Medlemsblad for

Dansk UNIX-system Bruger Gruppe

DKUUG-Nyt

Nummer 24, 1. august, 1989

Indhold

Redaktionelt

Redaktionen vil gerne gøre opmærksom på at ophavsretten til artiklerne i DKUUG-Nyt i hvert enkelt tilfælde tilhører artiklens forfatter, og at de naturligvis ikke må mangfoldigøres, helt eller delvis, uden forfatters accept. Artikler skrevet af redaktionen kan dog reproduceres uden forudgående aftale, såfremt DKUUG-Nyt og DKUUG nævnes som kilder.

DKUUG-Nyts redaktion består af Søren O. Jensen og René Seindal (ansvarshavende). DKUUG-Nyt nummer 25 udkommer d. 1. august 1989. Deadline er d. 21. august 1989. Deadline for nummer 26 er endnu ikke fastlagt, da de nødvendige resourcer først kan bevilges på næstkommende generalforsamling.

Vi er naturligvis altid interesserede i indlæg fra folk. Det behøver ikke være lange artikler, men kan også være annonceringer, opfølgninger af tidligere artikler, eller andet. Hvis I blot har ønsker eller gode ideer til artikler, er I også meget velkomne til at kontakte os. Bidrag til bladet bør indleverespå maskinlæsbar form.

Indlæg, foreslag, ønsker, etc. kan sendes med elektronisk post til redaktionen på adressen:

`dkuugnyt@dkuug.dk`

eller, hvis man foretrækker almindelig sneglepost, til:

René Seindal
Datalogisk Institut
Universitetsparken 1–3
2100 København Ø
Telefon: 01–39 64 66, lokal 221.

EUUG konference i Wien

Af *René Seindal*
seindal@diku.dk

Det er nu igen tid til at tage til EUUG konference. Konferencen dette efterår afholdes i Wien, i perioden fra den 18. september til den 22. september 1989 (det er uge 38). Tutorials vil blive afholdt den 18. og 19., efterfuldt af tre dages konference.

Hvad er en EUUG konference?

EUUG afholder to store konferencer hver år, normalt i april og september. Formålet med konferencerne er at lave et forum for udveksling af nyheder og information om EDB generelt, og natuligvis Unix i særdeleshed. Konferencerne er ret pænt besøgt, med normalt mellem 300 og 400 deltagere fra hele Europa, og fra USA. Der skulle være gode muligheder for at møde meningsfæller og ligesindede fra andre europæiske lande.

Konferencerne er normalt opdelt i to dele. Først er der to dage med 'tutorials,' som er et-dags kurser om udvalgte emner. Underviserne er ofte blandt de kendte eksperter på deres område, så ud over at give en højt kvalificeret indføring i et emne, giver en tutorial også tit mulighed for at møde en berømthed eller to.

Selve konferencen (tutorials regnes normalt ikke med som en del af selve konferencen) består af en række foredrag om forskellige af de emner, som er oppe i tiden. Foredragene bliver afholdt af eksperter fra universiteter og firmaer fra hele verden, som præsentere deres arbejde og produkter for deltagerne. Der bliver normalt afholdt i størrelsesordnen 30 foredrag på de tre dage. Alle foredragene er fulgt af en artikel i de udleverede proceedings.

Udover alt det tekniske, har konferencerne også et udpræget socialt aspekt. De giver gode muligheder for at møde gamle venner, og for at få nye bekendtskaber på tværs af landegrænserne. Med til konferencerne hører også en reception, en stor konferencemiddag, samt en udstilling med de fleste at de førende Unix leverandører. Derudover er

der naturligvis rige muligheder for at dyrke selskabeligt samvær med sine nye og gamle bekendtskaber.

Konferencen og alle tutorials foregår på engelsk, og der er desværre ikke muligheder for oversættelse.

Foredrag

Foredragene til konferencen i Wien er gennemgået i de papirer, som EUUG har udsendt til alle medlemmer. Hvis man af en eller anden grund ikke har modtaget konference-indkaldelsen, kan men enten henvende sig til DKUUGs sekretariat, eller direkte til EUUGs sekretariat i London, og bede om at få tilsendt et eksemplar.

For dem, som ikke har EUUG papirene lige ved hånden, kan det fortælles at foredragene bl.a. vil handle om standarder, multi-processor Unix implementationer, bruger grænseflader (såvel grafiske, som af den traditionelle slags), sikkerhed, RISC arkitekturer, netværks administration, og objekt orientende sytemer.

Tutorials

De udbudte tutorials er ligeledes beskrevet i EUUG papirene. Om mandagen (den 18. september) er der tutorials om emnerne "Unix netværks programmering," om programmering i netværks baserede Unix systemer, under brug af socket og TLI grænsefladerne, om "RISC arkitekturer" og de udfordringer de stiller programmørene over for, om inter-proces kommunikation under Unix System V, og om brug af make (som er let at bruge, men svær at blive klog på). De to sidste tutorials er halvdags, og overlapper ikke, så man kan melde sig til begge to, hvis man har lyst.

Tirsdagens tutorials omhandler fremtiden for 4.3BSD Unix (denne tutorial bliver præsenteret af to af systemets designere), om Andrew Toolkit'et, som er et sæt programmeringsomgivelser under X vindues systemet, og om OSI netværks-protokollerne.

Tilmelding

Tilmelding til tutorials og konference er separat. Konference tilmeldingen dækker foredragene, og de sociale aktiviteter. Ved tilmelding før den 21. august er prisen £240, hvorefter den stiger til £270. Disse priser er for EUUG medlemmer. Det er lidt dyrere for ikke-medlemmer. Priserne inkluderer ikke hotelophold, men inkluderer bespisning ved konferencen, og konference-proceedings.

Priserne for tutorials er £195 ved tilmelding før den 21. august for en heldags tutorial, og £100 for en halvdags tutorial. Herefter stiger det til £220 og £110. Priserne er igen for EUUG medlemmer.

