Laborationsrapport Projektarbete

Jack-Benny Persson

LX13

BSD & Solaris Grundkurs

2014-03-07

Jack-Benny Persson
LX13
BSD & Solaris Grundkurs
2014-03-07

Innehåll

1 Syfte

Utforska valfri del av FreeBSD och göra ett projekt av detta.

2 Mål

Till detta projekt väljer jag att göra en fördjupning om Jail och under projektet försöka skicka in ändringar och förbättringar till FreeBSD Handbook gällande just detta kapitel. Tanken är också att försöka hinna göra något om storage i FreeBSD, t.ex. någon om GEOM. Följande mål sätts alltså upp för detta projektarbete.

- Fördjupning inom Jails
- Fördjupning inom området storage
  - Snapshots (ej ZFS)
  - Quota
  - Kryptering
  - ZFS och ZFS i Jail
- Bidra till FreeBSD Handbook
- Testa FreeBSD på en Raspberry Pi

3 Genomförande

3.1 Jail

Jail blev jag intresserad av för länge sedan, och då ännu mer nu med föregående labb som handlade om just jails. Här såg jag även en chans att skicka in förbättringar till FreeBSD Handbook vilket hade varit hur coolt som helst om det gick vägen. Så av dessa anledningar väljer just Jail som projektlabb.

3.1.1 Bidrag till FreeBSD Handbook

Det första jag gör är att bli medlem i mailinglistan freebsd-doc. Detta görs via länken som finns i FreeBSD Handbook på förstasidan. Därefter laddar jag ner hela doc-trädet med **svn checkout http://svn.freebsd.org/doc /usr/doc**. Därefter läser jag FreeBSD's artikel "Contributing to FreeBSD" (http://www.freebsd.org/doc/en/articles/contributing/) om hur man bäst bidrar till de olika delarna av FreeBSD så som base, world, doc m.m. Precis som jag misstänkte så är det via de olika mailinglistorna man får skicka in sina bidrag. När det gäller dokumentationen är det helt ok att skicka in ren text som bidrag men det är alltid extra trevligt om man skickar en patch istället. Formatet på patcharna ska vara skapta med **diff -u** enligt denna artikel. När jag läser vidare så hittar jag dock att de gärna vill ha ändringar till dokumentationen genom "Problem-reports" med send-pr eller webben. Här hittar jag också att de vill ha patcharna som **svn diff** istället. Detta hittade jag i när jag läste vidare i "Contriubuting to FreeBSD". Det verkar lite rörigt ibland att bidra, men tar man det bara successivt så ska det nog gå att sätta sig in i det. Jag kollar också lite i arkivet för mailinglistan för att få en uppfattning om hur inläggen och patcharna brukar se ut. Under tiden som docs-trädet tankas hem via SVN så passar jag på att kika lite på hur Docbook-formatet ser ut och fungerar. FreeBSD Handbook är skriven i Docbook som sedan byggs till HTML, PDF eller något annat format man önskar. Allt detta sköts via en Makefile.

När jag testar att köra en **make** så misslyckas detta för att den en del dependiencies saknas som Makefilen kräver. Jag installerar porten/paketet *docproj* som är ett metapaket för att installera allt som krävs för att bygga dokumentationen. Jag testar nu **make** igen i handbokens katalog. Nu lyckas bygget och jag får HTML-filerna som output. Nu börjar jag göra mina redigeringar till handboken vilket tar ett tag. Varje gång man gör ett fel i formateringen och ska ändra det så måste man bygga om handboken med **make** vilket tar ett par minuter per gång. Efter ett tags redigerande är jag klar med min patch (bifogas labbrapporten som bilaga 1). Nu när patchen är skapad skickar jag in den till mailinglistan freebsd-doc@freebsd.org. Jag pratade med Victor om detta och han hade själv skickat in patchar till handboken via mailinglistan och det hade gått bra meddelade han. Jag skriver ett förklarande mail till vad det det gäller och bifogar patchen i mailet som jails.diff. Efter att ha läst på lite mer om jail.conf upptäcker jag dock att denna är relativt ny, den kom i FreeBSD 9.1. Av den anledningen borde ju det gamla stycket vara kvar, d.v.s. stycket om /etc/rc.conf. Jag skickar därför in en ny uppdaterad patch till mailinglistan med detta korrigerat. Jag passar också på att skriva ett inlägg till listan och fråga vilket det korrekta sättet är att skicka in patchar (via mailinglistan eller via send-pr). Den uppdaterade patchen bifogas rapporten som bilaga 2.

