DKuug nr. 182 December 2016

NYT

Dragonboard - 410c

En kraftig computer på størrelse med en tændsikæske (af de lidt større) side 4

Kernel Report side 8

Går solen ned for Posix?

Er Unix forældet?

Er Posix forældet?
Hvor knager det?
Vi har brug for standarder - side 12

Hvor mange systemkald er Posix-funktioner? side 16

ISO og LSB side 18

Dansk Forum for Åbne Systemer
DKUUG - Unix Brugere (Linux/BSD) system administratorer
Community for IT-specialister og IT-interesserede.

Et brev fra redaktøren

Jeg ville hellere skrive kode - fortalte Plauger om sit arbejde med at skrive specifikationerne for et standard-library

Men med alderen kunne han se, at det var lige så vigtigt at få opsat et effektivt og rationelt computer-miljø som at skrive og køre programmer.

Den situation havner vi alle i engang når vi har lavet så mange programmer, så vi kan se omridset af en løsning længe inden vi har kodet den - og vi kan også lave løsningen, for det har vi prøvet mange gange. Men standardisering af kommando-interface (og GUI for den sags skyld) sparer tid og på verdensplan millioner af kroner eller dollar, så nogen skal arbejde med det - nogen, helst med stor faglig indsigt i computersystemer, konfiguration, programmering, brugerkrav, IT-skandaler og menneskelige fejl.

Nogen, som hellere vil skrive programmer, bliver nødt til at interessere sig for og facilitere processerne med at få en standard til at hænge sammen.

Det er der økonomi i!

Engang MSoft havde en sløvhed, som bevirkede at en sekretær måtte bruge 10 eller 20 sekunder ekstra hver dag, regnede jeg ud at hvis der var 1 million sekretærer, som brugte det system, ville fejlen (eller uhensigtsmæssigheden) med en timeløn på 130 kr. koste 600,000kr. - men nu var det nok snarere 20 millioner eller endda 200 millioner, som brugte systemet, så ... 120 millioner spildte kr.om dagen er dog en sjat, selv om det begynder med kun 3 øre i sekundet.

På en eller anden måde er det spændende at følge med i krigene omkring standarder. Det er forbløffende og heldigt, godt, at så mange forskellige mennesker og fagfolk kunne blive enige om at Unix ville være en fantastisk god økonomisk ting for IT-virksomhederne, og at man arbejdede sig frem til en specifikation af, hvordan sådan et system skulle se ud, - Posix specifikationen, som langsomt udviklede sig til det, den er i dag.

Det er 27 år siden ISO frigav de første dele af Posix, og IT-systemer har ændret sig fra systemer med én CPU til multi core, clusters, netfilsystemer, cloudløsninger, multilevel cache, NUMA, smartphones og så videre.

Muligvis skal løsningerne findes i en objekt-tankegang, hvor applikationer får operativsystemet til at skabe et objekt, som er sikret med krypteret token, og som medbringer sine egne rettigheder, ikke Linux-Capabilities, men Capsicum-capabilities, et project fra Cambridge Computer Laboratory, støttet af bl.a. Google - og Darpa, som også støttede udviklingen af internet protokollerne på Berkeley.

Men under objekterne vil der stadig være brug for håndtering af blok-devices på fornuftige måder. Derfor gætter andre forskere fra Unix-miljøet på at vi vil se flere og flere API libraries (shared) som tilbyder grundlæggende funktioner af generiske typer som fx. at skabe et transportabelt objekt med sit eget sæt af begrænsede rettigheder.

Ved at slå op i Wikipedia (se link nederst) kan man finde beskrivelser af eksperimentelle systemer baseret på capabilities. De ældste er fra 1970'erne, altså samme alder som Unix. Forskellen er jo bl.a. at Unix som grundlæggende telefon-switch orienteret system ikke havde brug for at afskærme den ene proces fra den anden.

Man finder morsomme navne som EROS, The Extremely Reliable Operating System, som blev begyndt udviklet i 1991. Det var ren forskning og kom aldrig i produktion eller drift.

http://www.capros.org/

Det er et Open Source projekt - Gnu General Public License og med base på SourceForge.net

Hydra fra Carnegie Mellon begyndte i 1971 - man arbejdede med mange PDP-11 processorer (skønt små set med nutidens øjne)

Dagens Microprocessorer

I dette nummer har David Askirk skrevet en lille artikel om Dragonboard 410c, - et board, som ville kunne være hovedingrediensen i en mobiltelefon, skønt denne nok ikke ville ende med at være smal og fin som de flotte smarphones.

Hvis nogen af læserne vil købe sådan en, så kan vi advare mod at bestille sandisk sammested - for de bliver shippet hver for sig med store forsendelsesomkostninger til følge!

Visioner for DKUUG

Foreningen har haft generalforsamling, og man takkede redaktionen for at lave blad, hvilket vi (pluralis majestatis) selvfølgelig er glade for. Imidlertid skal det ikke være nogen hemmelighed, at der skal nye kræfter til, gerne meget yngre, og det er en stor opgave, så det er nok ikke noget, der løses i ét hug.

Sagen er imidlertid, at det vil være trist, hvis ikke de ressourcer, foreningen råder over, bliver brugt til at markere Open Source miljøet i Danmark. At foreningens historie er broget, burde ikke være nogen forhindring, den nuværende bestyrelse og medlemmerne er positivt indstillet overfor unge projekter, og vil hellere end gerne støtte. Således har vi støttet både Bornhack og The Camp og vistnok et par stykker mere i den forgangne periode.

Med lidt kendskab til IT-drift kan man finde mange opgaver, som ville kunne løses ved klog programmering, sådan som Poul Henning Kamp for godt 12 år siden satte sig for at lave et OpenSource projekt, hvor han kunne drage fordel af sit kendskab til kerne-programmering; det blev til Varnish, en server cache, som muliggør at et dynamisk website minimerer svartider på forespørgsler, som gentages.

Vi har søgt på en applikation, hvor en system-chef kan tildele tidsbegrænsede root-passwords til medarbejdere, der skal bruge dem for en kortere periode - uden at skulle huske dem.

DKUUG bruger først og fremmest mail, men har behov for både en wiki og newsgrupper/mailgate. Videoer fra foredrag her http://video.dkuug.dk/ er også forældede - det er ikke godt.

Den nye bestyrelse vil lægge vægt på at komme ud til medlemmerne, og det glæder vi os alle sammen til.

Donald Axel

https://en.wikipedia.org/wiki/Capability-based_security#POSIX_vs._Capsicum_capabilities

DKuug-NYT er medlemsblad for DKuug, foreningen for Åbne Systemer og Internet
Nr. 182 - December 2016

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

Forsidecredits:
Redaktionen

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

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

Eftertryk i uddrag med kildeangivelse er tilladt.

Deadline for nr. 183: Fredag d. 10. februar 2017.

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:

Kalender:

OSD 2017 - se Opensourcedays.org - 18. Marts 2017 kl.10 - ... Måske også en virksomhedsdag om fredagen - følg med på websitet.

Foreningens kontor:

DKUUG, Symbion, Fruebjergvej 3

DK-2100 København Ø

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

VI modtager gerne indlæg til bladet.

DKUUG udgiver foreningsbladet DKUUG NYT 3 eller 4 gange om året - og oftere, hvis der er stof.

Redaktionen vil forsøge at arrangere en hands-on aften med programmering af en grafisk applikation med en eller to call-back funktioner - en GUI-Hello-World. Desuden - hvis der er interesse - en aften med shell-programmering for begyndere.

Onsdag d. 18. januar kl.18, DKuug, Symbion, kl. 18: Hands on på scripting

Onsdag d. 4. februar kl. 18, DKuug, Symbion, Hello-World i GUI.

Dragonboard-410c – simpel demo

Af David Askirk

I denne artikel vil der blive behandlet et Dragonboard 410c fra Qualcomm. Hvis navnet Qualcomm lyder bekendt, er det nok fordi du har set det før i forbindelse med med din mobil telefon. Qualcomm er stor producent af beregningsenheder til mobil telefoner. Et dragonboard er et udviklingsboard der har samme CPU som sidder i nogle mobil telefoner. I denne artikel kommer vi til at kigge på boardet for senere at kunne bruge det som en IoT (Internet of Things) enhed.

Udpakning

Et dragonboard har følgende i kassen når man åbner den første gang:

- Dragonboard

Det er jo ikke meget, og slet ikke nok hvis man vil igang, derfor skal man også have en PSU der kan levere 12V/2A. Den kan heldigvis købes samtidig med boardet. Boardet kan pt. kun købes i USA, så man skal være klar på også at betale moms samt afgift til PostNord.

Det kommer som udgangspunkt med Android, men der kan installeres andre systemer hvis man ønsker det. I denne artikel kommer vi til at installere debian. For at installere andre styresystemer skal man have et microusb kort, gerne på 16 gb eller større.

Det kan anbefales at købe et USB netkort, så man kan bruge trådet netværk på sit board.

Specs på board inkl GPIO

Et dragonboard har følgende hardware specs:

- CPU: Quad-core ARM Cortex A53 64-bit
- RAM: 1 GB
- Onboard Storage (eMMC):
- Forbindelsesmuligheder: Wifi, bluetooth, GPS (Der er on-board antenner til alt dette)
- Stik: HDMI, Micro USB, 2x USB A, micro SD

Samtidig er der også følgende mere hardware nære forbindelses muligheder:

- En 40-ports connector: 12x GPIO, UART, SPI, 2x I2C
- En 60-ports connector som bl.a. har I2C og USB

60-ports connectoren har også mulighed for at man kan sætte et display på. Så kan man dog ikke bruge HDMI porten samtidig.

Hvis man er kvik med loddekolben, er det muligt selv at lodde stik på til stereo lyd, både ind og ud.

Se mere på [1].

Angående GPIO porten er det vigtigt at vide at den kun kører på 1.8V som er meget fint til rent digital elektronik, men det er lavere end både Arduino og Raspberry Pi, som kører på 5V og 3.3V.

Install af OS

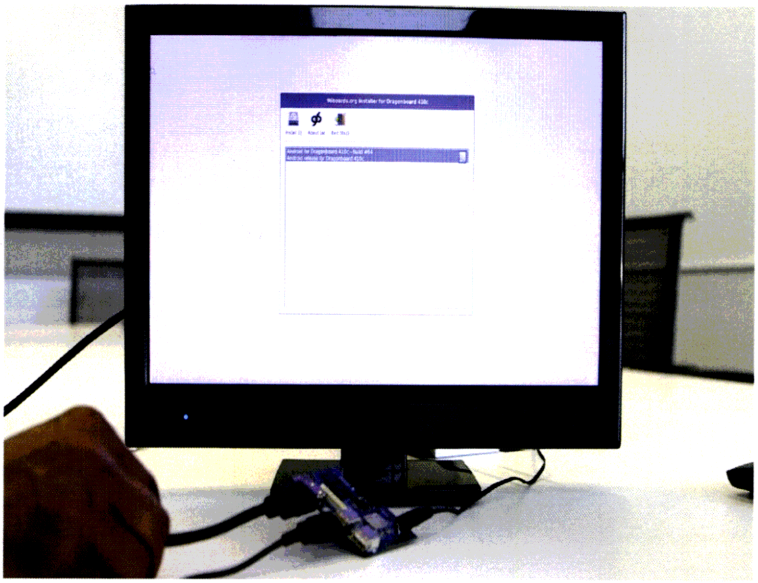
For at installere et andet OS kræves der et micro-SD kort, som bruges til at have install filerne. I dette tilfælde vil der blive brugt Debian.