Tilmelding sker direkte til EUUGs sekretariat i London, på en blanket, som findes i de papirer, som EUUG har sendt ud. Hvis du ikke har modtaget dem, så kan du henvende dig til DKUUGs sekretariat.

Vi ses i Wien.

Annoncer i DKUUG-Nyt

Vi er begyndt at bringe annoncer i bladet. Vores mål er at dække bladets trykningsudgifter vha. annoncer. Dette svarer til, at ca. 10% af indholdet vil bestå af annoncer, hvilket er forholdsvis beskedent i forhold til de fleste andre computer-tidsskrifter.

Prisen for en halv side ligger på kr. 500,– og redaktionen forbeholder sig retten til at anbringe annoncerne hvor *den* har lyst—vi vil dog naturligvis såvidt muligt anbringe annoncerne på fremtrædende pladser, men såvel forsiden som bagsiden er annoncefrit område.

Vi kan oplyse, at bladet for tiden udkommer i et oplag på 600 eksemplarer, dvs. at man kun betaler lidt over 1 krone pr. eksemplar for en halvsides-annonce. Endvidere bliver hvert eksemplar af bladet normalt læst af mere end én person.

Annoncerne skal indleveres til den almindelige deadline i reproklar tilstand (vi vil som sædvanligt påskønne, at få et praj på forhånd om, at der er materiale på vej)

POSIX 1003.2: Skallen og UPE

Af *Kim F. Storm*
Texas Instruments A/S

En af de egenskaber ved UNIX systemet, der har været afgørende for dets popularitet, er den lethed og fleksibilitet, hvormed man kan kombinere eksisterende programmer til at løse en foreliggende opgave, som ingen af de eksisterende programmer kunne have løst alene.

Kommandofortolkeren i UNIX (den såkaldte "skal" eller shell) tillader at alle programkald og kombinationer heraf kan lægges i en almindelig tekstfil. Denne fil kan derefter anvendes som et helt almindeligt program på lige fod med de eksisterende programmer, dvs. anvendes interaktivt eller indgå i andre shell-programmer, som disse filer også kaldes.

Da det desuden er muligt at give shell-programmer argumenter på lige fod med almindelige programmer, vil man kunne skrive selv meget store og komplicerede programmer på denne måde.

Omvendt kan praktisk taget alt hvad man vil kunne skrive i et shell-program også kunne skrives direkte på terminalen ved interaktiv brug af systemet. Dette forudsætter naturligvis at man anvender den samme shell til interaktivt brug og til udførelse af shell-programmer, men dette bliver faktisk mere og mere ualmindeligt, fordi interaktiv brug stiller udvidede krav til shellens funktionalitet på områder, der ingen betydning har ved shell-programmering.

Det primært mål med POSIX 1003.2 standarden er at fastlægge et grundlæggende sæt af programmer samt det kommandosprog (shell) hvormed disse programmer kombineres. Et shell-program, der kun anvender disse programmer, og kun kombinerer dem som foreskrevet vil være garanteret flytbarhed mellem alle POSIX systemer. Ligeledes vil C programmer, der via kald til *system*() eller *popen*() kun aktiverer de af 1003.2 definerede programmer ligeledes være garanteret flytbarhed mellem POSIX systemer.

Man sigter også på at standardisere bl.a. programmer til system vedligeholdelse, netværksrelaterede programmer, interaktive programmer (herunder editorer), tekstbehandlingsprogrammer og database pro-

grammer, men dels varetages flere af disse opgaver allerede i andre fora, dels imødeser man ret store kvaler med at opnå konsensus om flere af disse emner.

Som næste skridt sigter man derfor på at specificere den såkaldt UPE, der står for "User Portability Extension" eller "User Programming Environment" afhængigt af hvem man taler med. UPE vil omfatte programmer, der anvendes af en terminalbruger, men som ikke finder anvendelse i shell-programmer.

Men foreløbig koncentrerer man sig altså om flytbarhed af shell-programmer og lader brugerne blive hvor de er.

Et andet område, hvor man er meget aktiv er i forbindelse med forskellige grænseflader mellem programmer og terminaler/brugere. Et tiltag er den såkaldte FIMS (Form Interface Management System), der generelt set er en model for hvordan skemabaserede programmer (data-entry systemer) kommunikerer med en skræmbruger, således at applikationen frigøres for kendskab til den aktuelle skærms egenskaber.

I det følgende lægges vægten på shell-programmering og de programmer, der standardiseres i denne sammenhæng.

Skallen

Lad os begynde med at se på hvordan et shell-program til løsning af en specifik opgave kan opbygges udfra de basale programmer.

Vi ønsker at finde alle de filer i systemet, hvori et bestemt ord forkommer mindst et givet antal gange og få udskrevet navnene på disse filer og hvor mange filer der er.

Følgende kald vil producere en liste over alle filer i systemet:

```
find / -type f -print
```

Uddata fra dette kald vil blive skrevet på skærmen, men vi ønsker jo at arbejde videre med disse filnavne, så vi redirigerer uddata til en fil. Denne fil vil indeholde et filnavn på hver linie:

```
find / -type f -print > alle-filer
```

Vi ønsker nu at gennemløbe denne fil og søge efter det givne ord i hver eneste fil. Til dette anvender vi en løkke i shellen:

```
while read FILNAVN
do
     ...
done < alle-filer
```

Her vil løkken blive udført en gang for hver linie i filen `alle-filer` og variablen `FILNAVN` vil blive sat til det filnavn, der står på den aktuelle linie i filen.

