = pyb.Switch()

while True:
    if switch():
        led.toggle()
    pyb.delay(200)
```

Her er der puttet til delay ind til sidst, hvor vi siger at vi venter 200 ms. Så vil den være mere stabil når vi trykker på knappen.

En lysdiode er ikke det eneste vi kan bruge sådan et stykke elektronik til. Der er et indbygget accelerometer.

Nu skal der laves et lille "vaterpas" med de fire LED og accelerometeret.

For at aktivere accelerometeret bruges følgende stump kode:

```
accel = pyb.Accel()
```

For at vise hvordan det virker, vil vi udnytte at når man tilslutter et pyboard, får man også adgang til en repl. (**R**ead-**E**val-**P**rint **L**oop)

Forbind boadet til din computer, og der dukker nu et `/dev/ttyACM0` device op på din maskine. Forbind til dette med fx. screen.

Screen er en terminal - multiplexer med VT100/ANSI terminal emulering. GNU Screen terminal multiplexeren kører flere separate "skærme" på en enkelt fysisk karakter baseret terminal. Hver virtuel terminal emulerer en DEC VT100 plus flere ANSI og ISO 2022 funktioner. Screen sessions kan frakobles og genoptaget senere på en anden terminal.

Screen supporterer meget mere, bl.a. IOoversættelse og multiuser.

Description:

GNU ***Screen*** is a terminal multiplexer that runs several separate "screens" on a single physical character-based terminal. Each virtual terminal emulates a DEC VT100 plus several ANSI X3.64 and ISO 2022 functions. ***Screen*** sessions can be detached and resumed later on a different terminal.

Screen also supports a whole slew of other features, including configurable input and output translation, serial port support, configurable logging, and multi-user support.

Homepage: http://savannah.gnu.org/projects/screen

Når du er forbindet til boardet, får du en python repl til boardets python, - en kommando-linie eller loop, som man kender fra en almindelig computer.

Først skal pyb importeres så det er klart til brug:

```
import pyb
```

Så kan vi tage fat i acceleraomteret:

```
accel = pyb.Accel()
```

For at se hvad der er tilrådighed kan dir kommandoen bruges:

```
dir(accel)
```

Dette giver et overblik over hvilke ting der er til rådighed på accel objectet.

Accelerometerets metoder/funktioner

Accelerometeret har en **y** metode.

```
>>> accel.y()
-2
```

Det kan være den viser en anden værdi hos dig. Prøv at flyt rundt på boardet og prøv kommandoen igen:

Nu skifter y værdien. Det er denne y værdi som vi kommer til at arbejde med. Nu skal vi lave en lille "bold" der flyver frem og tilbage over de 4 LEDs.

Hop ud af repl og åben main.py.

y() metoden returnerer et tal mellem **-30** og **30**. Dette skal vi få til at passe på 4 LED, så hver LED får en range på ca 15 at dække over.

Her kommer programmet, så gennemgår vi det bagefter:

```
import pyb

ball_pos = 2
leds  =  [pyb.LED(1),  pyb.LED(2),  pyb.LED(3),
pyb.LED(4)]
accel = pyb.Accel()

for led in leds:
    led.off()

while True:
    y = accel.y()+30
    ball_pos = int(y/15)
    if ball_pos < 0:
        ball_pos = 0
    elif ball_pos > 3:
        ball_pos = 3
    for led in leds:
        led.off()
    leds[ball_pos].on()
    pyb.delay(200)
```

Udover *import* til at starte med begynder programmet med nogle variabel erklæringer, først **`ball_pos = 2`**

Her sætter vi en variabel op der holder styr på den position som bolden har, eller sagt på en anden måde, hvilken ***LED*** som skal være tændt.

Variablen ***leds*** indeholder alle de LEDs, der er til rådighed på boardet. Tilsidst laver vi en henvisning til accelerometere. Den henvisning skal bruges til at aflæse værdier fra.

```
ball_pos = 2
leds  =  [pyb.LED(1),  pyb.LED(2),  pyb.LED(3),
pyb.LED(4)]
accel = pyb.Accel()
```

Dernæst slukker vi lige alle LEDs. Dette er for at gøre klar til programmets kørsel.