Den nærmere guide er på [2], men de store træk er at:

1. Hent et debian image
2. Skriv filen med dd til micro-sd kortet
3. Set kortet i dragonboardet.
4. Sæt den til at boote fra SD kort
5. Start fra SD kortet og installer
6. Genstart og nu er der debian installeret

Processen er rimelig smertefri. Default brugeren er: linaro med kodeordet: linaro

Det er også muligt at installere Windows 10 IoT Core Edition, men det er en anden historie :-)



```
kchitrik-linux(63)> sudo dd if=db410c_sd_install_ubuntu.img of=/dev/sdf bs=2M
```

Kommando til at skrive install image på SD kortet

Opsætning af python

Python er heldigvis allerede installeret på det image der rullet ind. For at få et bibliotek der kan snakke med GPIO portene, kan det bibliotek der kan findes på [3] bruges. For at installere det skal følgende skridt tages:

```
git clone https://github.com/jackmitch/libsoc.git
cd libsoc
./autogen.sh && ./configure --enable-board="dragonboard410c"
make && sudo make install

git clone https://github.com/96boards/96BoardsGPIO.git
cd 96BoardsGPIO
./autogen.sh && ./configure
make && sudo make install

sudo ldconfig
```

Det vil give mulighed for at styre GPIO portene fra python.

Lav LED der blinker

For at bruge og vise brugen af GPIO portene skal der laves noget der kan blinke med en LED (lysdiode).

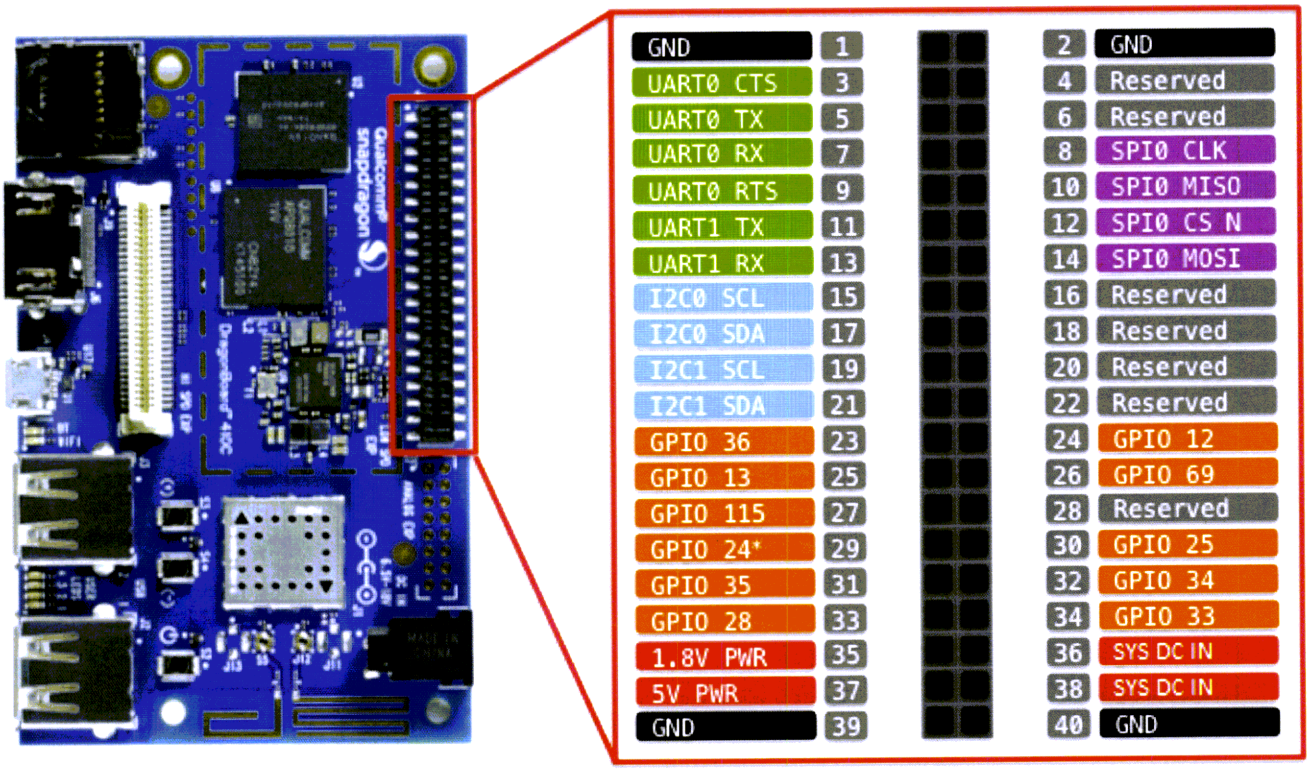
Dragonboard er i en familie af boards der hedder 96 boards. Det er forskellige boards der har de samme slags forbindelser. Derfor vil python biblioteket bruge nogle port mappings. De er som følger: ▸

➔

Pin mapping

```
GPIO-A = 36
GPIO-B = 12
GPIO-C = 13
GPIO-D = 69
GPIO-E = 115
GPIO-F = 4
GPIO-G = 24
GPIO-H = 25
GPIO-I = 35
GPIO-J = 34
GPIO-K = 28
GPIO-L = 33
```

Tallet ud for porten er pin-nummeret på selve CPU'en. Se på tabellen, hvordan de forskellige pins skal forbindes til 40-sokkelen på Dragonboardet:



For at få adgang til GPIO portene fra python skal følgende stump kode bruges:

```
from gpio_96boards import GPIO

GPIO_L = GPIO.gpio_id('GPIO_L')
pins = (
    (GPIO_L, 'out'),
)
```

Den importerer GPIO biblioteket, samt sætter GPIO_L op som output. L er på pin 33 på CPU'en, men er på pin 34 på selve bordet.

For at åbne GPIO porten, så den kan bruges, skal GPIO funktionen bruges. Den bruges således:

```
with GPIO(pins) as gpio:
```

Ved at bruge with bliver der også ryddet op efter koden er kørt, så den ikke står og hænger noget sted.

Så skal der blinkes. For at styre timingen skal python bibliotekets funktion *time* importeres.

Dernæst skal den blinke 10 gange med 500 ms tændt og 500 ms slukket.

For at skrive til en GPIO port bruge digital_write funktionen. Se følgende:

```
    for i in range(10):
        gpio.digital_write(GPIO_L, GPIO.HIGH)
        time.sleep(0.5)
        gpio.digital_write(GPIO_L, GPIO.LOW)
        time.sleep(0.5)
```

Dette giver en samlet kode listning som følger:

```
import time

from gpio_96boards import GPIO

GPIO_L = GPIO.gpio_id('GPIO_L')
pins = (
    (GPIO_L, 'out'),
)

with GPIO(pins) as gpio:
    for i in range(10):
        gpio.digital_write(GPIO_L, GPIO.HIGH)
        time.sleep(0.5)
        gpio.digital_write(GPIO_L, GPIO.LOW)
        time.sleep(0.5)
```

Dette er blot et simpelt eksempel, men ved at kombinere linux, python og elektronik kan man bygge kontakter, der styres fra internettet, eller hjem der styres fra en app eller noget helt tredje. Det spændende ved at bruge et Dragonboard frem for en Raspberry Pi er, at den hardware, som sidder på et Dragonboard, er den samme som sidder i mange mobil telefoner. Dette gør den både strømvenlig samt nem at udvikle til.

Links

- [1]: https://developer.qualcomm.com/hardware/dragonboard-410c
- [2]: https://github.com/96boards/documentation/wiki/Dragonboard-410c-Installation-Guide-for-Linux-and-Android#installing-image-using-an-sd-card-image
- [3]: https://github.com/96boards/96BoardsGPIO

Linux Kernel Development Report 2016

Hvordan kan 14.000 mennesker arbejde sammen på et softwareprojekt?

Samarbejde i denne størrelsesorden er aldrig set før . Det skriger på en forklaring. God udviklingsskik siger, at et delprojekt ikke skal være større end at 6 mand kan klare det på 6 uger. Det er selvfølgelig en tommelfingerregel - det er også OK med 10 mand i 10 uger, men så begynder rollerne at blive for mange og kommunikation mellem medlemmerne af projektet tager for lang tid. Det er rationalet bag begrænsning af antal deltagere og tidsrammer.

Men det forudsætter jo, at man kan opdele store projekter (som Linux kernen eller et Skattesystem) på en fornuftig måde. Det forudsætter, at der er en kompetent ledelse, som læser og giver kritiske kommentarer til programmeringen, peer review.

Måske er det et rygte, at der i de store danske IT-virksomheder kan sidde 20 mand ved et ledelsesmøde, hvoraf kun den ene er teknisk erfaren. Personligt har jeg oplevet en chef, der kom og spurgte "hvem ved egentlig noget om dette projekt", hvorefter den fastansatte overprogrammør i lokalet (hvor jeg var vikar) meldte sig. Nogle uger efter, da der skulle uddeles lokaler i et nyt domicil, fik han en plads i bunden - rent fysisk i kælderen.

Det gjorde ondt.

Men man må ikke generalisere ud fra én oplevelse.

Modularisering

I en tidligere artikel, DNyt-179, blev Accenture rapporterne om Skats fejlede systemer gennemgået, og det fremgik, at en funktionel opdeling i selvstændige, verificerbare projekter var mangelfuld. Man kunne ikke delteste moduler, og test med realistiske forhold (input/data osv) blev ikke afholdt.

Wikipedia har en artikel: *List of failed and overbudget custom software projects.* Heri optræder bl.a. Polsag med disse tal: Fra 2007 til 2012, uddelegeret til CSC, pris DKK 500m ($70m), technical problems with contractor. Den danske wikipedia fortæller mere om hvad Polsag skulle kunne og konkluderer: *En ekstern undersøgelse af systemet i efteråret 2011 var meget kritisk over for kvaliteten af CSC's arbejde med Polsag, og vurderingen lød, at det ville kræve meget store resurser at rette op på Polsag.*

Allerede i nullerne hørte vi om "billige IT-folk" fra fjerne lande i virksomheden CSC. Prosa, IT-fagets forening, måtte hjælpe de udenlandske arbejdere med at få de kontraktligt lovede betalinger. Det vigtige her er at det viser, at politikerne har den overbevisning, at man kan købe viden billigt uden at vide noget selv; det er en udløber af New Public Management. Sammenlign med VW-CO_2 - tidligere god track-record er ikke garanti mod svindel.

Drivkraften i Open Source

Mislykkede projekter og fører til frustration blandt gode programmører. Hvordan kan man bruge sin indsigt mere nyttigt, synes den fagligt kompetente at spørge.

Det, der sætter mange programmører igang med frivilligt arbejde i Open Source projekter er ***glæden ved at lave programkode af høj kvalitet.***

Kernen er et af de største Open Source projekter nogensinde

Linux mascotten TUX fylder 20 år

Udviklingen beskrives i Linux Foundations rapport, **Linux Kernel Development Report 2016.**

Den kan downloades (gratis) fra Linux Foundations websider; den er på 19 sider, klart skrevet. Denne artikel giver forhåbentlig lyst til at hente og læse den.