For at søge efter ordet `POSIX` i filen hvis navn findes i variablen `FILNAVN` anvender vi kaldet

```
grep -F -i -c POSIX $FILNAVN
```

Dette kald vil udskrive antallet af linier, hvori ordet `POSIX` forekommer. Dvs. hvis dette kald producerer en værdi større end nul, er det en af de filer vi er interesseret i. Lad os gemme resultatet af *grep* i en variabel:

```
(( ANTAL = `grep -i -c POSIX $FILNAVN` ))
```

Vi kan herefter teste om `ANTAL` er større end f.eks. 5 med en betinget sætning:

```
if (( ANTAL > 5 ))
then
     ...
fi
```

Vi kan nu kombinere alle disse stumper til et program, hvor vi desuden vil kombinere uddata fra en kommando med inddata på den næste via en pipeline. Det ord vi søger efter får vi som det første element i argumentlisten når vi kalder shell-programmet under navnet `$1`, og det minimale antal forkomster i hver fil får vi fra det andet argument under navnet `$2`:

```
(( filer = 0 ))

find / -type f -print |
while read FILNAVN
```

```
do
     if (( 'grep -F -i -c $1 $FILNAVN' >= $2 ))
     then
          (( filer++ ))
          echo $FILNAVN
     fi
done

echo Ordet $1 findes min. $2 gange i $filer filer.
```

Vi kan nu søge efter forskellige ord ved at kalde dette program:

```
find-ord POSIX 3
find-ord Kim 1
```

I eksemplet anvendes shellens kontrolstrukturer, variable og udtryk til at kombinere programmerne 'find,' 'grep' og 'echo' på den ønskede måde.

En balancegang i høj sø

1003.2 arbejdsgruppen har haft et svært valg mht. valg af kommandofortolker, da dette naturligt nok er en religion for mange erfarne UNIX brugere. Der har dog altid været en relativt stor enighed om at den gamle Bourne shell, som findes på ethvert UNIX system, umiddelbart er nemmest at gå til når man skal skrive shell-programmer, men at den lader en hel del tilbage at ønske når det drejer sig om interaktiv brug.

Den store konkurrent til Bourne shell er den såkaldte C shell, der har en række fornuftige faciliteter, der gør livet nemmere ved interaktiv brug, og som programmeringsmæssigt ligger ret tæt op ad C-programmeringssproget (deraf navnet). Man skulle tro at dette ville appellere til de fleste programmører, men reelt har C shellen aldrig rigtig slået an ved shell-programmering. For de programmører som imidlertid har skiftet til C-shellen, ville en standardisering af den almindelige Bourne shell dog være et stort tilbageskridt, da C shellen pga. slægtsskabet med C-sproget naturligt nok er en del mere fleksibel og også er bedre i håndteringen af programmeringsfejl.

Man har derfor vendt blikket mod den såkaldte Korn shell (der dog ikke har noget med Isak at gøre). Korn shellen er i syntaks og struktur bygget over Bourne shell, men med en række udvidelser, der giver en række af de programmeringsfordele som C shellen giver. Da Korn shellen desuden på flere områder er endnu bedre end C shellen til interaktiv brug kunne det tænkes at man vælger at standardisere hele Korn shellen i UPE.

Lad os kort opsummere, hvad det er shellen tilbyder:

- Omdirigering af inddata og uddata til filer eller andre programmer (pipelines).
- Variable og artimetik.
- Kontrolstrukturer såsom
  - while, der repeterer så længe en betingelse er sand.
  - for, der repeterer for hvert element i en ordliste.
  - if, der er en betinget sætning, evt med en else del.
  - case, der er en flervejsforgrening.
- Procedurer, dvs. man kan erklære underprogrammer, der kun anvendes i det aktuelle program, men som i øvrigt fungerer på lige fod med de almindelige programmer.

Forbedringer i POSIX shellen

Den væsentligste udvidelse i forhold til den almindelige Bourne shell er mht. betingelser og aritmetik. Traditionelt har man anvendt programmet *test* (der aht. eksisterende programmer er taget med i POSIX), men det har det problem at dets argumenter blev behandlet af shellen inden programmet blev kaldt, hvorved en række fejl ikke umiddelbart opdages.

Betingelser der anvender den nye syntaks [[...]] vil nu helt og holdent blive behandlet af shellen, hvorved en række klassisk fejl kan undgås. F.eks. vil følgende betingelse (der normalt vil virke) give en syntaksfejl, hvis variablen `X` indeholder den tomme streng (fordi test kun får argumenterne "= plus"):

```
if test $X = plus
```

I POSIX shellen vil man uden risiko kunne skrive

```
if [[ $X = plus ]]
```

I Bourne shellen var aritmetik en omstændelig og langsommelig affære, da det måtte gøres gennem kald af andre programmer (især *expr*). F.eks. skrives en løkke, der gennemløbes ti gange med en variabel `NUM` varierende fra 1 til 10, på følgende måde:

```
NUM=1
while [ $NUM -le 10 ] ; do
     ...
     NUM="`expr $NUM + 1`"
done
```

I POSIX shellen kan vi blot skrive

```
NUM=0
while (( ++NUM <= 10 )) ; do
     ...
done
```

Basale programmer

De fleste programmer, der standardiseres af 1003.2 er gode gamle kendinge fra UNIX. Imidlertid er UNIX efterhånden kommet i så mange varianter, at også disse velkendte programmer varierer ganske betragteligt i hvilke "ekstra" faciliteter, der er bygget ind i dem.

Nogle programmer er desuden implementeret på forskellig vis i forskellige varianter af UNIX, hvilket betyder at det "samme" program i visse grænsetilfælde (ingen inddata, tomme argumenter, etc) opfører sig forskelligt.

Man har i POSIX 1003.2 taget en ret konservativ indstilling til disse programmer, idet man har tilstræbt et minimalt sæt af programmer, der giver de samme muligheder som de mange ekstra faciliteter og programmer, men muligvis på en anden måde.

Dette betyder, at det reelt ikke vil kræve en større indsats af den enkelte leverandør at ændre sit system til at være i overensstemmelse med POSIX standarden. De fleste programmer findes i forvejen, og de vil kun kræve minimale ændringer—eller slet ingen. POSIX forbyder nemlig ikke at disse programmer også har andre funktioner end de der er standardiseret—blot de opfører sig som foreskrevet når standard-funktionerne anvendes.

Et eksempel er programmet *tail*, der findes på mange systemer, men som ikke er med i POSIX 1003.2.