```
for led in leds:
    led.off()
```

I den næste del kommer det, som bliver gentaget i while loopet:

```
    y = accel.y()+30
    ball_pos = int(y/15)
    if ball_pos < 0:
        ball_pos = 0
    elif ball_pos > 3:
        ball_pos = 3
```

I denne del får vi **y** fra accelerometeret. Det er en værdi mellem -30 og 30. Derefter beregner vi hvilken LED der skal tændes. Det skal ende ud som et tal mellem 0 og 3. For at være helt sikker på at det er indenfor området 0 .. 3, sætter vi **ball_pos** til 3 eller 0 hvis den den har overskredet de grænser.

Man kunne også have brugt følgende sætninger:

```
ball_pos = min(3, ball_pos)
ball_pos = max(0, ball_pos)
```

I den næste del tænder vi for en LED.

```
    for led in leds:
        led.off()
    leds[ball_pos].on()
```

Til sidst venter vi 200 ms for at nå se hvilken lys diode der er tændt.

```
    pyb.delay(200)
```

Når man kører programmet og vipper med boardet, vil man se at lyset flytter sig frem og tilbage alt efter hvilken vinkel man holder boardet i.

GPIO - General Purpose Input Output

Den sidste vigtige del af et micropython board som vi kommer til at beskrive, er muligheden for at tilslutte egen elektronik til boadet.

Dette gøres med GPIO porte. Dette er porte som man kan sætte høj eller lav, eller bruge som input.

Der er nogle af disse, der er special porte; som fx. PWM som man kan sætte til at være høj i **x** tid ud af **y**. Dette gøres ved at de skifter mellem høj og lav. Dette kaldes puls bredde modifikation. Dette kan bruges til at få en LED til at fade hvis det ønskes.

Få at få adgang til et en bestemt port eller pin bruges ***Pin klassen:***