Kanske borde man eventuellt skicka in en patch även om att man måste skapa ett alias manuellt i host-systemet? Det är särkerligen många (inklusive mig själv) som annars lätt missar detta. Kanske även något om vad som händer när jailet t.ex. inte har SSH-demonen igång och man försöker logga in till jailet och då istället hamnar på host-maskinen och vad detta då beror på. Jag får se hur det går med min första patch, lyckas detta så skickar jag eventuellt in en patch till om detta.

En dag senare fick jag svar från mailinglistan vilket som är det rekommenderade sättet att skicka in patchar till handboken. Warren Block från listan meddelade att det bästa sättet är att skicka in dem via send-pr, av de anledningar som Johan G också nämnde, d.v.s. att då förs patchen in i en databas och glöms inte bort eller försvinner lika lätt. Jag skickar därför in min uppdaterade patch (Bilaga 2) via den webbaserade send-pr. När jag postat min send-pr dyker den dessutom automatiskt upp i freebsd-doc mailinglistan, alltså är ett inlägg i listan helt onödig, då en send-pr både skapar en rapport *samt* skickar den till listan. Det korrekta sättet att bidra och skicka in patchar till FreeBSD's dokumentation är alltså att använda sig av send-pr. Efter att man skickat in sin send-pr får man även en länk i ett mail till rapporten så att man kan följa hur det går, d.v.s. om den blir accepterad, nekad eller förblir öppen. URL:en för min buggrapport är http://www.freebsd.org/cgi/query-pr.cgi?pr=187142

3.1.2 ZFS och Jail

Jag använder här samma jail som jag tidigare byggde upp i föregående labb om jails. Till denna labb försöker jag nu tilldela ett ZFS dataset. Först och främst börjar jag med att skapa ett nytt dataset ur poolen haxx som heter jailset som då ska tilldelas jailet *lab1*. Detta var inte det enklaste, det finns i princip ingen dokumentation tillgänglig för hur man åstadkommer detta. Det blir därför mycket läsande i manualsidorna och en hel del trial-and-error. Den lilla dokumentation man kan hitta på nätet är mest i form av poster i mailinglistor m.m. och då oftast hur det inte fungerar. Det första som måste göras för att överhuvudtaget kunna se dataseten och poolerna i jailet är att sätta parametern *jailed=on* på datasetet. För att sedan kunna se datasetet med **zfs list** i jailet måste man "jaila" ZFS-datasetet med **zfs jail <jailnr> <pool>/<dataset>**. I mitt fall blir detta **zfs jail 9 haxx/jailset**. I /etc/jail.conf behöver man också sätta en del nya parametrar. Nedan är de parameterar jag sätter för mitt jail lab1.

```
lab1 {
     host.hostname = lab1;
```

```
    ip4.addr = 192.168.122.61;  # IP address of the jail
    path = "/jails/lab1";
    devfs_ruleset = 4;
    mount.devfs;
    exec.start = "/bin/sh /etc/rc";
    exec.stop = "/bin/sh /etc/rc.shutdown";
    allow.raw_sockets;
    allow.mount;
    allow.mount.devfs;
    allow.mount.zfs;
    enforce_statfs=0;
}
```

Därefter startar jag om jailet lab1 från hosten med **service jail stop lab1** följt av **service jail start lab1**. Därefter måste man "jaila" ZFS-datasetet igen eftersom jailet nu får en ny *JID*. Kommandot **jls** visar vilket nummer jailet fått. Numret brukar öka med ett varje gång man stänger av/startar upp jailet. Så i mitt fall är det dags att köra **zfs jail 10 haxx/jailset** nu. Därefter testar jag att mounta mitt dataset i jailet. I jailet som root skriver jag nu **zfs mount haxx/jailset**. Nu lyckas monteringen och jag gör en **df -h** och datasetet dyker upp. Dock ser det lite märkligt ut här, se nedan.

```
root@lab1:/ # df -h
Filesystem      Size    Used   Avail Capacity  Mounted on
/dev/ada0p2     3.9G    2.1G    1.4G    60%    /
devfs           1.0K    1.0K      0B   100%    /dev
/dev/ada0p3     9.7G    8.6G    346M    96%    /usr
/dev/ada0p4     4.7G    1.3G    3.0G    31%    /home
/dev/ada7p1     5.8G    585M    4.8G    11%    /jails
haxx            5.8G     50K    5.8G     0%    /haxx
svn             2.1G     32K    2.1G     0%    /svn
svn/docs        9.8G    7.7G    2.1G    78%    /svn/docs
devfs           1.0K    1.0K      0B   100%    /jails/lab1/dev
haxx/jailset    5.8G     50K    5.8G     0%    /jails/lab1/haxx/jailset
```