En anvendelse af *tail* er at plukke de sidste linier i en fil ud, f.eks. de sidste 10 linier af en logfil. Dette kan også gøres med programmet *sed* på en noget mere besværlig måde, men da det jo drejer sig om anvendelse i shell-programmer og ikke til interaktiv brug, betyder det ikke meget at *sed* ikke er helt så nem at bruge som *tail*.

En anden anvendelse af *tail* er løbende at følge med i hvordan en fil vokser, og selvom der ikke er en god alternativ mulighed for at gøre dette med de eksisterende programmer, har man valgt at undvære *tail*, da denne anvendelse er af interaktiv karakter og derfor hører hjemme i UPE.

Et andet eksempel er programmerne *tar* og *cpio*, der anvendes til sikkerhedskopiering på bånd (og andre ting). Her har slaget om hvilket af programmerne man skulle foretrække været så voldsomt, at man faktisk er røget over i den anden grøft og i stedet har defineret et helt nyt program, *pax* (fred), som dels forstår både tar og cpio formaterne, dels definerer sit eget nye POSIX format.

Man har dog også været åben overfor at standardisere nogle af de fornuftige udvidelser af traditionelle programmer, som kun findes i nogle UNIX varianter, f.eks. har både *mv* og *cp* programmerne fået *-i* og *-f* optioner som ellers kun er almindelige på *rm* programmet, og *cp* har desuden fået *-rp* optioner, som vil kopiere et helt fil hierarki, hvor ejerskab og andre attributter på kopierne er identiske med originalen.

Filtre

De fleste basale programmer, som anvendes i shell-programmer fungerer som såkaldte filtre, hvilket vil sige at deres uddata består af en eller anden form for selektering eller transformation af deres inddata.

Ind- og uddata er normalt almindelig tekst, dvs. læsbare data, der kan opfattes som en samling af ord og linier.

Eksempler herpå er

- *grep*, der udvælger linier i inddata efter deres indhold.
- *sort*, der sorterer inddata på forskellig vis.
- *sed* og *tr*, der kan substituere tekstudsnit med andre.
- *awk*, der er et helt programmeringssprog i sig selv, og som er specielt indrettet på at arbejde på data i tabelform.
- *cut* og *paste*, der kan klippe og klistre "på langs" i inddata.
- *cat* og *pr*, der kan sætte filer sammen efter eller ved siden af hinanden.
- *comm* og *join*, der kan flette filer på basis af deres indhold.
- *uniq*, der kan smide gentagne linier i inddata væk.
- *cmp* og *diff*, der kan sammenligne to filer og producere en liste over forskellene som uddata.
- *wc*, der kan tælle antal tegn, ord og linier i inddata.

Det ser muligvis ikke ud af så meget, men alle disse programmer kan via styrende parametre (options) bringes til at udføre mange variationer af den grundliggende funktion. F.eks. kan *uniq* programmet også udføre følgende operationer:

-c Hver linie i uddata vil starte med et tal, der angiver hvor mange gange linien forekom i inddata.

-d Kun gentagne inddatalinier vil forekomme i uddata.

-u Kun unike inddatalinier vil forekomme i uddata.

Desuden kan man begrænse uniq til kun at se på et udsnit af hver linie i bestemmelsen af om to linier betragtes som ens eller ej.

Det er ligetil at lave shell-programmer, der selv fungerer som filtre, f.eks. vil følgende program sortere en række tal og producere en liste af de unike tal, der ligger i et givet interval (angivet som argumenter til programmet):

```
sort -n |
uniq -u |
awk "\$1 >= $1 && \$1 <= $2"
```

Filbehandling

En anden familie af basale programmer arbejder med filer og kataloger. Uddata fra disse programmer vil ofte blive viderebearbejdet vha. de ovennævnte filtre for at give det endelige resultat.

De vigtigste programmer er:

- *find*, der kan give en liste over filer, hvis beliggenhed, navn og attributter stemmer overens med et givet sæt af betingelser.
- *ls*, der kan give en liste over filer og deres attributter.
- *basename* og *dirname*, der uddrager bestanddele af et filnavn.
- *cp*, *mv*, *ln* og *rm*, der kopierer, flytter, linker og sletter filer.
- *mkdir* og *rmdir*, der opretter og fjerner kataloger.
- *pax*, der anvendes til sikkerhedskopiering af filer.

Hjælpeprogrammer

Et vigtigt nyt program i 1003.2 er *getopts*, der anvendes til afkodning af parametre til et shell-program (i lighed med *getopt*() funktionen i C). Afkodning af parametre har traditionelt været en besværlig affære, hvorfor mange shell-programmer har en meget primitiv (læs: usikker) behandling af sine parametre. Med *getopts* er det muligt at komme nemt om ved afkodningen af selv komplekse parametre, så på dette område har POSIX gjort det helt rigtige.

En typisk afkodning af styrende parametre til et shell-program vil ske på følgende måde:

```
# Mulige options er -a -b -oFIL

FLAG=""
FIL=""

while getopts abo: OPTC
do
     case $OPTC in
     a | b) FLAG=$OPTC;;
     o)     FIL=$OPTARG;;
     ?)     echo Fejl: ...
            exit 2;;
     esac
done
shift `expr $OPTIND - 1`
```

Tegnsæt og Internationalisering

POSIX 1003.2 forudsætter at følgende tegn er til rådighed og tillægger især symbolerne særlige betydninger. Det er imidlertid kun selve symbolet man standardiserer—ikke repræsentationen:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 ! @ # $ % ^ & * ( ) _ + + = { }
[ ] : " ~ ; ' ` < > _ , . | \ / blank tab linieskift
```

En implementation må gerne omfatte flere tegn end disse, men anvendelse af disse kan betyde at flytbarheden af programmet ikke længere kan garanteres. Specielt definerer POSIX den tegnmængde som man kan bruge i filnavne, hvis man vil sikre flytbarhed mellem forskellige systemer.:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . - _
```

POSIX 1003.2 tillægger altså tegnene [ ] { } \ og | en særlig betydning, så problemerne for danske terminaler med ISO 646 7-bit tegnsæt er ikke

løst med POSIX. På den anden side er der intet i POSIX, der forhindrer brug af f.eks. ISO 8859/1 eller endog ISO 10646 tegnsættene—forudsat at programmerne indrettes til at håndtere disse tegnsæt.