Arbejdsmodellen bliver her vist i tal: Datoer for opdateringer, antal forandringer (patches) for hver version, gennemsnitligt antal forandringer i timen, forklaring om vedligehold af en brugbar, stabil kerne, antal linier kode i kerne-source osv.

Det vigtigste i et regnskab eller en rapport som denne er formålsparagraffen (her: konklusionen) om hvad der er målet og hvordan man arbejder for at komme derhen.

En af de sidste nyheder er forresten, at Microsoft er blevet "Platin-medlem" af Linux Foundation:

> MICROSOFT FORTIFIES COMMITMENT TO OPEN SOURCE, BECOMES LINUX FOUNDATION PLATINUM MEMBER

I få ord

Siden 2005 har 14.000 bidraget til Linux-kernen. De mange udviklere kommer fra 1300 forskellige firmaer, som ellers i andre sammenhæng ligger i skarp konkurrence med hinanden.

Rapporten er den syvende af sin slags. De kommer med ca. et års mellemrum, den sidste i december 2014 efter kerne 3.18. Den foregående rapport blev omtalt i DNyt nr. 177 .

Linux kernen har 25 års jubilæum i år, det er bemærkelsesværdigt på et IT-marked, hvor dagens nyhed er forældet i morgen.

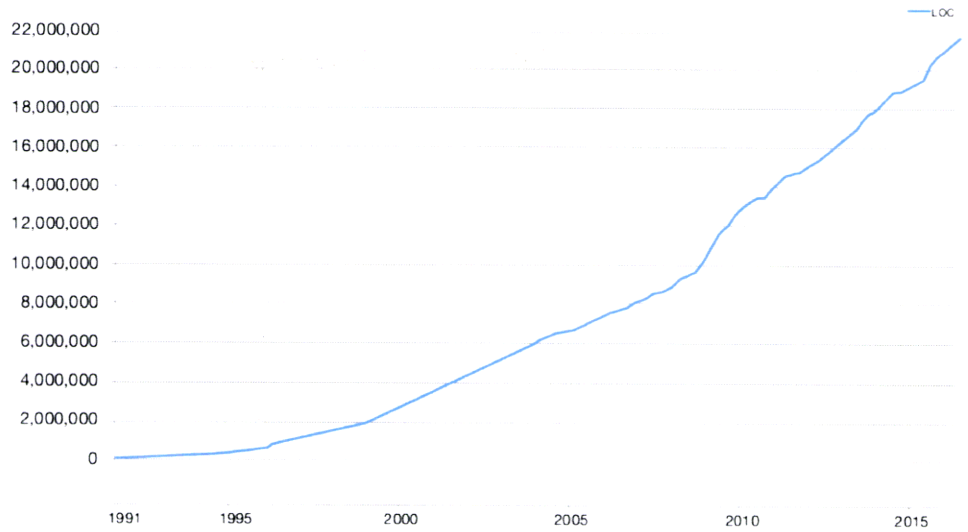
Der må altså være noget, som udviklerne gør rigtigt. Rapporten giver en opdatering af hvad udviklerne gør og hvorfor det går så godt.

Nyt i kernen

Der er så mange nyheder i kernen, at man med rette kan spørge: Hvordan kan det lade sig gøre at tilføje så mange nye features til et så gammelt system? Svaret er (selvfølgelig) at der er sket en del ændringer af det grundlæggende design siden 1991. ▸

The kernel has grown steadily since its first release in 1991, when there were only about 10,000 lines of code. At almost 22 million lines (up from nearly 19 million), the kernel is almost three million lines larger than it was at the time of the previous version of this paper. Another way of putting this it that, in the production of the 3.19 to 4.7 releases, the kernel community added nearly 11 files and 4,600 lines of code — every day.

Total Lines of Code in the Linux Kernel



De større ændringer i Linux var tidligere markeret af skift af major-versionsnummer. December 2014 havde vi version 3.18, rapporten slutter med 4.7 *[i skrivende stund har vi 4.8!]*

Den mest gennemgribende ændring er live update af kernen, d.v.s. at man på et kørende system kan foretage kritiske opdateringer uden reboot eller nedetid.

Kerneversion 4.7 har bedre support af permanent memory (SSD) og kan endvidere køre med krypteret ext4 filsystem.

Netfilter, pakkefilter, har inkorporeret Berkeley Packet Filter (BFP) for at man i kernen kan udvide med support af mange subsystemer, sikkerhedsmodul-stak osv.

Der er fokus på IPv6 og data-center forbedringer.

Mange forbedringer hænger sammen med, at Linux - ud over at være i brug på nogle desktoppe, er udbredt i de store datacentre, som står for de verdensomspændende ufatteligt store webservices, Google, Facebook, Instagram, Twitter osv (og der er mange flere).

Sikkerhed

Der er tilføjet support for et antal hardware baserede sikkerhedsfunktioner, så som Intels memory protection nøgler:

Hver memory blok, som bruges af et program, får et tal, en nøgle, og den tilhører programmet; ved hver memory access checkes, at programmet har denne nøgle. Det er en måde at sikre sig mod ondsindet kode og bufferoverflow exploits.

På de nye CPUer introducerer Intel også bound tabeller, som bruges når de (ligeledes forholdvis nye) bound registre ikke har plads nok. Ideen er at en pointers access kan kontrolleres med nedre og øvre grænse i et bound-register.

Der er fornyet interesse for mekanismer, som kan sikre kernen mod angribere, som ønsker at tage kontrol med maskinen, selv når der skulle være en sårbarhed, som kan udnyttes. fx Der er også support for ***ARM kør aldrig*** (ARM privileged execute-never) mekanisme, som blandt andet kan forhindre at applikationskode bliver kørt med root privilegier.

Byggeprocessen er forbedret. Det er stadig meningen, at man skal kunne bygge kernen uden større armbevægelser (selv om konfiguration af kernen efterhånden er sort magi for nogle områders vedkommende!)

Zero-day build and boot robot systemet har fundet 400 fejl i denne periode (og de er alle udbedret!) 🙂

Udviklerne kan, med andre ord, automatisere en test af systemet og få feedback med det samme. Det var samme mekanisme, som lå til grund for udbredelsen og popularieten af den første generation Unix fra Bell Labs, letheden ved udvikling og mulighed for hands on på hele kernen. Selv om man kun har et forslag til en lille ændring af et subsystem, så kan man (og kunne allerede i Unix' barndom) prøve det af ved at bygge kernen og køre en (automatiseret) test.

Udviklingsmodellen

Omkring 2005 ændrede arbejdet omkring kernen sig meget, dels var kernekoden vokset, og dels var der mange subsystemer, mange gode udviklere var kommet med og man havde gennem længere tid kunnet følge med på Kernel.org hvordan tingene foregik. Dels begyndte virksomheder at bruge Linux kernen i endnu højere grad - Google købte Android Inc. ▸

Google igangsatte udviklings arbejde af Android for mobiler, baseret baseret på Linux kernen. Google forestod et arbejde med at markedsføre Android som et flexibelt, opdaterbart system, som kunne bruges af (alle) handset-makers og telefon-fabrikanter, med besked om at den var åben for samarbejde fra alle sider.

Måske er det en del af forklaringen på, at arbejdet med kernen ændrede sig på dette tidspunkt. Mobil-enheder kan bygges af et væld af komponenter, og grafik og net-effektivitet er afgørende for salget.

Processen blev dengang, som i dag, baseret på major release, d.v.s en feature-ændrende release, med 8-12 ugers mellemrum.

Det er ikke uden grund at Linux Foundation med denne rapport sætter fokus på, hvordan det i det hele taget kan lade sig gøre.

Hvor hurtigt kommer nye ting med i den stabile kerne?

Rate of change, forandringshastighed:

Med mellemrum kunne man i de tidligere Linux-versioner se at Linus Torvalds havde forkastet forslag til drivere, som ellers var indsendt fra hæderkronede gode firmaer og som af udenforstående blev betragtet som bedre kode end den, der så blev godkendt og optaget først. Da Torvalds blev spurgt, svarede han at en kode-klump (patch) på 300 kb med massive problemstillinger simpelthen var for meget at læse.

Torvalds brugte stærke ord - det gjorde han i det hele taget, når noget forekom ham stupidt og hensynsløst. Måske troede indseren at Open Source er et magisk ord, som får alting til at fungere. Men det er jo så langt fra virkeligheden som man kan komme!

Men store programmer kan rumme mange, mange fejl som er svære at opdage. Sund udviklingspolitik kræver, at man har en slags terrasse-system, så man går fra funktionerende kode, mindre ændring - stadig funktionerende kode. Man må aldrig indsende en *patch,* som resulterer i broken code.

Sidst i 90'erne og først i nullerne begyndte Alan Cox at hjælpe med at modtage kode og give en første evaluering, - evt. nedbryde den i mindre enheder.

Torvalds om Alan Cox:

> *Note that nobody reads every post in linux-kernel. In fact, nobody who expects to have time left over to actually do any real kernel work will read even half. Except Alan Cox, but he's actually not human,* ***but about a thousand gnomes working in under-ground caves in Swansea****. None of the individual gnomes read all the postings either, they just work together really well.*

(Redaktionens fremhævelser) Alan Cox har blandt andet arbejdet på open source Multi-User Dungeons:

> *AberMUD was the first popular open source MUD. It was named after the town Aberystwyth, in which it was written. The first version was written in B by* ***Alan Cox****, Richard Acott, Jim Finnis, and Leon Thrane based at University of Wales, Aberystwyth for an old Honeywell mainframe and opened in 1987.*

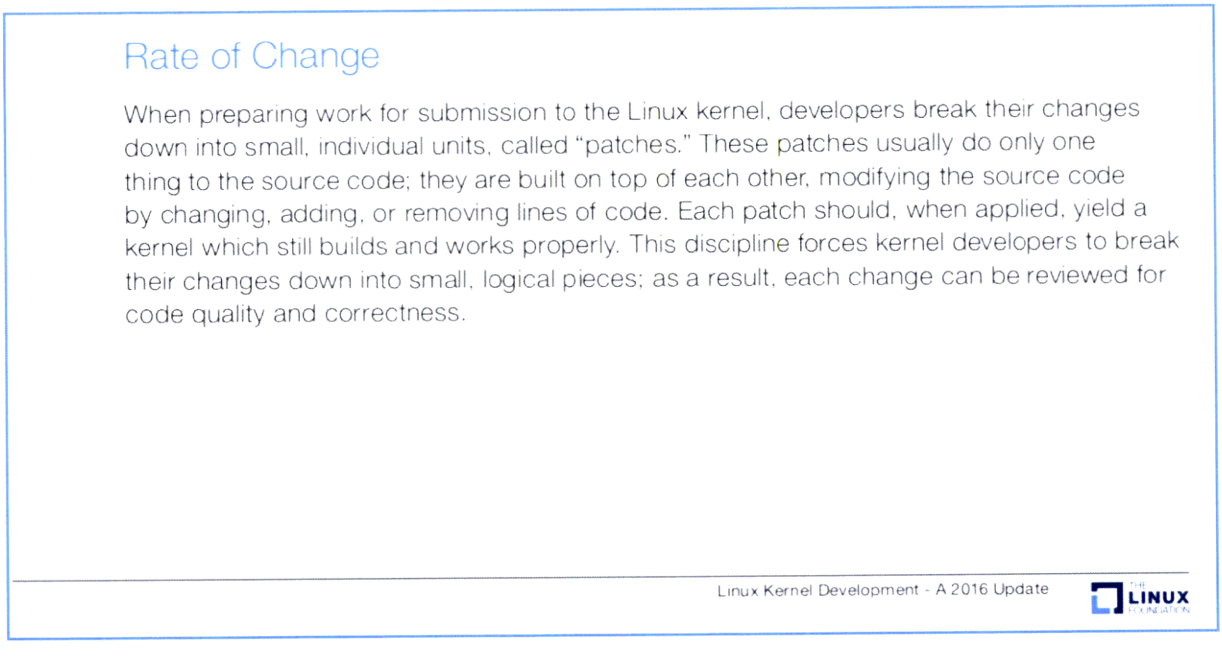
Resultatet er at det lykkes at indføje patches i kernen med stor hast og at kernen derfor tilbyder nyeste features for hardware produkter. Hastighed i sig selv er ikke et mål, men fleksibilitet og tilpasning til markedet er.