Här ser man alltså alla pooler och dataset från host-systemet. Dock är det bara datasetet *jailset* som är mountad. Och då inte under /jails/lab1/haxx/jailset, utan under /haxx precis som i host-systemet även om df -h visar annat. Likaså är INTE svn/docs monterad även om df -h visar detta. Nedan visas en **ls -l** / av jailet, här ser man att allt inte är monterat även om det sägs så i df-outputen.

```
root@lab1:/ # ls /
.cshrc          boot            lib             rescue          usr
.profile        dev             libexec         root            var
.rnd            etc             media           sbin
COPYRIGHT       haxx            mnt             sys
bin             home            proc            tmp
```

Jag testar även att skriva några filer till datasetet inifrån jailet vilket fungerar bra. För att repetera stegen och försöka skala bort steg och konfigurationer från en eventuell guide hur man går tillväga tas en inställning i taget bort från /etc/jail.conf. Jag börjar helt enkelt med att

låta jail.conf vara oförändrad tillsvidare och bara starta om jailet och därefter se om vad som händer om man inte ger kommandot **zfs jail 10 haxx/jailset** efter omstarten. Nu är datasetet *jailset* fortfarande tillgängligt och monterad redan vid start av jailet dock visar **zfs list** att inga dataset är tillgängliga. I host-systemet syns haxx/jailset i ZFS listan, men däremot är inte jailset monterat, vilket är bra. Jag testar också om jag kan skriva till datasetet i jailet vilket jag kan. Jag testar att göra en **zfs jail 10 haxx/jailset** i hosten och nu dyker datasetet upp i jailet. Detta känns som en liten inkonsekvens. Utan **zfs jail** så fanns inga dataset i jailet, trots det så automonterades datasetet vid start av jailet. Det känns som att jag behöver sätta upp ett helt nytt jail för att sakta men säkert testa mig fram till vilka steg som behövs och vilka rättigheter som jailet behöver för att det ska fungera.

Ett nytt jail, lab2. Ett helt nytt jail kallat lab2 sätts upp för att försöka göra en så ren setup som möjligt med jail och ZFS. Kommandona för detta blir

```
root@bsblad:~ # setenv D /jails/lab2
root@bsblad:~ # mkdir -p $D
root@bsblad:~ # cd /usr/src
root@bsblad:/usr/src # make installworld DESTDIR=$D
root@bsdlab:/usr/src # make distribution DESTDIR=$D
root@bsdlab:/usr/src # mount -t devfs devfs $D/dev
```

Därefter aktiverar jag det nya jailet i /etc/rc.conf genom att lägga till *lab2* direkt efter *lab1* på jails-raden. Detta är en space-separated lista av jails som ska aktiveras. Därefter lägger jag till en grundläggande konfiguration av det nya jailet i /etc/jail.conf

```
lab2 {
     host.hostname = lab2;
     ip4.addr = 192.168.122.62;  # IP address of the jail
     path = "/jails/lab2";
     devfs_ruleset = 4;
     mount.devfs;
     exec.start = "/bin/sh /etc/rc";
     exec.stop = "/bin/sh /etc/rc.shutdown";
}
```

Därefter skapar jag ett nytt dataset som ska bli tillgängligt i *lab2* som heter *lab2set*. Datasetet skapas i poolen *haxx* som redan finns på host-maskinen. Den första extra inställningen som läggs till i /etc/jail.conf för lab2 är *allow.mount.zfs*. Denna krävs enligt manualen för att det ska gå att montera ett ZFS-dataset i ett jail. Vid vidare läsning i manualen hittar jag här att just denna parameter är beroende av *allow.mount* samt att *enforce_statfs* är satt till ett lägre värde än 2. Vad är då enforce_statfs? Enligt manulen är detta en parameter som styr hur mycket information jailet kan få om monteringspunkter i host-systemet. Ett värde av 0 ger jailet tillgång till all information om alla monteringspunkter i hosten. Ett värde på 1 ger bara tillgång till de monteringspunkter som är under chroot-katalogen, d.v.s jail-katalogen. Ett värde på 2 (default) innebär att jailet bara kan komma åt filsystem och monteringspunkter som är direkt innuti den katalog jailet befinner sig. Jag testar med 1 här först då jag inte är helt på det klara med om ZFS-dataset som har jail-flaggan "magiskt" dyker upp under rätt katalog eller inte. De parametrar jag lägger till i jail.conf under lab2 är alltså

```
allow.mount;
allow.mount.zfs;
enforce_statfs = 1;
```