```
x1_pin = pyb.Pin.board.X1

g = pyb.Pin(pyb.Pin.board.X1, pyb.Pin.IN)
```

Her bliver den pin der hedder X1 sat til at være en input pin. Så kan man aflæse værdien af pinen.

Arduino eller Micropython?

Hvad skal man så vælge?

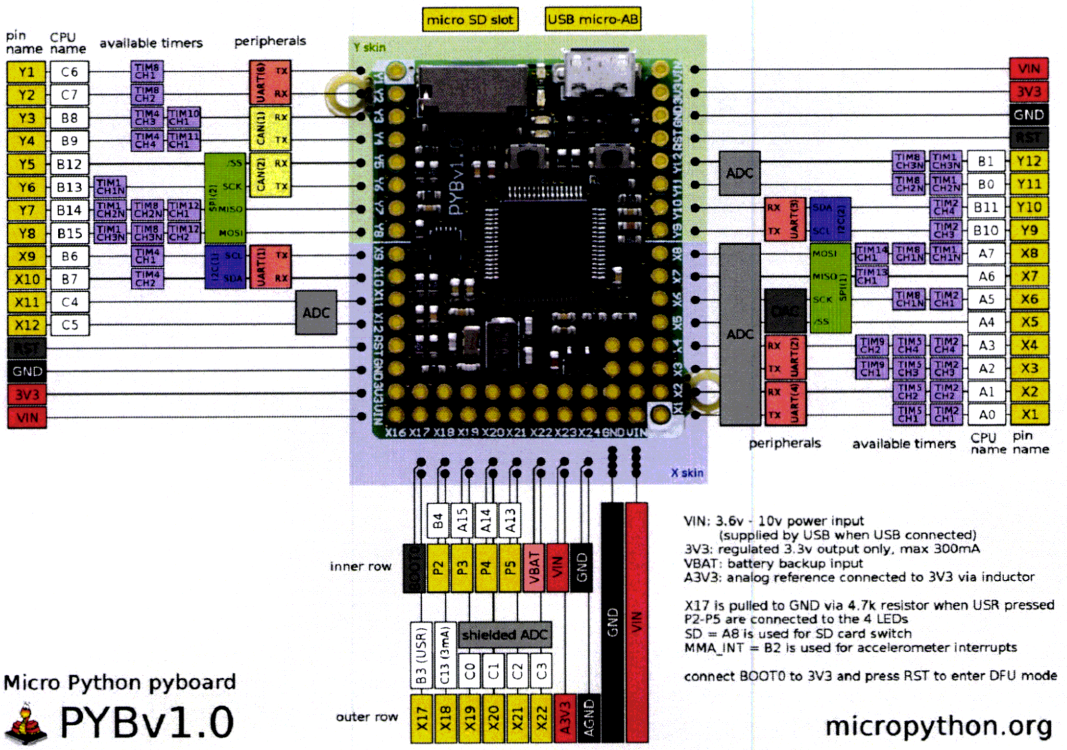
Hvis man søger efter noget hardware til at styre et eller andet stykke elektronik er der stor sandsynlighed for at der allerede er nogle der har gjort det med arduino.

Men micropython programmers i python, som er et nemmere sprog end den c udgave som arduino benytter.

Et micropython board er en smule dyrere end en arduino, men det skal nok komme ned i pris.

Personligt skal jeg helt sikkert bruge micropython boardet til noget mere, da det er så nemt at programmere, og samtidig har virkelig mange muligheder. Herunder også til undervisning i elektronik og programmering.

■



Future Internet - FI-ware

16. marts blev et Open Source initiativ i EU regi annonceret ved CeBIT

Af Donald Axel

Ved CeBIT blev det annonceret af EU kommissionen at man støtter et projekt med ambitioner om at lette arbejdet for en række miljøvenlige projekter ved at sponsorere og koordinere projekter, som betegnes *Smart Cities*. Projektet kaldes Open & Agile Smart Cities (OASC).

Kort fortalt går det ud på at anvende FI-ware rammerne til hurtigere at få resultater med digitalisering i infrastrukturer. Dermed håber man, at give borgerne bedre service med offentlig transport når der er brug for den, bedre trafik-regulering, sundhed (e-journaler) og materialer og metoder til undervisning mv.

FIware er middleware (godt dansk ord :)) og et framework for ud- og afvikling af applikationer, som drager nytte af netværk, "cloud", og skal især lette udviklingen af applikationer (herunder smartphone apps) der bruger byernes egen infrastruktur. Eksempler på middleware er netværksprotokoller, talegenkendelse, scanningskoder.

FIware har forskellige sektioner: sikkerhedsprocedurer (herunder ID og login), storage, deployment, konfiguratiion og monitorering og som nævnt, programudvikling.

Man vil også kunne forbinde byerne - d.v.s. dele informationer også kommer jura ind i billedet.

Det var Mario Campolargo fra EU-kommissionens DG-Connect, som kom med nyheden ved konferencen. Hans announcement er led i EU-kommissionens beslutning om at støtte Open Source anvendelse, så man opnår flest muligt fordele og samlet løser de lovmæssige problemer. (Wikipedia har 10 siders beskrivelse af retssager, herunder EU vs Microsoft, den sidste en antitrust-sag fra 2011.)

Danmark deltager

Der deltager 31 byer i OASC, byer fra Finland, Belgien, Portugal, Italien, Spanien, Brasilien - og **Danmark.**

Det var ordet *Danmark,* som fangede redaktionens opmærksomhed, og på http://www.fiware.org/smart-cities/ ses at det er København, Aalborg og Aarhus fra Danmark som er med på vognen.

Hvad er FIware?

Det erklærede formål med OASC og dermed også FIware er at gøre det lettere, mere rationelt og effektivt, at udvikle services indenfor flere forskellige områder herunder logistik, vedvarende energi og miljømæssige, bæredygtige projekter.