I perioden med release 3.19 til 4.7 har antal ændringer i timen været omkring ***8 i timen.***

Tallet underdriver den faktiske aktivitet, mange patches gennemløber flere revisioner og nogle bliver aldrig optaget i kernen.

Evnen til vedblivende at holde denne forandrings-hasttighed er uden sidestykke i noget tidligere offentligt IT-projekt.

➔↗



Opdateringer af den stabile kerne

Hver release er en major release med gennemgribende ændringer. Alle skal være stabile, *stable releases*. Det er ikke som i tidlige Linux kerner at 2.0 var en major release, 2.1 var en ustabil eller forsøgsvis kerne med nye features; ved release 2.2 blev de nye features så anerkendt som stabile.

Det er uundgåeligt, at der vil være problemer i releasede kerner, og patches bliver så lavet for at rette de fejl.

Opdatering af den stabile kerne foregår på den måde, at man både opdaterer den stabile release og den kerne, som der arbejdes på, *mainline kernel*. Den stabile kerne er den kode, som de fleste distributører bruger som basis for kernen.

Normal policy for stabil release er, at hver kerneversion vil blive opdateret i minimum én development cyklus, fx. 8 uger for kerne 4.5 fra marts 2016 til midten af maj 2016, hvor kerne 4.6 blev releaset.

Cirka en gang om året er der en kernerelease, som bliver valgt til langtids-support, fx. kerne 3.14, som bliver opdateret i to år.

Der er også langtids-kerner udenfor den normale stable-proces, fx. har Debian udvikler Ben Hutchings vedligeholdt en kerne 3.2 med 81 releases som omfatter 7.072 fix!

Jamen er det ikke dårlig kode, når man skal lappe på den?

Man skulle tro, at når kernen var stabil og "færdig", så var der ikke grund til at rette og komme med sikkerhedsopdateringer.

Men der er en forklaring: Ikke alle udviklere har adgang til alle de devices, som kernen supporterer, og belastning af en server kan variere og skabe problemer, som udviklerne ikke så let kan se.

Trods den forbedrede bygge- og test-proces er Linux-community stadig afhængigt at sine brugere og tilbage-meldinger om problemer. Vedligehold af en stabil kerne er godt, fordi releasede kerner kan få færdige fix (rettelser, som er afprøvede og har været igennem review) mens udviklerne arbejder videre med nye opgaver.

Hvem bidrager til kernen?

Det er en rekord at så mange som 251 virksomheder kom med bidrag - patches - til Linux-kernen. Men trods det store antal udviklere, som bidrager, er det trods alt et fåtal, som kommer med de fleste kode-filer/patches.

Top 10 bidragsydere har skrevet 42.344 ændringer, 7.5% af total antal ændringer, siden release 2.6.11 i 2005, hvor man begyndte at bruge git-repositoriet.

Hvem sponsorerer?

Mange sponsorer er ikke udviklere og er glade for at kunne støtte udviklingen og bruge produktet.

Og mange udviklere giver rettelser og udvikler uden at få penge for det. Linux Foundation har lavet statistik på det og konstaterer at antallet af kompetente kerneudviklere ikke er så stort, - det er en grund til bekymring.

Men Linux Foundation konstaterer også, at en kerneudvikler hurtigt får job i firmaer, som bruger Linux og som derfor gerne vil have kompetence in-house.

Hvordan kommer kode med i kernen?

Kode kommer ikke direkte i kerne-repositoriet, men går via en af 100 subsystem-"træer", som administreres af en programmør, som vedligeholder dette system (fx. SCSI drivere) - en ***maintainer*** (se filen MAINTAINERS).

Når en maintainer godkender en patch, skriver han en sign-off. Med Git systemet kan man lave statistik på det. Torvalds optræder ikke blandt to30 i den statistik (i tidligere versioner af kernen kom al kode gennem hans kontor), hans arbejdsområde er at få organisationen til at fungere.

Konklusionen er, at Linux-udvikling går forrygende hurtigt, at den udfylder mange behov, og at det fungerer godt trods hastigheden og omfanget.

Dette Open Source projekt burde være et oplagt emne for et studie i moderne ledelse.

■

Those developers are:

Name	Changes	Percent
H Hartley Sweeten	5,960	1.1%
Al Viro	5,433	1.0%
Takashi Iwai	4,723	0.8%
Mark Brown	3,960	0.7%
David S. Miller	3,950	0.7%
Mauro Carvalho Chehab	3,943	0.7%
Tejun Heo	3,852	0.7%
Johannes Berg	3,707	0.7%
Russell King	3,467	0.6%
Thomas Gleixner	3,233	0.6%
Hans Verkuil	3,119	0.6%
Greg Kroah-Hartman	3,117	0.6%
Ingo Molnar	2,873	0.5%
Joe Perches	2,778	0.5%
Christoph Hellwig	2,697	0.5%

Name	Changes	Percent
Eric Dumazet	2,633	0.5%
Axel Lin	2,604	0.5%
Dan Carpenter	2,562	0.5%
Geert Uytterhoeven	2,460	0.4%
Laurent Pinchart	2,381	0.4%
Alex Deucher	2,340	0.4%
Bartlomiej Zolnierkiewicz	2,279	0.4%
Trond Myklebust	2,269	0.4%
Paul Mundt	2,268	0.4%
Daniel Vetter	2,224	0.4%
Ben Skeggs	2,216	0.4%
Arnd Bergmann	2,199	0.4%
Lars-Peter Clausen	2,176	0.4%
Arnaldo Carvalho de Melo	2,107	0.4%
Ralf Baechle	2,097	0.4%

The above numbers are drawn from the entire Git repository history, starting with 2.6.12 in 2005.

Er Posix forældet?

Har Posix standarden ligefrem udspillet sin rolle?

af redaktionen

Posix er akronym for Portable Operating System Interface. Det er mere end 25 år siden, at de første dele af Posix blev sat sammen af dele hentet fra de forskellige Unix-versioner.

Det er derfor oplagt at mistænke den for at være helt unyttig, ubrugelig. På den anden side er hjulet jo ikke gammeldags, selv om man har opfundet bedre kuglelejer og bedre dæk. Det er mere gavnligt at betragte Posix som en basis-idé, som skal revideres.

Posix - og før den Unix - var et fremskridt for især brugere og programmører: Hvis man kunne programmere til flere maskintyper sparede man tid og kræfter og kunne koncentrere sig om løsningens kvalitet: Skriv én gang, kør mange gange, på mange systemer.

Unix beskriver en multi-user arkitektur

En computer var, dengang Unix blev skabt og senere bredte sig som en løbeild, enten en PC eller en skærme (skærmterminaler) tilsluttet en flerbrugermaskine (en såkaldt minicomputer eller mainframe). Set med nutidens øjne var det små maskiner, men de gjorde stor nytte.

Internettet var tilgængeligt fra sådanne terminaler, hvis centralenheden havde forbindelse. hovedsageligt fra universiteter. Tyskland fik forbindelse til Nordunet (Nordisk Universitetsnet) i Stockholm, og DKUUG kom med på den forbindelse og drev netforbindelser i 1989.

DKUUG var først i Danmark med at give adgang til Internettet for firmaer og private.

Man kunne hente og sende filer (mail, FTP) foretage login og bruge programmer på den centrale server.

Alt dette er utænkeligt uden en standard for både bogstaver, net-data, filnavne, og programmer til at læse filer og vise dem, bearbejde data osv. så derfor fik Posix mere og mere anseelse. US-Defense og andre statsenheder forlangte, at leverandører skulle overholde Posix - standarden.

Posix kom ikke ud af den blå luft

Forud for Posix var der dannet grupper, X/Open, OSF og flere andre, som arbejdede med specifikation af en fælles fornuftig version af Unix. *US Army Ballistic Research Laboratory* viste at man godt kunne køre alle programmer fra én rivaliserende gruppe på den anden gruppes kerne.

For at få styr på rivalerne søgte man ind i Den Internationale Standard Organisation (ISO), som dannede en fælles komite. Alle virksomheder og forskere kan deltage i ISO (selv om man ikke lige kan gå ind fra gaden). Sammen med *IEEE Computer Society* dannede ISO Joint Technical Committee - nummer et, *JTC-1*.

I-triple-E står for Institute of Electrical and Electronics Engineers, en US baseret virksomhed, som har 200 afdelinger verden over.

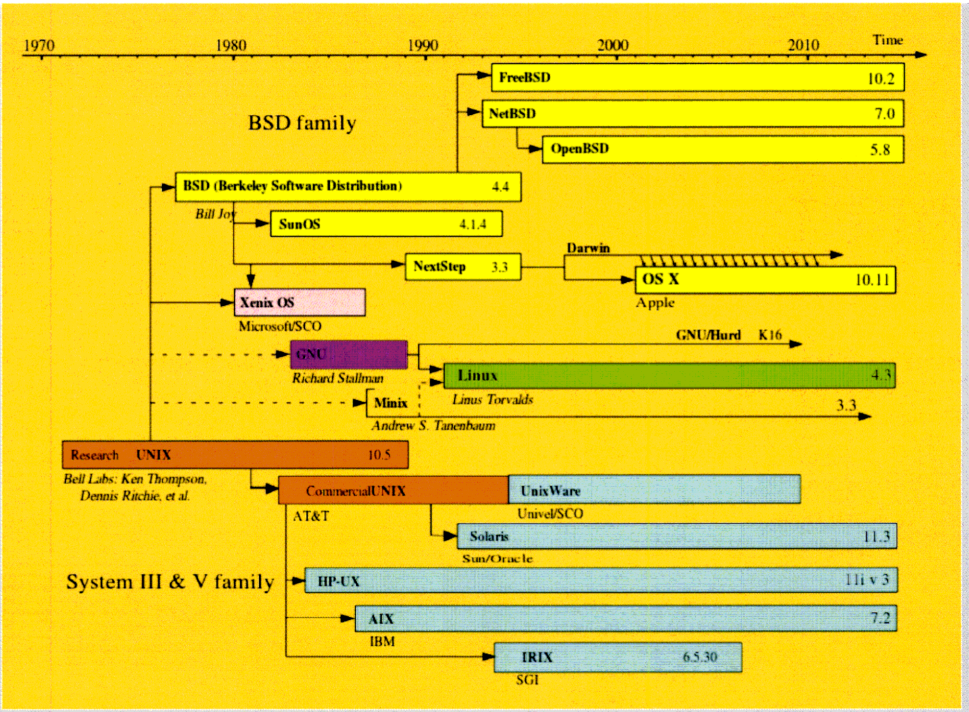
ISO repræsenterer med andre ord bredt; og JTC-1 brugte det eksisterende arbejde. System V Revision 4 (SVR4) var basis, og denne Unix var netop resultat af samarbejde mellem en række producenter - ***ellers kunne Posix ikke være lykkedes.***

Det blev til ISO IEEE Std 1003.1-1988, fra 1988, og den fik tilnavnet Posix - så var det nemmere at huske, og man kunne se, at det havde noget at gøre med Unix. (ISO-1001 er en specifikation for tape-filsystem og filnavne, ISO-1002 er, så vidt redaktionen kan se, en standard for kuglelejer.)