Jag startar nu upp jailet med **service jail start lab2** och kör därefter **jls** för att få reda på JID. Jag loggar därefter in i det nya jailet med **jexec <JID> tcsh**. Jag gör först en **df -h** när jag loggat in och ser att jag faktiskt ser alla filsystem från hosten trots att jag satte en etta på *enforce_statfs*. En **zfs list** visar dock en tom lista vilket ju är korrekt eftersom jag ännu inte satt jail-parametern på datasetet och inte heller kört **zfs jail <JID> haxx/lab2set** ännu. Jag börjar därför att sätta jail-parametern på datasetet med **zfs set jailed=on haxx/lab2set**. Därefter kontrollerar jag i ZFS listan på jailet om det går att se *haxx/lab2set* men det gör det ännu inte. Jag kör därför **zfs jail 11 hax/lab2set** nu i hosten. Därefter kontrollerar jag igen ZFS-listan i jailet och nu syns den faktiskt i listan. Jag testar nu också att montera datasetet i jailet med **zfs mount haxx/lab2set** vilket lyckas! Frågan är ju nu hur man ska kunna få datasetet att monteras vid uppstart av jailet. Efter lite läsning i manualen för ZFS så kan man tydligen köra **zfs jail <jailname> pool/dataset**, d.v.s. att använda namnet på jailet istället för dess JID. Jag testar detta genom att först stänga ner jailet och sedan köra **zfs unjail 11 haxx/lab2set** och sedan istället köra **zfs jail lab2 haxx/lab2set**. Jag får då bara *invalid jail id or name*. Kanske är det så att jailet måste vara igång för att det ska gå att sätta detta? Jag testar att första starta upp jailet och sedan köra kommandot med namnet istället för JID och se om det håller i sig mellan omstarter av jailet. Detta fungerar till viss del. Efter en omstart av jailet går det inte längre att se datasetet med **zfs list** men däremot är datasetet monterat i /haxx/lab2set och fungerar att använda. För att säkerställa att det verkligen är ZFS-datasetet som är monterat i /haxx/lab2set skriver jag en rad större filer i /haxx/lab2set i jailet och kontrollerar så att använt diskutrymme i datasetet i hosten växer, vilket det gör. Det är alltså helt korrekt haxx/lab2set som blev monterat när jailet startades upp. Notera dock att jag har lagt till *zfs_enable="YES"* i jailets /etc/rc.conf. Något som också krävs för att det ska fungera är DevFS-regeln *add path zfs unhide*. Denna finns dock redan som default i regel 4 i /etc/defaults/devfs.rules.

3.2 Storage

Här tänker jag framförallt fördjupa mig lite i UFS & UFS2 och lite i de funktioner som FreeBSD tillhandahåller vad gäller t.ex. quota och även testa någon GEOM-modul som här blir geli eller GEOM-ELI som modulen heter. Jag tänkte även bygga en ZFS-pool av fysiska diskar (USB-minnen) med ett komprimerat dataset.

3.2.1 UFS snapshots

Här testar jag att göra snapshots i UFS. Jag testar detta på min /jails-disk som jag gjorde som UFS. Jag läser i manulen att detta kan göras på två sätt, dels med **mount** och dels med **mksnap_ffs**. Jag börjar med att testa det senare. Detta görs med **mksnap_ffs /jails/snapshot1** där snapshot1 är namnet på snapshoten. Jag undersöker nu snapshot-filen med **file snapshot1** och **du -sh snapshot1**. Filen är 2,3MB stor direkt efter skapande och file-kommandot visar: *Unix Fast File system [v2] (little-endian) last mounted on /jails, last written at Fri Feb 28 19:27:23 2014, clean flag 1, readonly flag 1, number of blocks 1572855, number of data blocks 1522510, number of cylinder groups 10, block size 32768, fragment size 4096, average file size 16384, average number of files in dir 64, pending blocks to free 0, pending inodes to free 0, system-wide uuid 0, minimum percentage of free blocks 8, TIME optimization.*