Det lyder som den sædvanlige omgang velmenende ord og gode viljer uden noget at have dem i, men i dette tilfælde er det måske alligevel noget andet, for man har valgt at basere sig på store Open Source projekter, som er i brug, og - som der står på den engelske wikipediaside for FIware: Det er essentielt at man får brugere og udviklere til at benytte det udviklede framework, hvis platformen skal blive en standard og en løsning, som kan gøre nytte. Man kunne fx. forestille sig at de store projekter indenfor politi og rejsekort ("Amanda-projekter") kunne være undgået, hvis der havde været vedtagne rammer, som tilbød et velfungerende stabilt grundlag, så man ikke skulle opfinde den dybe tallerken hver gang.

Faren ved at udbrede FIware og OASC som en fix og færdig applikationsplatform for avancerede distribuerede projekter er imidlertid at de bevilgende myndigheder ikke mener, at man skal bruge så mange penge udvikling.

Jeg ville ønske, at platformen blev markedsført som en ***stabil platform, med tryk på vedligehold og investering*** og hvor ***resultatet er bedre IT*** og ikke bare smarte byer.

Men præsentationerne - der er nemlig mange - er ikke så dårlige endda.

Hvordan præsenteres FIware?

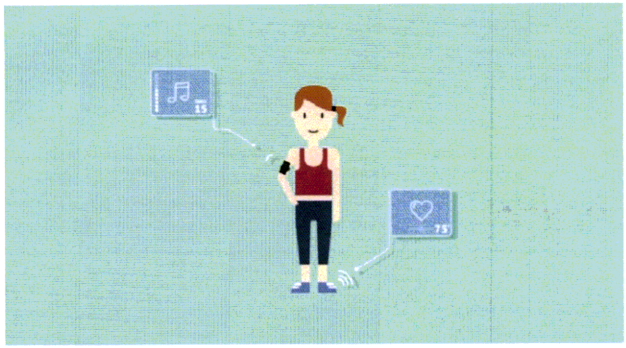
De første websider på fiware.org er en flot facade (se nedenfor) af den slags, hvor man undrer sig om der er noget bag facaden, blot buzzwords, lidt som en politiker der begejstret argumenterer for digitalisering af folketingsvalg mv. uden at have respekt for problemer som fx. adgangsforhold til E-journal osv.

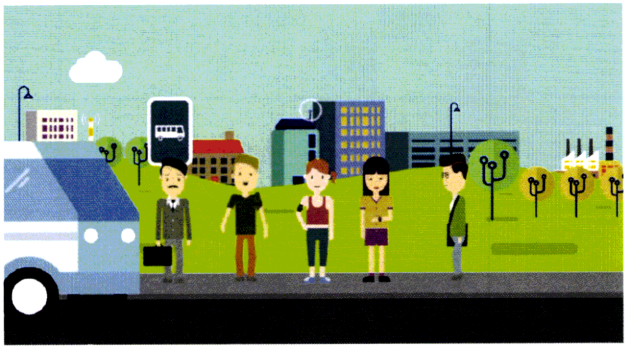
Men der er også andre præsentationer. Slå op på engelsk wikipedia, ***fiware,*** og find eksternt link til FI-WARE wiki. Her kan man klikke sig til en quick tour.

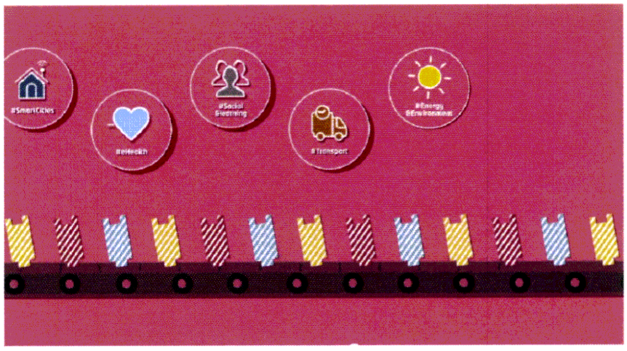
Det interessante ved den tour er, at man bliver bedt om først at læse rationalet for projektet, baggrund og visioner. Derefter skal man gå igang med de tekniske beskrivelser.

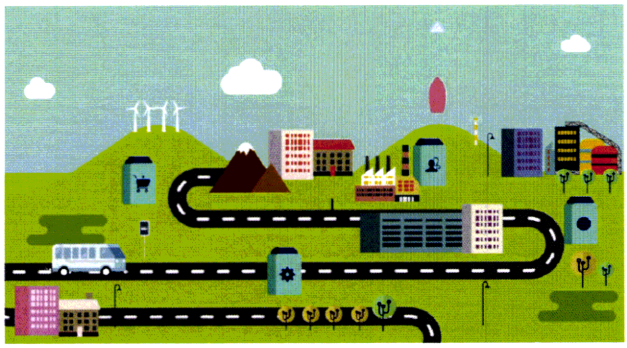
Forsiden på fiware.org viser de fire hovedområder

1. Hjælp til iværksættere (Small Medium Entrepreneurs).
2. FIware udviklere af selve applikationsplatformen.
3. Drift og hosting - det kræver også uddannelse og indsigt.
4. Domain stakeholders: Offentlige administrationer og virksomheder.









Tv: Video-præsentation af ideer og visioner i Future Internet, Internet of Things (IoT).

Målet er udbredelse af standardiseret API

Set i et teknisk og system-administrativt perspektiv er formålet udbredelse af en stak af Open Source elementer, - lidt i stil med LAMP, men længere ud til de enkelte applikationers protokoller - så man kan bygge infrastruktur uden at begynde forfra hver gang.

En bemærkning fra en anden foredragsholder på Kernel Collaboration Summit kan illustrere det: Et hold programmører undrede sig over, at to mobiltelefoner i nærheden af hinanden ikke kunne kommunikere direkte til hinanden og undersøgte hvordan og hvorledes, og fandt ud af at 99% (ca.) af app-programmører ikke anede noget om radiokommunikation. Så man fandt ud af, at man kan lave drivere til Smartphone-hardware, der taler direkte til andre mobiltelefoner i nærheden og derved kan man åbne for applikationer af en anden slags.

På samme måde kan man ikke forvente at finde en stribe programmører, som alle har kendskab til cloud ressourcer for IoT, Internet of Things (som visionerne på billededserien til venstre).