Senere er der kommet en del opdateringer (se fx. artiklen om Posix i Wikipedia).

Med ISO stempel var Unix i form af Posix endelig en af de vigtigste spillere på markedet. Kernen i Windows, NT, har også API som svarer til Posix.

Ikke desto mindre forandrer kravene sig, mens Posix står stille. De nye features, som fx. X-Window til Unix, en basis som kan bruges til mange grafiske user-interfaces (Gui'er) får hele tiden



nye features, og der ændres lidt hist og her eller kommer nye ting til.

Udviklingen af kernens API ser man bedst ved at tælle, hvor mange systemkald, der er til en kerne version 2-3-4 (samme gælder MS Windows). Det er forholdsvis let i Linux og BSD at se, hvor mange systemkald der er.

```
        Linux kernel release 4.x <http://kernel.org/>

These are the release notes for Linux version 4.  Read them carefully,
as they tell you what this is all about, explain how to install the
kernel, and what to do if something goes wrong.

WHAT IS LINUX?

  Linux is a clone of the operating system Unix, written from scratch by
  Linus Torvalds with assistance from a loosely-knit team of hackers across
  the Net. It aims towards POSIX and Single UNIX Specification compliance.
```

Fortjenstmedalje!

Posix har været basis for udviklingen af Linux, som i dag er kernen i millioner af routere og fladskærms-TV, mobiltelefoner og så videre. Alene af den grund er det super-interessant at undersøge standarden og finde ud af, hvad kritikken går på og om der er bedre løsninger til fx. netværksprogrammering, som bør inkorporeres i Posix-standarden.

Opdatering af Posix

Posix er skam opdateret mange gange.

> *As of 2016, Base Specifications, Issue 7 (or IEEE Std 1003.1, 2013 edition) represents the current version. A free online copy is available.*

Som med andre standarder er der ikke nogen idé i at ændre den hele tiden, så man kan såmænd godt genkende de fleste af tingene fra de tidligste versioner.

Vi må se på et eksempel senere, men først må vi lige have et overblik.

Der er fire hoveddele:

XBD, *Unix Base definitioner*, beskriver det særlige standardsprog (*shall* & *can* og den slags), for at man kan være sikker på, hvad der menes, når der fx. står:

> *The fsync() function shall request that all data for the open file descriptor named by fildes is to be transferred to the storage device associated with the file described by fildes. The nature of the transfer is implementation-defined.* [...]

XSH, *Unix System Interface & Headers,* API.

XCU, *Unix Command Utilities,* kommandolinie værktøjer.

XRAT, *Rationale* eller begrundelser for at tingene er blevet formuleret og besluttet, sådan som de nu står.

Net og skærmstyring, cursor-adressering

Den del af netværk som drejer sig om ledninger, elektriske signaler og transport, hører naturligt under IEEE - det skal nævnes at det er Ethernet som er udbredt og *token ring,* som var et alternativ i de tidlige netværksdage, er glemt og borte.

I operativsystemet skal tilgangen til netværk ske gennem et API og dette hører ikke ind under det "gamle" Posix, men er (selvfølgelig) med i de nye. En socket, på dansk en sokkel, er et endepunkt for en forbindelse - ligesom man pløkker et jackstik i en guitar-sokkel så bruger man en *"sokkel"* på computer-systemet til at være forbindelses-punkt til en netledning, som helst skulle have et andet endepunkt på en anden computer.

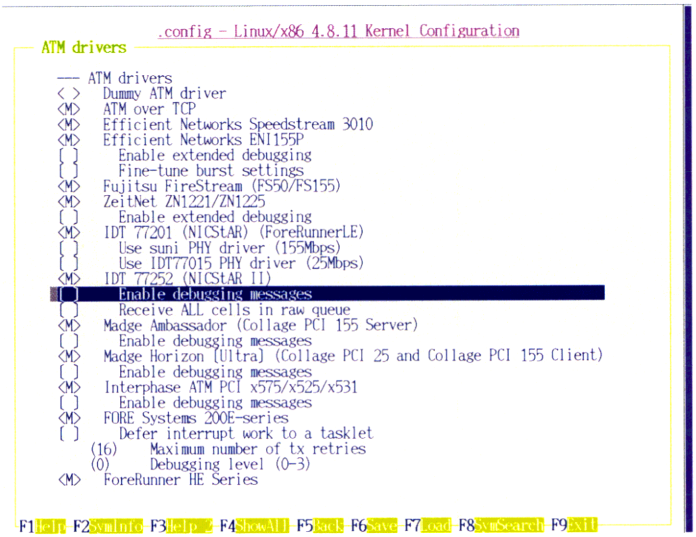
Med socket(2) får man skabt en sådan sokkel, og med bind forbinder man den med IP adressen (eller andre adressefamilier).

Lad os her nøjes med at konstatere, at netværk systemkald er med i de nyeste versioner af Posix. Netværksdelen i Unix/Posix stammer fra BSD, første gang i 4.2BSD.

Desuden er der en sektion om skærmstyring for tegnbaserede vinduer/terminaler, XCURSES, *X/Open Curses*, (det bruges i Xterm, gterm, xvt og hvad de nu hedder, varianterne af kommando-linie vinduer.)

På Linux fås denne skærmstyring ved installation af ncurses5.

Her er et eksempel på en menustyring til konfiguration af Linux-kernen, den fås ved at hente source, udpakke, og cd top-dir og så køre kommandoen *make nconfig:*



Udenfor Posix:

- Graphics interfaces
- Database management system interfaces
- Record I/O hensyn
- Binær portabilitet
- System konfiguration
- Pakke management

- Der er flere ting, som enten ikke er med i Posix eller som er bortfaldet (fx. ulimit(3)).

→→→

En af de ting, som Posix minimerede, var krav til opsætning af filsystem. I basedefinitionerne foreskrives:

Filsystemets rod betegnes med *"slash"* **/**

/dev Et device directory.

/dev/console, **/dev/null**, og **/dev/tty** tilgang til skærm/tastatur.

/tmp Til midlertidige filer.

At forskrifterne er så få, har bevirket at Linux distributioner og andre Unix-varianter har placeret programmer og shared libraries mv. på lidt forskellige steder i filsystemet, og at der er udviklet distributions-specifikke *stier* (paths) for placering af programmoduler og lignende.

Derfor kom ***File Hierarchy Standard,*** FHS, som vedligeholdes af Linux Foundation, og hvortil der henvises fra en anden standard, Linux Standard Base, LSB, som omfatter flere andre standarder - herunder det meste fra Posix.

De vigtigste filtræer

De vigtigste forskelle er i /etc og /var. Etc lyder selvfølgelig som om det var en morsomhed - her lægger vi dit og dat etcetera etc.etc. men det kunne nu lige så godt være ***editable text configurationsfiles*** - editerbare konfigurationsfiler, eller extended tool chest og lign.

Desværre fandt man somme tider binaries (programmer) og symbolske links til noget udenfor /etc filtræet, hvilket vanskeliggør brugen af /etc/* som essensen af et systems opsætning.

Alle distributioner bruger /usr, /usr/bin, /usr/lib, /usr/local, /usr/sbin, /var, /var/lib, /var/log osv. men derefter kan der være forskelle, som mere eller mindre forhindrer en applikation i at finde ressourcer.

LSB stræber efter at specificere systemer på en sådan måde, at hvis et system overholder LSB kan man køre programmer, der distribueres i binær form.

Det tager noget af arbejdet fra Linux-distributioner, men der er så meget andet, de enkelte distro'er kan profilere sig på.

Men disse tilskud til Posix er - strengt taget - bagateller, lad os lige se på den kritik af Posix, som går på at selve miljøet, som Posix er opstået til, er forældet. Altså, at det med en central computer, multiuser operativsystem ikke opfylder kravene i en tid, hvor man bruge smartphones, gemmer ting i skyen og ser på hinanden med et betydningsfuldt blik når man fortæller, hvad man nu kan med den.

LSB registreret som ISO standard

Linux sigter mod Posix compliance, som der står i README for kernel source base directory.

LSB, derimod, skønt stadig baseret på Posix, sigtede i første omgang mod at specificere alt det, som Posix manglede, og løse de konflikter, der er opstået ved at Linux har udviklet sig; - som eksempel kan nævnes at skønt gets(3) var en første-klasse funktion i Posix indtil 2008; den tillader stack-overflow og bør derfor ikke bruges. Ved hands-on for begyndere er gets(3) en god lærestreg, og der er jo sådan set ikke noget i vejen for at man har den som eksempel på design-fejl.

LSB var derfor vigtig, og den interesserede derfor ISO komiteen SC22 (Subkomite nr 22, Sub Committee 22 for operativsystemer og programmeringssprog i JTC-1).

LSB påvirker også Posix (ses bl.a. ved at gets(3) er bortfaldet).

Nu skal det imidlertid først og fremmest handle om Posix:

1. hvad er det, der bliver kritiseret, og

2. hvor meget af den er i brug.

```
011 Date: 28 July 2003


012 Introduction:


013 /* Notice on Draft:
014  * This is a draft based on the current LSB 1.8 draft in progress.
015  * When LSB 1.9 is available this document will be reworked.
016  */


017 1.1 Purpose


018 The purpose of this Type 3 Technical Report (informative) is to document
019 the areas of conflict between ISO/IEC 9945 (POSIX) and the Free Standards
020 Group's Linux Standard Base specification such that it can be utilized
021 by the appropriate technical committees when considering harmonization
022 between the standards efforts.


023 ISO/IEC 9945 (POSIX) is an important standard in use throughout the world.
024 There is a significant investment in applications developed for the ISO
025 POSIX standard. With the emergence of a standardization initiative for
026 the Linux operating system there are some areas of conflict that have
027 been identified between the Linux Standard Base specification and the
028 ISO POSIX standards.  There is an essential market requirement that the
029 conflicts be resolved so that an application can be written to conform
030 to both standards.  Hundreds of millions of dollars of applications are
031 built upon these standards. This report is intended as  a starting point
032 to look at resolution of this issue.
```

Sammenligner man fsync(2) specifikation fra 2000 med den fra edition 2016, opdager man at der ikke er ændret et ord. Specifikationen for fsync() forklarer at funktionen er at tvinge systemet til at opdatere en permanent storage, typisk en disk, således at man er sikker på at have data også efter et eventuelt crash.

Denne problematik er - for udenforstående - temmelig kompliceret, men selv en almindelig programmør kan forestille sig den situation, at man ønsker at registrere en betaling fra en kunde og inden systemet har skrevet den i konto-data, så er strømmen gået og systemet svarer ikke. Værst: Vi ved ikke om systemet nåede at skrive den helt eller halvt.