Mht. internationalisering har man dog ikke ligget på den lade side. Man har haft flere forslag oppe at vende, og det er endnu for tidligt at sige hvordan den endelige standard vil se ud.

Det man foreløbig er nået frem til er samlet i begrebet *locale*, der følger anbefalingerne fra */usr/group*. Et locale specificerer relationer mellem tegnsæt og sprog, f.eks. sorteringsorden, ækvivalensklasser (f.eks. e é è ê), konvertering mellem store og små bogstaver, datoformater, valutasymboler, decimaltegn, ja/nej svar osv.

Man vil typisk have et locale for hvert sprog, og en bruger kan så vælge det locale der passer i den givne sammenhæng (f.eks. vil en dansker, der arbejder med en engelsk tekst, kunne vælge det engelske locale). POSIX 1003.2 definerer selv kun et locale svarende til den traditionelle UNIX/ASCII opførsel, men man regner med at anvende et dansk og et japansk locale som eksempler på mere avancerede locale specifikationer.

Vi er blevet bedt om at skrive det danske locale, men vi har jo et lille problem med "å" kontra "aa". I en navneliste vil vi typisk ønske at opfatte disse som helt ækvivalente (Xgård og Xgaard), hvilket kan gøres helt trivielt i et locale. I en simpel ordliste vil en sådan ækvivalens imidlertid skabe problemer med sammensatte ord, f.eks. ville ekstraarbejde og ekstrajob blive sorteret i forkert orden.

Et locale (kan) beskrives i et flytbart tekstformat, som via et program omsættes til et internt uspecificeret format. Ved en passende opsætning af variable i shellen kan man fortælle programmer hvilket locale de skal anvende ved f.eks. sortering. Det står leverandører og brugere frit for at sammensætte nye locale specificationer, men forhåbentlig vil man enes om "standard" specifikationer for de almindelige sprog.

Program distribution og installation

POSIX 1003.2 definerer dels et format, hvori programmel skal distribueres (*pax*), et simpelt beskrivelsessprog og et par applikationer til brug ved installation af programmel.

Dette betyder at diverse forudsætninger for at en pakke kan installeres på et givet system (f.eks. om der er en POSTSCRIPT-skriver) kan specificeres, add-on moduler kan specificeres, placering af filer og programmer kan specificeres, osv. og disse kan gives et såkald ressource-navn.

Et shell-program kan bruge programmet *findrsrc* til at hente f.eks. et filnavn svarende til en ressource, og et C program kan med *getrsrc*() kaldet gøre det samme. Dvs. det er ikke længere nødvendigt at bestemme placeringen af diverse ressourcer på oversættelsestidspunktet, og dette sker på en standardiseret måde, således at konfigureringen af forskellige programpakker vil følge samme mønster.

Programmeludvikling

POSIX 1003.2 har også standardiseret de programmer, der skal være til rådighed i forbindelse med programmeludvikling. Det er valgfrit om et POSIX system vil inkludere disse programmer, men hvis det gør, skal alle programmer være til stede. Specifikationen er ikke bundet til et bestemt programmeringssprog, idet der specificeres en række styrende parametre, der skal have samme funktion på enhver oversætter i programmeringsmiljøet.

Et C programmeringsmiljø vil omfatte følgende programmer:

cc	C oversætteren.
make	Program til styring af oversættelsen.
ar	Arkiv program til opbygning af biblioteker.
lex	Leksikalsk analysator generator.
yacc	Oversætter generator.

UPE — User Portability Extension

Man har endnu ikke besluttet hvilke programmer, der skal standardiseres i UPE, men nogle af de programmer og systemer der overvejes er:

SCCS	Kildetekstkontrolsystem.
UUCP	Programmer til kommunikation mellem unix systemer. POSIX 1003.2 omfatter nu kun uux programmet, der kan starte programmer på et andet system.
ex/vi	Editorer.
pg	Skærmfilter.

Hvis det hidtidige slag om hvilke basale programmer, der skulle standardiseres i 1003.2, har været hårdt, så bliver slaget om UPE næppe mindre. Mange erfarne brugere har et nærmest religiøst forhold til netop de programmer, de foretrækker, og det er jo ofte dem, der deltager i standardiseringsarbejdet.

F.eks. kunne det med lige så god ret hævdes at følgende programmer burde vælges fremfor de ovennævnte—og de vil da med sikkerhed også blive overvejet:

RCS	Alternativ til SCCS.
emacs	Alternativ til *vi*.
more/less	Alternativer til *pg*.

Det bliver spændende at følge slagets gang, og ringen er stadig åben for nye kombattanter.

System-administration – del 4

Af *Kim Chr. Madsen*
UniTack A/S

Opsætning af Terminaler og Printere

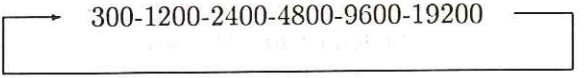
Denne side af system-administratorens opgave er lidt af en broget affære, især hvis der skal vælges mange slags forskelligt hardware, der skal tilkobles den samme Unix-maskine. Opgaven går i al sin enkelthed ud på at finde ud af hvilke karakteristika den enkelte terminal eller printer har, for derefter at få den til at fungere sammen med resten af systemet.

Opsætning af Terminaler

Når man ønsker at tilkoble en terminal til systemet er første skridt at få den fysisk tilkoblet, dvs. finde en ledning med de rigtige stik i begge ender, finde ud af om systemet eller terminalen kræver at ledningen skal være krydset eller ej.