Der skal uddannelse til, og hvis EU politikere sammen med politikere fra andre lande og store organisationer, som bruger Open Source (Brasilien, Nasa, Google, Facebook, etc) lægger sig fast på et applikations-interface (API) til infrastruktur programmering, så er der større chancer for, at det lykkes at få først byer og senere provinser og lande til at bruge et fastlagt, velanskrevet, velfungerende system bestående af Open Source komponenter.

En departement i EU har skrevet en målsætning for Open Source anvendelse i EU i 10 punkter. Her står bl.a.

Kommissionen skal formelt fortsat vedtage brugen OSS ved hjælp af produkt management procedurer.

Kommissionen skal promovere brugen af produkter, som understøtter kendte, veldokumenterede, åbne tekniske specifikationer som frit kan bruges, implementeres og udvides. Interoperabilitet er et kritisk spørgsmål og brugen af veletablerede standarder er måden at opnå dette.

Kommissionen skal fortsætte med at udvikle best practice og værktøjer, som stammer fra OSS communities, samtidig med anvendelse af nyeste praksis særligt indenfor sikkerhed.

(Det er kun udpluk af den 10 punkter lange Open Source Software Strategi for 2014-2017.)

Det er vigtigt at kommissionen som sådan skal udvikle best practice for udrulning af løsninger som omfatter både OSS og proprietær software. Man ser en blandet verden og - med rette - skal man lægge vægt på interoperabilitet.

Så må vi se, om politikerne kan presse leverandørerne til at følge standarderne. Som regel er der altid en grund til at prøve noget nyt, det skal man ikke være blind for, men det må foregå i passende små trin udenfor systemer, som er afgørende for samfundsfunktioner.

Hvis det skal lykkes, skal kommissionen (og dermed politikerne) have rådgivere, som kan overskue hele feltet af tekniske løsninger.

Findes de rådgivere? Kan politikere og EU kommissærer forstå hvad rådgiverne siger? Som det så ofte har været nævnt har de store spillere lobbyister i Bruxelles, og karakteristisk for Open Source er netop, at der ikke er en stor marketingsuddannet skare lobbyister. Måske udbredelsen af Linux i smartphones har overbevist politikere om værdien af Open Source princippet. IT-"økosystemet" er ekstremt dynamisk, og Open Source løsninger har vundet stort pga. at vidensdeling nu engang er mere rationelt.

Ovenover: Der skal være en rumraket med i en salgspræsentation - det signalerer højteknologi. Det er nu ikke helt ved siden af, for Nasa er med i OpenStack, som er af basis for FIware projekterne.

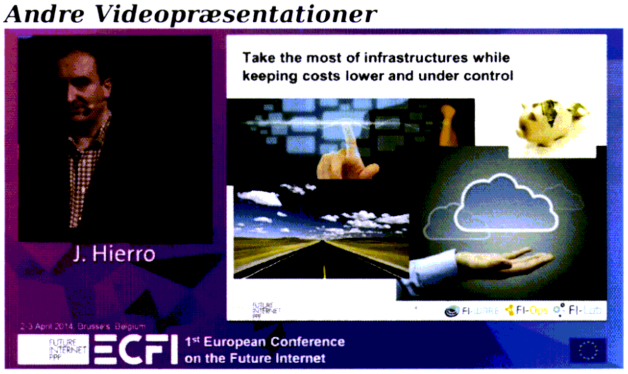
Finansiering og Public-Private Partnership

Hvis det skal lykkes, skal EU sørge for at der er finansieringsmuligheder for små og mellemstore virksomheder, og teknologi-opstartsmuligheder. Det er der ikke samme tradition for i Europa som der er i USA.

Men finansiering indgår som et program i FIware; det kaldes Entrepreneur & SME Accelerator Programme. Traditionelt er investorer mere positive overfor ingeniør-uddannede, men indenfor computerprogrammering er det kendt, at det er track record, man skal tage bestik af. Der findes mange Open Source projekter, som styres af programmører med andre uddannelser end softwareingeniør hvilket også afspejles i måden de nye IT-uddannelser er opbygget.

Finodex projektet har partnere from hele Europa, inklusive Copenhagen Business School. Det er et EU finansieret projekt. Det giver gode finansieringsmuligheder for de mest lovende startup-firmaer i EU som bruger Open Data og FIware teknologi.

Andre Videopræsentationer

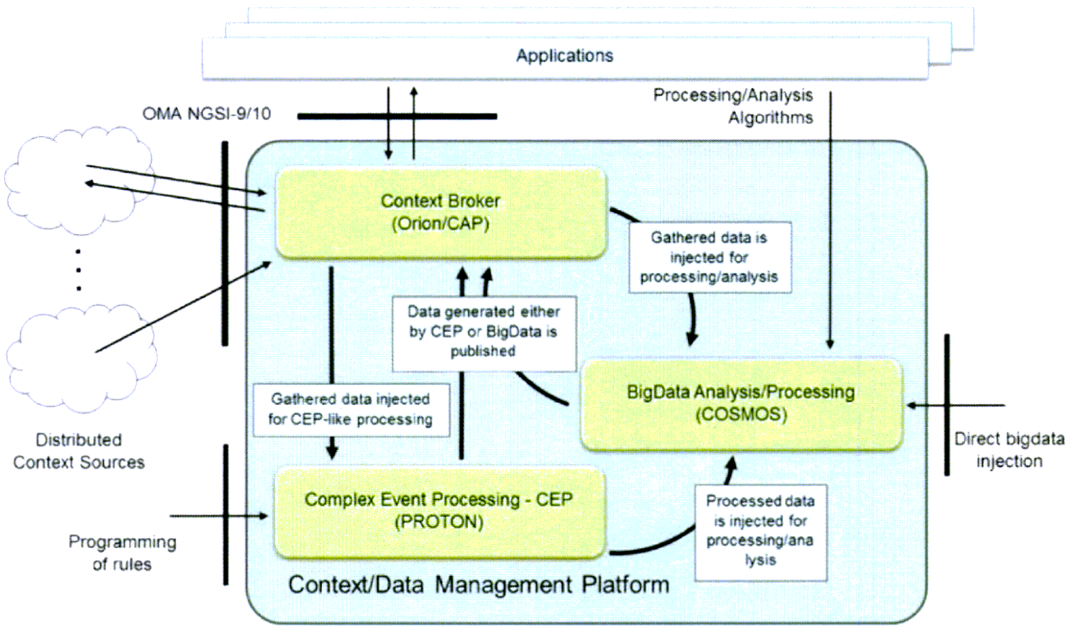