Hvis fsync() kommer tilbage til applikationen med besked om at nu er disk opdateret med filens data, så kan programmet med andre ord på brugerniveau gå videre med en saldo, som er gyldig for næste transaktion.

Men Posix "fædrene" har skam tænkt længere end til blot *"det ville nu være rart hvis ..."*

Forklaringen til fsync() siger bl.a.:

> *[...] It is explicitly intended that a null implementation is permitted. This could be valid in the case where the system cannot assure non-volatile storage under any circumstances or when the system is highly fault-tolerant and the functionality is not required. In the middle ground between these extremes, fsync() might or might not actually cause data to be written where it is safe from a power failure.*

Med andre ord, man kan ikke nøjes med at bruge fsync() for at sikre sine databasers integritet, selv om det var en oplagt ting.

Men det bliver værre: En database af de store vil bruge flere maskiner, parallelle CPU'er osv og der kan være en hel del konflikter, som bare ikke kan løses ved at bruge fsync().

Databasen PostgreSQL understreger at en konsistent tilstand af basen ikke er nødvendig for recovery. Ved at bruge en WAL (Write Ahead Log) kan man komme langt i retning af at have de sidste ændringer i basen med, skulle strøm eller maskineri fejle, bedst selvfølgelig ved continuous archiving på et andet site med nødstrøm eller lign. Ved extern logning af requests *inden* man sender dem til DB-systemerne kan man komme så langt som det er muligt med at sikre sig det hele kan reddes i tilfælde af brand eller oversvømmelse og lignende.

Men - kan man spørge - hvad skal man så med fsync() i en standard-specifikation?

pl.atyp.us om fsync()

Jeff Darcy skriver i et blog-indlæg *Updating POSIX* at alle ved at fsync() kan forårsage store forsinkelser - latency-bubbles.

Fsync

> *Everyone seems to know about one problem with fsync - that it can create huge latency bubbles. However, I see that problem as only the visible tip of the crapberg that is fsync, O_SYNC and all of their friends. In a way, it's really a side effect of the most fundamental problem with POSIX - that it's clueless about the relationship between consistency, durability, and ordering.*

Latens-problematikken er imidlertid kun toppen af isbjerget, det generelle problem er ikke forsinkelse, men at Posix ikke har noget begreb om konsistens, holdbarhed og rækkefølge *(consistency, durability and ordering).*

Konsistens er fremhævet i Posix, når én bruger har skrevet noget til en fil, skal det strax være til rådighed for en anden bruger, som vil læse filen (eller skrive videre i den).

Det var ikke så svært for en single-CPU maskine i 80'erne, idét man ikke behøver at skrive til disk for at andre processer kan læse fra den buffer, som første bruger har skrevet til.

På et moderne distribueret filsystem (netfilsystem eller filserver) er det knap så let. Selv med et lokalt filsystem kan det være et problem på en moderne centralenhed, multicore, multicache. Et kald til fsync(2) kan i værste fald komme til at gigabyte af memory skrives til disk + en rundtur for advisering om ændring af status på disk/fil.

For *Non Uniform Memory Access* (NUMA) systemer kan *fsync()* give stor belastning, fordi der er flere memory-niveauer (cache levels) og hver processor skal omkring alle de andre i systemet for at kunne give den slags garanti.

Andre funktioner med samme problem

Har man tygget lidt på denne problematik, ses det jo hurtigt at alle filsystem-modifikationer kan give problemer: rename(), readdir(), chmod() (fil-attribut ændringer) har samme problematik som fsync() og sync().

Løsninger

En erfaren programmør vil gå udenom de problemer ved hjælp af interproces kommunikation mv. Mange applikationer bliver tvunget ud i at bruge databasesystemer i stedet for "de løse" datafiler. Filsystemer bevæger sig i retning af at blive håndteret som database-systemer.

Objekter, Capsicum

University of Cambridge Computer Laboratory har udviklet et *letvægts* capabilities system på operativsystem-niveau. Det er et projekt, som er støttet af Google, FreeBSD foundation og Darpa. Capsicum udvider Posix API og giver derved OS-funktioner, som muliggør objekt-håndtering af sikkerhed. Tænk på det som et objekt, som er kryptografisk sikret (umulig at genskabe/kopiere). Objektet bruges via et "håndtag". Det medbringer egne rettigheder på systemet (en "nøgle") til fx. netaccess, og modtageprocessen får disse rettigheder. I capsicum modus kan processer ikke selv søge i globale namespaces. Metoden er ikke ny. Nedenunder er stadig et OS.

Har Posix udspillet sin rolle?

Posix har været en milliongevinst for alle - men er blevet ældre. Posix er både en idé og et konkret arbejde. En del af Posix er kodificering af operationer på abstraktions-laget ***"filsystem"*** ovenpå fysiske enheder som harddiske. Det er opstået i en tid med andre IT-miljøer end vi har i dag, og bærer præg af det.

Imidlertid var man faktisk allerede i 1990 klar over, at sync og fsync() var problematiske, og derfor skrev man i Posix følgende om fsync:

It is explicitly intended that a null implementation is permitted.

Hvilke systemkald i Posix bliver mest brugt?

Script, som registrerer systemkald - og script som tæller op

Man skal nok være lidt nørd for at begynde at se ned i applikationernes motorrum - på den anden side, hvis man vil programmere enten for at leve af det eller for at forstå det bedre, så er her nogle små scripts, som viser lidt om, hvilke funktioner der kaldes af et program.

Artiklen forudsætter en smule kendskab til kommandolinien - disse programmer kan dog også køres i en (Unix-)editor eller som debugging-funktioner i nogle udviklings værktøjer, (Integrated Development Environment, IDE).

Først en forklaring af det nyttige program ***strace:***

Strace er et nyttigt værktøj til diagnose, kurser, læring, og debugging.

Programmet strace skal have et program, som skal følges (traces). For ikke at få skærmen overfyldt med output fra programmet blandet sammen med output fra strace, bruger man option -o <filnavn> og deri samles informationerne om systemkaldene.

Der sættes en fælde eller et stop ved systemkald, ligesom ved debugging, og med systemkaldet ptrace() får man de relevante oplysninger (ved at få indholdet af CPU registrene på dette tidspunkt).

Et eksempel:

```
# strace -o /tmp/fox-log.txt /usr/local/firefox/firefox
```

Firefox vindue åbner sig, - for ikke at lave for meget, lukker vi det strax igen.

Man skal indtaste det, der står med bold. Hvis et program-eksempel kommer med output, er det tyndere (ikke bold).

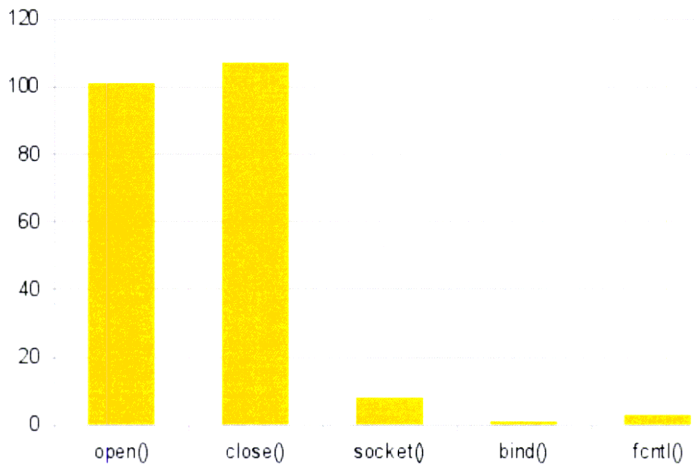
Hvad står der så i denne fil?

```
execve("/usr/local/firefox/firefox", ["/usr/local/firefox/firefox"], [/* 68 vars */]) = 0
brk(0)                                  = 0x9063000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xf7743000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=116227, ...}) = 0
mmap2(NULL, 116227, PROT_READ, MAP_PRIVATE, 3, 0) = 0xf7726000
close(3)                                = 0
```

Den første linie fortæller, at programmet firefox (i directory /usr/local/firefox) exekveres.

Hver linie viser et funktionskald, parentes med argumenter og til slut en retur-værdi, d.v.s. den værdi, som OS giver tilbage som fejlkode eller oplysning om hvordan det nu er gået. Linie 6, **`open("/etc/ld.so.cache"...)`** får returværdien 3, som i dette tilfælde er en fildescriptor, et lille tal, som er index til data med oplysninger om filen. Fra andre gange (eller vores Unix-kursus) ved vi at stdin, stdout og stderr er henholdsvis fd=0,1 og 2, så 3 er den laveste værdi for et program, der ikke smider 0,1 og 2 overbord.

Lidt længere nede, linie 9, ser vi at programmet *lukker (close())* fildescriptor 3 og får 0 tilbage, - det er en status, som betyder at der ikke var nogen fejl, filen kunne lukkes.



For at få at vide hvor mange systemkald, der er kørt, bruger vi grep. Option -wc (eller -w -c) betyder tæl kun med, hvis det er hele ord (med andre ord lad være med at tælle *xopen* som *open*).

```
# grep -wc open /tmp/fox-log.txt
101
# grep -wc close /tmp/fox-log.txt
107
# grep -wc socket /tmp/fox-log.txt
8
# grep -wc bind /tmp/fox-log.txt
1
# grep -wc fcntl /tmp/fox-log.txt
3
```

Tre IT-specialister har udviklet en mere avanceret metode at spore kald, ***libtrack.***

De tre specialister er Vaggelis Atlidakis, Ph.D studerende i Department of Computer Science ved Columbia Universitet, Ph.D Jeremy Andrus, og Roxana Geambasu, også Columbia.

Libtrack tracer ikke blot én applikation, men alle processer, tråde, i systemet. Der installeres en wrapper for hver Posix funktion med det samme navn, og når en native Posix funktion kaldes, logger *libtrack* tidspunkt, og lave backtrace til applikationen, som foretog kaldet (så man får navnet). Det er en dynamisk test.

Libtrack projektet har også en statisk del, som simpelthen identificerer og tæller de link-entries til standard libraries, som en applikation har. Hvis det lyder mystisk, så prøv følgende kommando på et simpelt (lille) program:

```
# objdump -j .text -D -G /bin/date | grep printf | head

80493a9:       e8 f2 56 00 00          call   804eaa0 <__sprintf_chk@plt+0x5720>
80493e9:       e8 b2 8c 00 00          call   80520a0 <__sprintf_chk@plt+0x8d20>
8049435:       e8 c6 86 00 00          call   8051b00 <__sprintf_chk@plt+0x8780>
804943d:       0f 84 05 02 00 00       je     8049648 <__sprintf_chk@plt+0x2c8>
8049446:       0f 84 ac 01 00 00       je     80495f8 <__sprintf_chk@plt+0x278>
8049450:       7f 2e                   jg     8049480 <__sprintf_chk@plt+0x100>
8049457:       0f 84 be 01 00 00       je     804961b <__sprintf_chk@plt+0x29b>
8049460:       0f 8e d8 00 00 00       jle    804953e <__sprintf_chk@plt+0x1be>
8049469:       0f 84 35 01 00 00       je     80495a4 <__sprintf_chk@plt+0x224>
8049472:       75 2b                   jne    804949f <__sprintf_chk@plt+0x11f>
```

Man kunne nok finde et mindre program end *date,* som giver 1443 liniers output med ovenstående kommando. Det er jo derfor, at der er tilføjet en ***"head"*** som kun viser de første 10 linier.