Når man er kommet i forbindelse med terminalen, dvs. at terminalen er forbundet med systemet og terminalens setup er i orden, kommer næste øvelse i at snakke med systemet. Normalt køres der fuld duplex mod systemet, dvs. når der skrives på terminalen er det ensbetydende med at der er hul igennem til systemet. Nu gælder det om at finde ud af hvilken *baudrate* systemet forventer at terminalen kører. Man kan da enten indstille terminalen til denne baudrate eller ændre systemets baudrate på terminal-udgangen, dette gøres normalt i filen `/etc/inittab` (på System V Unix) eller i filen `/etc/ttys` (Xenix og BSD Unix). Man kan normalt få Unix-systemet til at skifte baudrate temporært ved at trykke på `BREAK` tasten gentagne gange indtil den baudrate man ønsker kommer frem. Unix-systemet vil normalt gennemløbe en cyklisk sekvens (se figur 1)

Når man har fået forbindelse til systemet og etableret den rigtige baudrate, skulle systemet gerne svare med en login-meddelelse. Du kan nu logge ind på systemet.



Figur 1: Baud-Rates

Nu gælder det om at finde ud af om Unix-systemet kender til terminaltypen, dette gøres på en af følgende måder:

- Hvis systemet anvender *terminfo*-systemet (Unix System V) skal du lede i kataloget `/usr/lib/terminfo/x`, hvor '`x`' refererer til begyndelses-bogstavet for teminalnavnet. I dette katalog ligger der en række binære terminal-beskrivelser for terminaler, der begynder med dette startbogstav eller tal. F.eks. hvis du tilslutter en *concept-100* terminal, ser du i kataloget `/usr/lib/terminfo/c` og ser om der er en fil, der hedder noget i retning af `concept100` eller `concept-100` eller lignende. Hvis en sådan fil eksisterer er alt i orden, og du skal blot sørge for at brugere, der benytter denne terminal husker at få sat `TERM` variablen til navnet på filen f.eks. `concept-100`.

  Hvis terminalen ikke findes skal der opbygges en terminfo-beskrivelse af terminalen, dette kan gøres ved hjælp af en editor (hvis du ikke har nogen terminaler, der er kendt af systemet er du nødt til at bruge en linieorienteret editor f.eks. *ex*(1) eller *ed*(1). (Se afsnittet om opbygning af terminfo-beskrivelser).

- Hvis systemet anvender *termcap* (Version 7 Unix, BSD Unix, Xenix, etc.) undersøger du filen `/etc/termcap` for en indgang til terminalen, hvis en sådan findes er alt i orden, ellers må du opbygge en ny indgang i filen. (Se afsnittet om opbygning af termcap-beskrivelser).

Opsætning af termcap

Hvis dit system benytter termcap-systemet til at beskrive sine terminaler, og du vil tilføje en ny terminal til dit system som ikke allerede

```
mytty:Min egen lille terminal:\
:co#80:li#24:am:bs:cl=^Z:cm=\E]%i%2;%2H:\
:nd=^L:up=^K:do=^J:le=^H:ce=\E^L:cd=\E^K:\
:so=\EEB:se=\EEE:ku=^K:kd=^J:kl=^H:kr=^L:\
:k1=\E]OP:k2=\E]OQ:k3=\E]OR:k4=\E]OS:
```

Figur 2: Eksempel på termcap indgang

er beskrevet i filen `/etc/termcap`, skal du lave en termcap-beskrivelse til denne terminal. Dette gøres nemmest på følgende måde:

Lad os sige at du vil oprette en terminal som ikke er kendt på forhånd i termcap-filen, som hedder *mytty*, du opretter da i dit eget katalog en fil ved navn `mytty`, i denne fil laver du nu en termcap beskrivelse af hvilke egenskaber denne terminal er i besiddelse af. Til dette får du brug for den tekniske manual til terminalen samt manualen til termcap fra manualafsnit 5 i Unix Manualerne.

Når du har lavet termcap-indgangen til din terminal sætter du omgivelsesvariablen `TERMCAP` til at indeholde det fulde stinavn til den fil du havde lavet termcap-indgangen i. Du kan nu teste om terminalbeskrivelsen er korrekt, ved at prøve forskellige skærmorienterede programmer, f.eks. *vi*.

TIP: Hvis den nye terminal ligner en eksisterende terminal meget kan det godt betale sig at kopiere termcap-beskrivelsen fra denne terminal til din nye terminal-beskrivelse og rette den til.

Eksempelet i figur 2 på beskrivelse af terminalen *mytty* er egentlig en lang linie, men tegnet "\" for enden af linien binder de enkelte linier sammen. Hvert felt er adskilt af kolon, og beskriver en attribut ved terminalen. Disse attributter har hver en to-tegns kode og kan opdeles i tre grupper: *numeriske*, f.eks antal linier på skærmen, antal tegn per linie etc. Disse attributter skrives som *kode#nummer*. *Flag* som enten kan være sat eller ej, dette kan f.eks. være om terminalen har automatisk liniewrap eller ej. Disse slås til ved at angive koden og slås fra ved at udelade koden. Den sidste slags attributter, er *streng*-attributter, f.eks. den tegnsekvens, der skal sendes til terminalen for at placere cursoren på et bestemt sted. Her anvendes en del meta-tegn til at markere

kontroltegn (f.eks. \E for escape, "^*tegn*" for Kontrol-*tegn*), samtidigt bruges en simpel form for parameter overførsel til tegnsekvenser, der behøver dette, f.eks. sekvensen for at placere cursoren på en given koordinat på skærmen. Strengkoder angives som *kode=tegnsekvens.*

Første linie i beskrivelsen er en identifikation af de navne som terminalen er kendt under, hvis der er flere navne adskildes disse ved hjælp af tegnet "|", sidste felt i denne opremsning er et kommentarfelt og er det eneste felt, der må være blanktegn i.

Ulemper ved termcap-systemet

Selve ideen med termcap er genial, idet det sikrer at man kan sætte en vilkårlig terminal på sit Unix-system og blot fortælle Unix hvilken type terminal man benytter, for derefter at kunne køre sine skærmorienterede applikationer.