Inför testet med **mount**-kommandot stänger jag först ner mina jails för att sedan kunna skriva en ny fil i katalogen efter att jag gjort min snapshot och sedan göra en rollback för att se vad som händer. Därefter görs en ny snapshot med **mount -u -o snapshot /jails/snapshot2 /jails**. Snapshoten skapas direkt. Efter att läst på lite så verkar det inte finnas någon enkel "rollback" så som det finns för ZFS utan man får manuellt montera snapshoten och sedan flytta

över de filer man vill eräsätta eller använda. Jag testar därför att montera min nyligen skapade snapshot *snapshot2* med **mdconfig -a -t vnode -f /jails/snapshot2 -u 4** och **mount -r /dev/md4 /mnt**. mdconfig är ett kommando för minnesdiskar, där flaggan *-a* "attach" en ny minnesdisk, *-t* är typen som här är vnode, d.v.s. en virtuell nod. Flaggan *-f* talar om var filen finns som vi ska montera och flaggan *-u* talar om vilket enhetsnummer den nya minnesdisken ska få, här nummer 4, istället för ett slumpmässigt nummer. Sedan monteras snapshoten med mount i read-only läge. **umount /mnt** avmonterar snapshoten igen och **mdconfig -d -u 4** detachar enhet 4 igen från systemet.

3.2.2 Quota

Då quota ingick i föreläsningarna men inte i någon labb så bestämmer jag mig för att labba lite med quota här. Då jag satt upp min maskin *bsdlab* med en separat /home partition blir denna utmärkt för att testa quota på. Jag börjar med att skapa två nya användare som heter *kalle* och *lisa* som båda är medlemmar i gruppen *testgrp* för att kunna testa även grupp-quota. Detta görs med följande kommandon

```
root@bsdlab:~ # pw groupadd testgrp
root@bsdlab:~ # pw useradd kalle -d /home/kalle -m -g testgrp
root@bsdlab:~ # pw useradd lisa -d /home/lisa -m -g testgrp
```

Därefter aktiveras quota i /etc/rc.conf genom att lägga in raden *quota_ enable="YES"* och därefter modifieras /etc/fstab så att raden för /home blir

```
/dev/ada0p4     /home           ufs     rw,userquota,groupquota         2       2
```

Nu startas maskinen om för att aktivera quota-stödet och skapa tomma quota-filer. Därefter loggar jag in som både kalle och root i varsin terminal. Båda användarna kontrollerar sin quota med **quota -v** som är "none" för båda användare då jag ännu inte lagt in någon quota. Nu skapas däremot en quota för kalle genom att som roota starta redigeraren för quota med **edquota -u kalle**. Här sätts följande quota:

```
Quotas for user kalle:
/home: in use: 0k, limits (soft = 500k, hard = 2048k)
        inodes in use: 0, limits (soft = 5, hard = 10)
```

Därefter kollar användaren kalle sin quota och nu visas exakt den quota som angavs i filen.

```
$ quota -v
Disk quotas for user kalle (uid 1005):
Filesystem        usage    quota   limit   grace  files   quota  limit   grace
/home                 0      500    2048              0       5     10
```

Jag testar nu att skapa lite olika stora filer för kalle, först en på 600kb så att jag ska nå soft-limit, sedan en till på 1500kb så att jag slår i taket på hard-limit. Sedan tas filerna bort och jag testar istället att skapa små filer för att testa inode-quotan. Allt fungerar direkt! När användaren har nått upp till sin softlimit så visas även grace-perioden som i det här fallet är default 7 dagar.

Gruppquota måste också testas. Detta ställs in med **edquota -g testgrp**. Filen ser ut enligt nedan när redigeringen är klar

```
Quotas for group testgrp:
/home: in use: 2044k, limits (soft = 10000k, hard = 20000k)
        inodes in use: 16, limits (soft = 20, hard = 50)
```

De siffror man ser för "in use" är alltså de utrymme som kalle redan använt i sin hemkatalog. D.v.s. gruppquotan är precis vad det låter som, en quota för hela gruppen. Jag testar att att logga in som lisa och och kontrollera quotan och skriva några filer och kontrollera igen så att allting fungerar. Kontrollen för userquotan görs med **quota -v** som vanligt och kontroll för gruppquotan görs med **quota -g**. Här visas exakt de siffrorna som visades i redigeringen av quotan. Userquotan visar dock att lisa inte har någon quota satt för sig. Jag testar nu att skriva några stora filer i lisas hemkatalog. Jag får då genast upp en varning att grupp-quotan har överskridits.

```
$ mkfile 10mb testfile1
/home: warning, group disk quota exceeded
```

Allting fungerar således även med grupp-quotan och det var enkelt att sätta upp.