Så selv om man ikke har kildetexterne, kan man - statisk, uden at køre alle programmer - finde ud af hvor mange kald der er i selve programmerne. Man siger at programmerne linker dynamisk til et shared library, når det (ved kørselstidspunktet) kobler sig til en funktion i et library så denne funktion kan kaldes i programmet.

De tre kørte både statisk og dynamisk analyse på et Android system og på et Ubuntu system.

> Af de 821 systemkald på Android bliver 114 aldrig brugt ved dynamisk linkning fra applikationer.
>
> Ca. halvdelen er dynamisk linket til mindre end 1% af applikationerne.
>
> Ubuntu følger samme trend, Posix kaldene bruges i mindre og mindre grad.
>
> I den levende analyse af et kørende system fandt de tre, at *memory* allokering var de Posix-funktioner, som blev brugt mest.
>
> Næst i rækken kommer *threading* funktionerne - typisk for at en grafisk brugerflade kan svare på brugerens klik med det samme.
>
> En undtagelse er *ioctl* ansvarlig for 16 % systemtid skønt selve funktionskaldenes antal er ganske ringe, 0.6% af samtlige kald.
>
> Et andet interessant resultat er, at Ubuntu med hensyn til pipes mellem processer er hurtig for små datamængder, men falder bagud i forhold til OS-X (BSD derivat til Apple-maskiner) med større mængder data. Android systemer ser ud til at have samme problem.
>
> Grafiske kald udgør et særligt problem, som de tre og andre har forsket i de senere år. Posix har ikke nogen grafisk standard. Linux kernen mangler et standardiseret interface til GPU.
>
> Det lyder slemt - men man skal lige huske at der er grafiske systemer som QT og GTK, der gør det muligt at skrive applikationer, som kan kompileres til de tre typer systemer; men binær distribution synes ikke muligt på tværs af platforme som overholder Posix standarden, og der er lang vej igen.

De tre forskere ender med at konstatere at OS-X grafiske funktionalitet ikke belaster systemet så meget med kald til ioctl, og at grunden er den, at driver designet af, *IOkit,* er mere struktureret end Android og Ubuntu's tilsvarende. Det skaber et genbrugeligt, producent-neutralt (agnostisk) API. Men også OS-X systemer må slås med "black box" grafisk hardware. Derfor, mener de, vil også OS-X kunne drage fordel af et standardiseret OS-interface til grafiske processorer.

Med andre ord: Standardisering kan betale sig og Posix eller tilhørende standard-specifikationer bør udstrækkes til også at omfatte kobling mellem applikationer, system og grafisk hardware.

DKUUG opstod som en forening for brugere af operativ-systemet Unix, som fra sin spæde start var blevet udbredt til mange maskiner på rekordtid. Unix, som oprindeligt er fra AT&T der ikke måtte konkurrere på software, blev distribueret gratis til universiteter. Tilgængelig kildetekst, men ikke til kommerciel brug.

ISO/IEEE JTC-1 blev dannet for at få fælles retningslinier for den (dengang nye) computerteknik.

DKUUG var med på Unix-bølgen fra 1983 og senere kunne DKUUG levere Internet-forbindelse til private (virksomheder). En af veteranerne fra dengang, **Keld Simonsen,** har i alle årene arbejdet med både Unix og standardisering, herunder Linux.

Hans interesse går bla. i retning af at lade ISO/IEEE JTC-1 give autoritet til standard-specifikationer. Standarder opstår ved at man kan se en økonomisk fordel i at alle laver ting på samme måde: fx. samme sporvidde for jernbaner, mange fælles reservedele for bilmotorer osv. - for IT folk er det en fordel ikke at være bundet til en bestemt maskine. Så kan man gå over og lave noget for en anden arbejdsgiver. For firmaerne er det en fordel at systemerne kan "snakke sammen" og at det er nemt at få fagligt kompetente medarbejdere.

Keld Simonsen ønsker at Linux Standard Base (LSB), som bygger på Posix, får det blå stempel fra JTC-1.

Nyt om standardisering i ISO SC22

Tanker om arbejde i en standardiserings-organisation

Af Keld Simonsen

Hvad er SC22?

ISO SC22 er standardiseringskomiteen *(SubCommittee **22**)*, der handler om operativsystemer og programmeringssprog. Altså POSIX og Linux, og C og C++, Fortran, Ada mm. DKUUG har igennem mange år været ganske aktiv omkring arbejdet i SC22.

Jeg var til SC22 plenarmødet i Wien, Østrig i september 2016. Bl.a. ville jeg gerne sammen med andre Linux-interesserede lande få opdateret LSB standarden fra 3.1 til 5.0, som kom for et års tid siden. Version 3.1 er fra 2006, så den trænger til at blive opdateret.

LSB står for Linux Standard Base og er en specifikation for et binært interface, der gør at man kan benytte binære pakker på de distributioner, der opfylder standarden. Altså at man kan lave et marked for binære programmer, som så kan eksistere i blot én version. Det kan også bruges som specifikation for et helt pakkearkiv som så kan benyttes på alle Linuxdistributioner. De fleste linux-distributioner er kompatible med en version af LSB. En lille undersøgelse afslørede følgende understøttelse:

Distribution	LSB-verioner understøttet
Centos 5	4.0
Centos 6	4.0
Debian 7.8	2.0-4.1
Mint 17	2.0-4.1
PClinuxOS	2.0-4.1

Dansk Standards stemme har på tidligere plenarmøder været afgørende for at vi kan undersøge fortsat hvordan vi kan få opdateret ISO-standarden. Hvis vi ikke havde været med de foregående gange, så ville USA have stoppet det sammen med deres proselytter. Denne gang var der enstemmighed om at fortsætte arbejdet, formentlig fordi Linux Foundation havde svaret positivt.

Koreansk professor henvender sig til Linux Foundations (LF) direktør

Her er et uddrag fra Professor Yong Woo Lee fra Korea, som på vegne af SC22 har skrevet et brev om baggrunden for et samarbejde til LFs direktør, Jim Zemlin: "SC 22's parent committee, JTC 1, is involved in all facets of developing international standards for Information Technology. We hope to have significant discussions with you about other areas of standardization for Linux-based specifications. Internet of Things (IoT), Smart Cities, and user interfaces are just three topic areas that come to mind. As discussed between yourself and Professor Lee, SC 22 can help the Foundation influence developments in these standards through your inclusion as a liaison in our processes. Hopefully we can initiate discussions about areas for standardization in which we could be of assistance."

Det har været svært at finde ud af hvad der sker i Linux Foundation (LF), som står for at lave LSB standarden. Det viste sig, at efter de offentliggjorde LSB 5.0 var arbejdet blevet droslet ned. Deres direktør besvarede ikke henvendelsen (som beskrevet ovenfor) om det, da der ikke var afsat ressourcer til det.

På mødet i Wien kom der dog et positivt svar. Det har i skrivende stund (december 2016) ikke været muligt at få mere at vide om dette.

ISO arbejder med at standardisere SmartCities og Internet-of-Things (IoT), som begge skal bygge på Linux, og begge skal være grundlæggende infrastruktur i fremtidens samfund.

Det er derfor vigtigt at ISO træder ind og der laves en ny ISO-standard.

Hvis ikke, kan der være risiko for at Microsoft tager en bid af kagen, ved at de gør deres operativsystem gratis til små maskiner, og det er et tab for Åbne Standarder.

Grunden til at LF drosler arbejdet med LSB ned, kan jeg nok spekulere en del over.

Der har været tanker om formålstjenligheden i en "New LSB" - forstået som arbejdet med LSB 5.0.

Værktøjerne har måske været besværlige, men det er nok blevet forbedret med 5.0, hvor man er gået væk fra et gammelt CVS værktøj og nu bruger github.

Man har gjort testning nemmere. LSB testning finder også en masse fejl der også er vigtigt at rette selv om man ikke søger at opfylde LSB.

At standarden ikke er på forkant, men konsoliderende, det er et kendetegn ved mange standarder, og en nødvendighed hvis man vil lave grundlaget for et marked med mange produkter.

LF siger at LSB ikke er så interessant, mens skyen er mere interessant. Jeg tror firmaerne i LF forregner sig, fordi modstanden mod overvågning og USAs snagen i data gør at skyen bliver mindre interessant, og at bevægelsen mod milliardvis af små dimser i Internet-of-Things gør at der er brug for en standard som LSB til at gøre udbredelsen af dimserne lettere.

En andet formål med SC22-rejsen var at følge med i de organisatoriske tiltag der er omkring standardiseringen omkring programmeringssprog og operativsystemer.

Det er jo et kerneområde for DKUUG, og DKUUG har fået mange resultater inden for dette område, bl.a internationalisering er kommet med i POSIX og Linux standarderne, hovedsageligt pga. en indsats fra DKUUGs side.

Også organisatorisk har vi stået for mange resultater, bl.a. har vi stået for email og web for mange grupper i ISO, og dette har dannet skole.

ISO holdning bremser for åbenhed

Nu vil ISO begrænse disse muligheder og selv overtage tjenesterne, så de kan styre at det kun er medlemmer der får adgang til informationen.

Det gør arbejdet langt mere besværligt, og forhindrer en del eksperter i at deltage, enten fordi de har svært ved at betale, eller de ikke er med i en nations standardiseringsorgatisation der er med i arbejdet, eller at standardiseringsorganisationen har problemer med at tilmelde dem.

Også indholdet af dokumenterne i arbejdet bliver ofte ikke gjort tilgængeligt, hvor det i de systemer som DKUUG har promoveret, var frit tilgængelige og søgbare i søgemaskinerne. Så det kan bive svært for almindelige mennesker og firmaer hvad der egentlig er under standardisering i ISO, og hvad der besluttes.