Men da termcap systemet blev designet først i halvfjerdserne er der ikke taget højde for nogle af de ting vi i dag forlanger af moderne terminaler: Farver, Window-management, forskellige tegnsæt, mapning af tegnsæt, grafik, forskellige tegntyper etc. Dette har betydet, at visse applikationer ikke kan nøjes med at benytte termcap, men derimod definerer en ny, med de udvidelser de har brug for til de terminaler, der kan klare disse udvidelser. Dette skyldes at man endnu ikke er blevet enig om en ny standard for termcap. Det kommer forhåbentligt, idet det er uholdbart at hvert applikations-program der benytter avancerede terminalfunktioner, skal have deres egen termcap, og dermed gøre byrden for system-administratorer større end nødvendigt.

Opsætning af terminfo

I stedet for termcap-systemet er der nogle systemer, der anvender terminfo-systemet, som er en videreudvikling af termcap-systemet, hvor man anvender enkelte filer for hver terminal. Hver af disse filer indeholder en oversat udgave af terminal-beskrivelsen. Man har dog lavet en del udvidelser i forhold til termcap, idet der er medtaget en del flere funktioner og attributter til terminalerne, til behandling af streggrafik, farver etc., samt mere sigende navne for attributterne.

```
mytty|Min egen lille terminal,
cols#80, lines#24, am, bs, clear=^Z,
cup=\E]%i%2;%2H, cuf1=^L, cuu1=^K,
cud1=^J, cub1= ^H, el=\E^L, ed=\E^K,
smso=\EEB, rmso=\EEE, kcuu1=^K, kcud1=^J,
kcub1=^H, kcuf1=^L, kf1=\E]OP, kf2=\E]OQ,
kf3=\E]OR, kf4=\E]OS,
```

Figur 3: Eksempel på terminfo indgang

```
$ su                Bliv superbruger
password:           tast superbruger password
# tic mytty.ti      oversæt terminfo beskrivelse
```

Figur 4: eksempel på brug af *tic*

På samme måde som vedtermcap-systemet skal der for en ny terminal oprettes en fil indeholdende en beskrivelse af den nye terminals attributter. For at danne denne fil er det en god ide at have en manual for terminalen samt manualen for *terminfo* (afsnit 5 i Unix-manualen) ved hånden.

På figur 3 kan det ses hvordan termcap-indgangen fra figur 2 ser ud omsat til terminfo.

Når man har lavet sin terminfo-beskrivelse af terminalen skal denne oversættes, således at de skærmorienterede applikationer på systemet kan benytte denne information. dette gøres ved kommandoen *tic* (Terminal Information Compiler) som vist på figur 4 (Hvis filen hedder `mytty.ti`):

Hvis man er i besiddelse af en termcap-beskrivelse til en terminal, som man er interesseret i at få en terminfo-beskrivelse af til sit system, kan terminfo-oversætteren *tic* også håndtere at omdanne termcap-beskrivelser til oversatte terminfo-beskrivelser.

Ulemper ved terminfo-systemet

Ideen med terminfo-systemet, var at gøre tilgangen til terminal-beskrivelserne hurtig, ved dels at splitte termcap-filen op i separate filer

for hver terminal samtidigt med at oversætte de enkelte terminal-beskrivelser til absolutte filer, igen for at gøre tilgangen til de enkelte attributter hurtigere.

Dette har medført at tilgangen til terminal-beskrivelserne og de enkelte attributter er blevet hurtigere, på bekostning af det ekstra besvær brugeren skal have med at oversætte nye terminal-beskrivelser og at man ikke umiddelbart kan læse eksisterende terminal-beskrivelser. Visse systemer har dog en medfølgende kildetekst-listning af terminfo, eller en kommando kaldet *infocmp* til at udskrive en oversat terminal-beskrivelse i læsbar form.

Ydermere har systemet den begrænsning at der ikke umiddelbart kan tilføjes nye attributter, uden at ændre terminfo-oversætteren *tic*.

Flere forskellige Unix-leverandører har dog lavet en del udvidelser til terminfo, hvilket gør terminfo-beskrivelser "non-portable" mellem forskellige systemer. Dog er tilpasning af en udvidet terminfo-beskrivelse fra et system dog rimelig nem at tilpasse et andet system, ved at fjerne de attributter som *tic* beklager sig over. Den modsatte vej er der ingen problemer.

Printer opsætning

Dette område er straks mere kompliceret end terminal opsætning, idet der ikke normalt findes en printer beskrivelse i stil med termcap eller terminfo. Dog har Berkerley's BSD Unix en `printcap`, der tilsvarer termcap, der kan bruges af applikationsprogrammer til at finde escape-sekvenser til diverse attributter på printeren. Der findes også en del applikationsprogrammer, der selv definerer en `printcap`-fil for forskellige standard-printere. Dette er dog lidt kedeligt, idet man da får en del printcap-filer liggende forskellige steder i filsystemet, som alle skal opdateres når en ny printer introduceres på systemet.

Et andet problem ved printer-opsætning er de ting, der skal laves for overhovedet at få en korrekt kommunikation i stand til printeren. Dette består normalt i to tempi:

Tilslutning af ny printer

Der findes to forskellige typer af printer-tilslutninger, seriel-tilslutning og parallel-tilslutning, hvor der normalt bruges RS232C-kabler til seriel-tilslutning og Tektronix-kabler til parallel-tilslutning.

Når man skal tilslutte printeren, er det nødvendigt at sætte den op til den rette kommunikationsform. Dette inkluderer indstilling af *dip-switches* på printeren samt evt. gennemgang af opsætningsmenu på printeren. Hvis printeren f.eks. er i stand til at køre både parallelt og serielt skal den sættes til den kommunikationsform, som du ønsker at benytte.

Parallel-tilslutning af printer

Opkobling af parallel-printere er normalt den nemmeste tilslutning til et system, da det parallelle interface er standard og der ikke skal tænkes på baudrate, XON/XOFF-protokol, etc. Dog skal der enten på printeren eller på Unix-systemet sørges for at linieskift foregår korrekt, dette er nødvendigt idet Unix skiller linier ad vha. `Line-Feed`, hvorimod de fleste printere forventer at linier skilles af ved `Carriage-Return` efterfulgt af et `Line-Feed`. Mange printere kan dog selv mappe `Line-Feed` om til `Carriage-Return` efterfulgt af `Line-Feed`, ellers må dette gøres med et *stty*-kald, se opsætningen af seriel-printer.