Der er mange video-præsentationer på YouTube, som vedrører FIware. En af de mere omfattende er et foredrag af J.Hierro, - søg på *FIWARE Lab: A guide to the most exciting Future Internet ecosystem - Juanjo Hierro*

Et diagram over en data management platform fra denne præsentation ses nederst på siden.

■

FI-WARE Context/Data Management Platform



Kommando(linie)centralen

Nye stillingsbetegnelse, mere ansvar

Kommandoliniefortolkere og fortolkere i det hele taget muliggør udrulning og konfigurering i stor skala - scalability. Det kræver hele tiden nye værktøjer. Systemadministration er blevet til Development, Quality Assurance & Technology Operations - DevOps.

Grunden til, at vi har denne rubrik i DNyt er, at både erfarne og uerfarne sysadm'er (og DevOps, SRE'er - Site-Reliability Engineers) har brug for at kunne *automatisere* opgaver - og det kræver scripting og programmering.

Når det forekommer, at en statsvirksomhed betaler for en simpel opsætning af 4 Xterm, som bruger tcpstat til at vise netbelastningen på 4 punkter i virksomhedens netværk, så kan man godt få mistanke om at virksomhedens kompetencepleje er ikke-eksisterende.

Vi er ikke født som eksperter i IT. Der skal uddannelse og evner og slid til. Det er 1% evner og 99% slid!

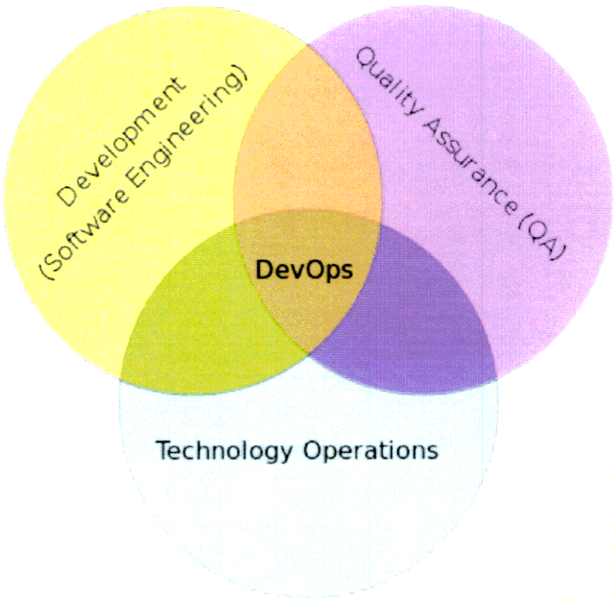
Moderne drift

Det sker, at en ledelse mener, at "driftsfolk ikke må programmere". Men det er at skyde sig selv i foden. Effektiv drift kræver programmering - mest i form shell-scripts.

De store interaktive web-tjenester har vist, at en anden holdning til udvikling og drift er nødvendigt.

DevOps - Development+Pålidelighed

DevOps er en programmør med ansvar for drift eller rettere systemkonfiguration og start-stop af driftsoperationer, og med indsigt i pålidelighed og sikkerhed.



Der er faktisk miljøer, hvor drift og udvikling skal være adskilt med vandtætte skodder.

En udvikler er en meget betroet person, og rolledelingen var skrap for at undgå at privilegierne blev misbrugt. Hvem husker ikke Richard Pryor som den utroligt charmerende "skurk" Gus Gorman i Superman III? Gus får et job som programmør og sender alle cent-afrundinger til sin egen bank-konto, hvilket skaffer ham mange penge. Hvis han nu ikke havde haft adgang til driftssystemet ... så var der ikke blevet en morsom film ud af det.

Men de mest dynamiske virksomheder bruger ikke længere driftsoperatører som nat efter nat holder et øje på monitoreringen mens der drikkes kaffe og læses Anders And og ses TV. Sikkerhedsproblemerne løses på andre måder.

Programmering i driftsmiljø

For at opnå høj oppetid på store sites med mange servere bruger man fx. Puppet til udrulning, konfiguration og monitorering. Men selv om CFEngine og Puppet mv. er gode systemer, så oplever man problemer, når man har brugt dem et stykke tid: Der er mange opsætningsfiler til Puppet & Co. med domænespecifikke syntax, og der skal ustandselig skrives og rettes. Somme tider er det lettere at bruge grundværktøjer (ssh og python eller lignende) til deployment og cross-system konfiguration, fordi man ved, hvad de bagvedliggende problematikker er.

Man finder også hele tiden selv nye metoder til monitorering af HW tilstand. Man vil finde ud af, hvor og hvorfor systemerne (og tidligere: H/W) fejler - gerne ***inden*** de fejler. Her er et par groft simplificerede eksempler:

Alle kender det at bruge kommando-linie tools til logscanning.