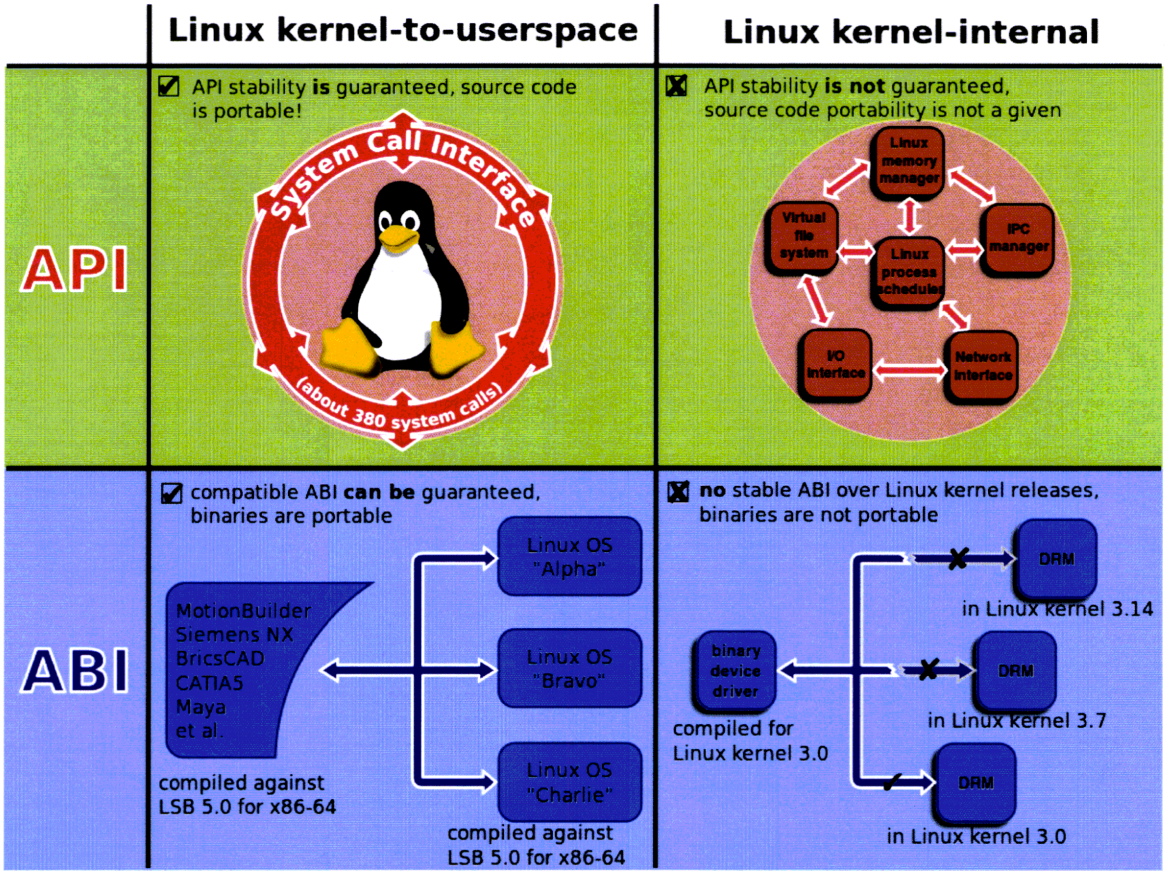
Arbejdsgrupperne er stærkt utilfredse med de nye ISO retningslinjer, og finder på måder hvor de alligevel kan få det meste information frem til den brede offentlighed, samtidig med at ISO-retningslinjerne følges.

Email kan ikke leveres i brugelig form fra ISO, som fx en diskussionsliste.

I SC22 har man haft en liste som ANSI har stået for, men den blev nedlagt for et års tid siden. Vi snakkede så om at lave en uformel email-liste, og det har jeg så gjort i samarbejde med formanden. En manglende emailliste ville have besværliggjort arbejdet betydeligt.

Ydermere sker der ofte fejl i det organisatoriske, som jeg så får rettet på møderne, Såsom at de glemmer at jeg er repræsentant fra SC22 til POSIX og til Linux Foundation. Her er det vigtigt at være fysisk til stede på mødet.Også har de glemt at POSIX er en ISO standard, hvor vi deltager direkte, her mente de at det var IEEE alende der udviklede standarden. Dette fik jeg også rettet nogenlunde op på i Wien.

Alt i alt synes jeg at vi har fået meget ud af at deltage i møderne.



TIL DET DANSKE IT-FOLK

KURSER TESTS CERTIFICERINGER FIRMAKURSER KONSULENTER

Samlet certificering til attraktiv pris

Køber du en certificeringspakke,
så har vi i øjeblikket en kampagne,
hvor du får stor rabat på alle kurser i pakken

SCD - Object Oriented Programming Certified Developer

Certificeringen dokumenterer din erfaring med at designe og udvikle programmer med objektorienterede programmeringssprog.

~~20.400 kr~~ • 14.520 kr

- **Certificering: SCD Object Oriented Progr. Certified Developer**
- 2 kurser (SU-199, SU-202)
- 2 tests (TSU-199, TSU-202)

SCD - Python Certified Developer

Certificeringen dokumenterer din erfaring med at designe og udvikle programmer med programmeringssproget Python.

~~28.800 kr~~ • 19.980 kr

- **Certificering: SCD Python Certified Developer**
- 2 kurser (SU-225 og 227)
- 2 tests (TSU-225 og 227)

SCD - IoT Windows 10 Certified Developer

Certificeringen dokumenterer overblik over IoT-teknologier, evne til at udvikle software med Windows 10 IoT samt forståelse af sikkerhed.

~~36.200 kr~~ • 25.770 kr

- **Certificering: SCD Windows 10 IoT Developer**
- 3 kurser (SU-300, 310 og 330)
- 3 tests (TSU-300, 310 og 330)

SCD - IoT Linux Certified Developer

Certificeringen dokumenterer overblik over IoT-teknologier, evne til at udvikle software med IoT på Linux samt forståelse af sikkerhed.

~~36.200 kr~~ • 25.770 kr

- **Certificering: SCD IoT Linux Developer**
- 3 kurser (SU-300, 312 og 332)
- 3 tests (TSU-300, 312 og 332)

SCD - IoT MicroController

Certificeringen dokumenterer overblik over IoT-teknologier, evne til at udvikle software på Microcontrollers til IoT samt forståelse af sikkerhed.

~~36.200 kr~~ • 25.770 kr

- **Certificering: SCD IoT MicroController**
- 3 kurser (SU-300, 314 og 334)
- 3 tests (TSU-300, 314 og 334)

SCD - Java Certified Developer

Certificeringen dokumenterer din erfaring med at designe og udvikle programmer med det objektorienterede programmeringssprog Java.

~~55.800 kr~~ • 38.160 kr

- **Certificering: SCD Java**
- 3 kurser (SU-210, 211 og 212)
- 3 tests (TSU-210, 211 og 212)

SCD - C Certified Developer

Certificeringen dokumenterer din erfaring med at designe og udvikle programmer med verdens mest anvendte programmeringssprog C.

~~33.000 kr~~ • 22.710 kr

- **Certificering: SCD C**
- 2 kurser (SU-200, SU-201)
- 2 tests (TSU-200, TSU-201)

SCD - C++ Certified Developer

Certificeringen dokumenterer din erfaring med at designe og udvikle programmer med det objektorienterede programmeringssprog C++.

~~57.000 kr~~ • 38.940 kr

- **Certificering: SCD C++**
- 3 kurser (SU-203, 204 og 206)
- 3 tests (TSU-203, 204 og 206)

MCSD .NET Developer - Web Apps

Certificeringen dokumenterer din erfaring med at udvikle og håndtere moderne web-applikationer og -services.

~~83.900 kr~~ • 59.540 kr

- **Certificering: MCSD Web Apps**
- 4 kurser (MS-480, 486, 487 og SU-207)
- 3 tests (70-480, 70-486, 70-487)
- 1 MS Surface Pro 4 inkl. keyboard
- 3 stk. Exam Ref materiale

SCDP - UML & Design Patterns Certified Developer Professional

Certificeringen dokumenterer din erfaring med at bruge UML og Design Patterns til objektorienteret design og diagrammering.

~~20.400 kr~~ • 14.520 kr

- **Certificering: SCDP UML and Design Patterns**
- 2 kurser (SU-205, SU-255)
- 2 tests (TSU-205, TSU-255)

SCD - Perl Certified Developer

Certificeringen dokumenterer din erfaring med at designe og udvikle programmer med det fortolkede programmeringssprog Perl.

~~37.200 kr~~ • 25.440 kr

- **Certificering: SCD Perl**
- 2 kurser (SU-220, SU-221)
- 2 tests (TSU-220, TSU-221)

SCDP - LAMP Certified Developer Professional

Certificeringen dokumenterer din erfaring med at installere, konfigurere samt at kunne systemudvikle på LAMP-platformen (~LAMP-stak).

~~45.450 kr~~ • 31.432 kr

- **Certificering: SCDP LAMP**
- 3 kurser (SU-142, 150 og 151)
- 3 tests (TSU-142, 150 og 151)

SCDP - Software Test Java Certified Developer Professional

Certificeringen dokumenterer viden og overblik over muligheder og fordele ved software-test samt kendskab til de forsk. slags tests.

~~17.200 kr~~ • 12.440 kr

- **Certificering: SCDP Software Test Java**
- 2 kurser (SU-260, SU-266)
- 2 tests (TSU-260, TSU-266)

SCDP - Software Test C# Certified Developer Professional

Certificeringen dokumenterer viden og overblik over muligheder og fordele ved software-test samt kendskab til de forsk. slags tests.

~~17.200 kr~~ • 12.440 kr

- **Certificering: SCDP Software Test C#**
- 2 kurser (SU-260, SU-263)
- 2 tests (TSU-260, TSU-263)

SCD - Android Certified Developer

Certificeringen dokumenterer din erfaring med at udvikle Android apps m. Android Studio el. Eclipse, Java + håndtering af applik. data.

~~48.900 kr~~ • 34.375 kr

- **Certificering: SCD Android**
- 3 kurser (LX-901, 902 og SU-210)
- 3 tests (TLX-901, 902 + TSU-210)
- 1 stk. Android tablet

SCDP - Android Certified Developer Professional

Certificeringen dokumenterer din erfaring med at udv. Android apps med prof. grænseflader, services og hardware + distr. til Google Play.

~~43.200 kr~~ • 29.970 kr

- **Certificering: SCDP Android**
- 3 kurser (LX-904, 905 og 909)
- 3 tests (TLX-904, 905 og 909)

SCD - iOS Certified Developer

Certificeringen dokumenterer din erfaring med at udvikle iOS apps med Xcode, Swift, Objective C samt håndtering af applik. data.

~~52.900 kr~~ • 38.305 kr

- **Certificering: SCD iOS**
- 3 kurser (AP-901, 902 og SU-209)
- 3 tests (TAP-901, 902 + TSU-209)
- 2 stk. hardware (iPad Mini og Mac Mini)

SCDP - iOS Certified Developer Professional

Certificeringen dokumenterer din erfaring med at udvikle iOS apps med prof. grænseflader, services og hardware + distr. til App Store.

~~49.200 kr~~ • 35.445 kr

- **Certificering: SCDP iOS**
- 3 kurser (AP-904, 905 og 909)
- 3 tests (TAP-904, 905 og 909)
- 1 stk. iPhone 6

SCD - Web App Certified Developer

Certificeringen dokumenterer din erfaring med at udvikle Web apps med HTML5, CSS3 og JavaScript + håndtering af applikationens data.

~~26.100 kr~~ • 18.925 kr

- **Certificering: SCD Web Apps**
- 2 kurser (SU-901, SU-902)
- 2 tests (TSU-901, TSU-902)
- 1 stk. Android tablet

SCDP - Web App Certified Developer Professional

Certificeringen dokumenterer din erfaring med at udvikle Web Apps m. prof. grænseflader, services og hardware + distr. disse på nettet.

~~45.200 kr~~ • 31.970 kr

- **Certificering: SCDP Web Apps**
- 3 kurser (SU-904, 905 og 909)
- 3 tests (TSU-904, 905 og 909)
- 1 stk. Android tablet

SCDP - Web Frameworks Certified Developer Professional

Certificeringen dokumenterer din forståelse og erfaring med det samlede udviklingsforløb af Web Apps både på client- og på server-side.

~~29.800 kr~~ • 20.980 kr

- **Certificering: SCDP Web Frameworks**
- 2 kurser (SU-093, SU-095)
- 2 tests (TSU-093, TSU-095)
- 1 stk. Raspberry Pi

SCDP - REST AngularJS Certified Developer Professional

Certificeringen dokumenterer dine erfaringer med REST Webservices samt AngularJS, både design, programmering og test.

~~28.800 kr~~ • 19.980 kr

- **Certificering: SCDP REST AngularJS**
- 2 kurser (SU-096, SU-097)
- 2 tests (TSU-096, TSU-097)

www.superusers.dk