Seriel-tilslutning af printer

Opkobling af seriel-printere er lidt mere besværlig end parallel-printere, men til gengæld kræver de ikke specielle Tektronix-udgange og kabler, men kan klare sig med almindelige terminal-udgange, normalt RS232C-udgange.

En seriel-tilslutning kræver af kommunikationen mellem printeren og Unix-systemet stemmer overens, dvs. at kommunikationshastigheden (baudrate) er rigtig, om der benyttes paritets-kontrol på overførte data, og hvilken, om hvor mange stopbit der benyttes, etc.

Disse kommunikationsparametre sættes normalt op i printsystemets backend-program, som en række *stty*-kald

Hvis en printer forventer følgende kommunikationsparametre:

- 9600 Baud

- 8 Databit
- 1 Stopbit
- Ingen Paritet
- XON/XOFF-Protokol
- NL omformes til CR-NL
- Tabulatortegn omformes til tilsvarende antal blanktegn

Kan denne opsætning fås ved at bruge følgende *stty*-kald:

```
stty 9600 cs8 cread clocal -cstopb
stty opost onlcr ofill tab3
stty ixon ixany
```

Hvor stty kaldene skal sættes på den pågældende port.

NOTE: `cread` option til stty er nødvendigt for at printeren kan sende XON/XOFF-signaler til Unix-systemet. Hvis dette udelades og printeren kører XON/XOFF-protokol, kan man komme ud for at data mistes på printeren ved store mængder data.

Installation af nye software-pakker

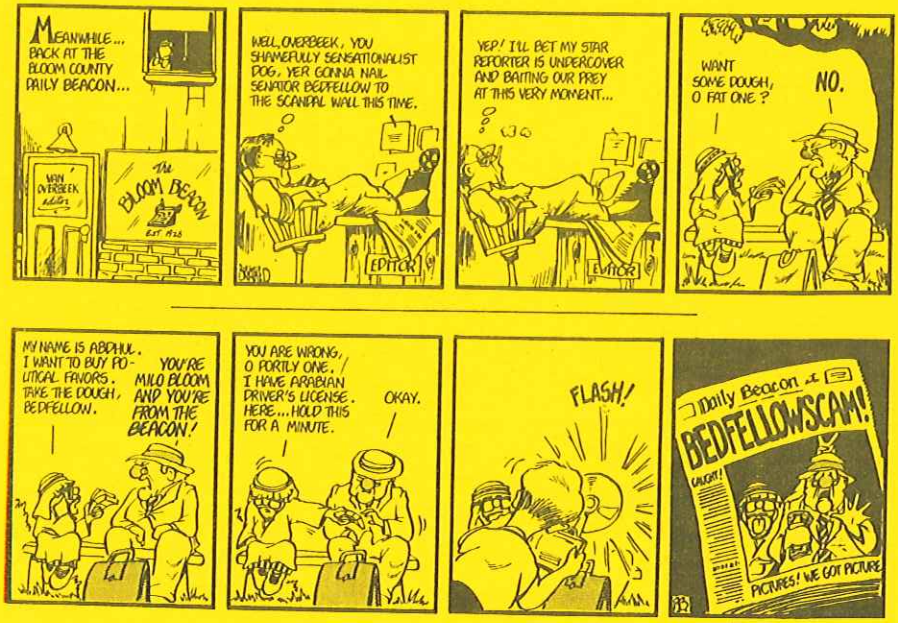
Denne del af system-administrationen er normalt et ret simpelt job, idet det meste af arbejdet er gjort af software-leverandøren. Normalt vil opgaven bestå i at læse en installationsvejledning igennem for derefter at sætte diskette eller bånd i maskinen og udføre de kommandoer, der er beskrevet i installationsvejledningen.

Normalt vil nye produkter komme som backup-bånd, som kræver at du læser dem ind det rigtige sted i filsystemet enten vha. af *tar* eller *cpio*. Når dette er gjort skal der normalt køres et installationsprogram, der følger med systemet. Dette installationsprogram, vil normalt komme og spørge dig om forskellige ting angående installationen, f.eks. et serienummer og licensnummer på produktet, og vil derefter gøre hvad der er nødvendigt for at få systemet til at køre.

Dog kan installationsvejledningen angive yderligere opgaver, der skal laves f.eks hvis din system konfiguration ikke er standard. Sådanne

ekstra opgaver kan f.eks. bestå i at tilføje beskrivelser af terminaler og printere til systemets egne termcap-lignende filer, etc.

Installation af nye software-pakker skal normalt gøres som superbruger, idet mange software-pakker forventer at kunne skrive i kataloger, hvor kun superbrugeren har adgang til at skrive. Vær dog opmærksom på ved installation af software-pakker, der ikke kommer fra autoriserede leverandører, f.eks. public domain programmer, at deres installationsprogrammer ikke indeholder *Trojanske Heste*, prøv derfor at kigge dem igennem om muligt, eller prøv først at installere programmerne, uden at være superbruger, idet du da kan forminske chancen for at disse program-pakker ødelægger dit system totalt. Denne advarsel er ikke lavet for at skræmme folk væk fra at bruge public domain programmer, da der findes mange virkelig fortræffelige programmer af denne slags, men det sker desværre en gang i mellem at der er ondsindede programmer i omløb.



Oversigt over medlemsmøder i 1989

Dato	Sted	Tema
28/9	København	Distribuerede systemer.
11/10 †	Provinsen	Industrielle systemer.
27–28/11	København	Årsmøde med netindslag.

Det med † markerede møde er et eftermiddagsarrangement, som typisk er af ca. 2 timers varighed, placeret efter normal arbejdstid. Ditte møde er gratis. De øvrige møder er heldagsarrangementer.

Tid, sted og program for hver enkelt møder vil blive annonceret i DKUUG-Nyt forud for mødets afholdelse.