```
# grep -i warning /var/log/messages*
```

Disk-IO testes typisk med smartctl og grep på output fra loggen, men for en ordens skyld laver vi et ***urealistisk simpelt script*** som demonstration. Vi bruger device-copy, dd, som "motionerer" hardwaren:

```
# dd if=/dev/sdb1 of=/dev/null bs=1M
```

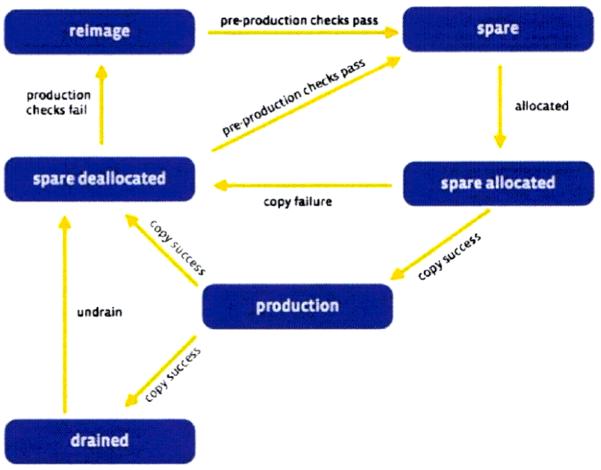
Det går ikke at belaste systemet med monitorering, så vi må hellere sende jobbet afsted i små portioner:

Fx. læs 2 GB data ad gangen fra en disk, find antal megabyte, som disken rummer, med df -m kommandoen:

```
#!/bin/bash
  let position=0
  let MAX=587640
while ((position < MAX))
{
  nice ionice -c idle \
    dd if=/dev/sdc1 of=/dev/null bs=1M \
    skip=$position count=2000
  let position=((position + 2000))
  sleep 300
}
grep -q -i error /var/log/messages || alarm
```

IOnice option **`-c idle`** (class idle) bevirker at dd (device-copy) kommandoen kun får lov at læse disk, når andre ikke gør. Ved mine små 2GB test-reads kan jeg ikke måle forskel på en applikation, som kører med/uden dd i baggrunden.

For at se, om dynamisk webside kører som den skal, må man "skyde" mod serveren med fx. wget eller andet scriptbart netværktøj, fx. netcat.



De store webservices (fx. Facebook) har automatiseret en server life-cycle, og når en igangværende server ses at fejle, drænes den for data og derefter repareres og geninstalleres.

■

TIL DET DANSKE IT-FOLK

KURSER TESTS CERTIFICERINGER FIRMAKURSER KONSULENTER

FINDER

Vælg Kurser | Vælg producent | Vælg kategori | Vælg teknologi

APPLE CISCO GOOGLE MICROSOFT OPEN SYSTEMS ORACLE ØVRIGE

MOBILE PLATFORME SERVER & DESKTOP NETVÆRK DATABASER, BI & SQL PROGRAMUDVIKLING WEB TEKNOLOGIER ØVRIGE

Hvorfor vælge os?

Kurser med omhu og kvalitet
Du går på kursus for at lære noget nyt, som du kan bruge, når du kommer hjem. Vores erfarne, fastansatte instruktørkonsulenter gør deres ypperste for, at du forstår teori og begreber, inden du laver øvelser for at opnå praktisk håndelag.

Undervisningen foregår i spændende og rolige omgivelser, hvor du kan koncentrere dig – og blive forkælet på samme tid.

300 forskellige kurser
Vi har Danmarks største udbud af kurser, tests og certificeringer inden for operativsystemer, netværk og programmering.

Hands-On med mening
Hos SuperUsers tror vi ikke på "trial and error", altså at prøve sig frem. Hands-on er vigtigt, men først skal man have begreber og baggrund på plads.

Mesterlære = Master Class
Den dygtige instruktør er kernen i SuperUsers. 80% af et godt kursus kan tilskrives en motiverende, fagligt dygtig og velformuleret instruktør. Formidling er altafgørende for dit udbytte af kurset!

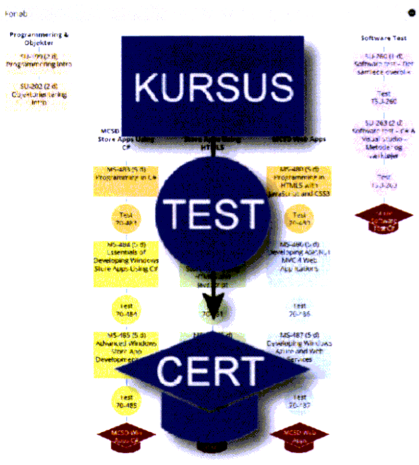
50.000+ kursister siden 1984
I SuperUsers har vi undervist mere end 50.000 kursister inden for operativsystemer, netværk og programmering. Spørg efter referencer, vi har dem i massevis.

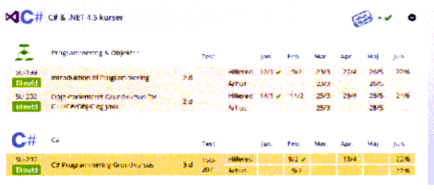
AFHOLDELSESGARANTI
på mere end 150 kurser i nærmeste fremtid – se superusers.dk

SUPERUSERS

Danmarks største IT-kursuscenter har fået ny hjemmeside

Nemt og overskueligt:
Det nye website er let at betjene, og du finder lynhurtigt det, du søger.





Mere end 300 kurser og 100 certificeringer med tilhørende tests

Mobile Platforme
Europas største udbud af kurser, tests og certificeringer på både Android, iOS og Windows Phone.

Server & Desktop
Windows: Vi er Microsoft Certified Training Partner og holder alle Windows- og BackOffice-kurser inklusive PowerShell og Hyper-V.
LINUX / UNIX: Vi holder masser af kurser i Linux og UNIX inkl. Apples OS X og LAMP – med tests og tilhørende certificeringer.

Netværk
TCP/IP, den åbne standard som bruges alle steder. Mange TCP/IP- og CISCO-kurser, tests og certificeringer.

Databaser, BI & SQL
Fokus på SQL & BI samt kurser i T-SQL, Microsoft SQL Server, MySQL, MariaDB, PostgreSQL, SQLite, …

Programudvikling
Kurser, tests og certificeringer i C, C++, C#/.NET, Obj-C/Swift, Perl, Python, PHP, Ruby, UML, Software Test …

Webteknologier
Aldrig har Web Apps haft større fokus end nu: AngularJS, Bootstrap, jQuery, jQuery Mobile, PhoneGap, NodeJS, …

FIND UGENS PÅSKEÆG
på superusers.dk – og vind en iPad Air

www.superusers.dk