3.2.3 Kryptering med geli

Här följer jag handboken ganska noga då jag inte har någon som helst tidigare erfarenhet av kryptering av partitioner och diskar i FreeBSD. Först och främst skapar jag en ny KVM-disk som jag kan använda för att experimentera med geli på utan att riskera att förstöra några av de andra diskarna och partitionerna. Den nya disken dyker upp som /dev/ada9 i systemet. Nu aktiveras GEOM modulen eli genom att lägga till *geom_ eli_ load="YES"* i filen /boot/loader.conf. Därefter startas maskinen om igen. En listning av laddade moduler med **kldstat** visar nu att *geom_ eli.ko* är laddad. När det är verifierat att modulen är laddad så är nästa steg att skapa nyckeln och initalisera disken. Här används en sektorstorlek på 4kb enligt handbokens rekommendationer för att få bättre prestanda.

```
root@bsdlab:~ # dd if=/dev/random of=ada9.key bs=64 count=1
root@bsdlab:~ # geli init -s 4096 -K ada9.key /dev/ada9
```

init är för att initalisera disken, *-s 4096* är för att sätta sektorstorleken till 4096 bytes, *-K ada9.key* anger att nyckeln ada9.key ska användas och till sist så anges */dev/ada9* som är disken. Här får man också ange ett lösenord som ska anges vid montering av disken. Nu ska det gå att attacha disken med nykeln och lösenordet som angavs vid geli init-kommmandot. Detta görs med **geli attach -k ada9.key /dev/ada9**. Här får man då ange lösenordet som angavs tidigare. Nu ska en ny enhet dyka upp i /dev som heter ada9.eli vilket också sker. Nu skapas ett nytt filsystem på eli-disken med **newfs /dev/ada9.eli** och därefter monteras den med **mount /dev/ada9.eli /mnt** vilket fungerar. Lite filer skapas nu på disken och disken avmonteras och "detachas" med **umount /mnt** och **geli detach /dev/ada9.eli**.

Det går att automatisera attach-steget genom rc.conf genom att lägga att in följande

```
# Krypterade diskar
geli_devices="ada9"
geli_ada9_flags="-k /root/ada9.key"
```

Datorn startas nu om och vid uppstart blir jag tillfrågad att ange lösenordet för disken. Väl inloggad försöker jag montera filsystemet men ada9.eli finns ej. **dmesg** visar att strax efter att den blivit attachad blir den detachad igen. Jag googlar lite på det och hittar i FreeBSD forumet att för att den ska förbli attachad måste man också montera den direkt med /etc/fstab. Jag skapar därför en ny entry i min /etc/fstab som innehåller:

```
/dev/ada9.eli   /mnt            ufs     rw      0       0
```

En ny omstart av maskinen görs och nu monteras mycket riktigt /dev/ada9.eli under /mnt och **dmesg** visar inte längre någon detach av ada9.eli. Ingenting om detta nämns i handboken så här skriver jag en lite "note" till detta i handboken och skickar in som en patch med den webbaserade send-pr. Patchen bifogas rapporten som bilaga 3.

Något som är viktigt att tänka på här att ta en off-site backup av nyckeln *ada9.key*. Försvinner denna så går det inte att montera filsystemet igen och all data kommer vara förlorad.

3.2.4 ZFS komprimering

Detta blir ett litet kul experiment jag funderat på under en tid; att använda fyra st USB-minnen i en RaidZ-1 med komprimering på poolen. Dels så förlorar man ytterst lite diskutrymme eftersom tre minnen används för lagring och endast ett för paritet, d.v.s 3/4 är användbart utrymme. Dels så kommer komprimeringen att göra att man får ytterligare utrymme som går att använda. Jag läste också på en av Oracles bloggar att det faktiskt även kommer att ge en viss prestandaförbättring eftersom det är snabbare för CPU:n att komprimera data än vad det är för t.ex. mekansika diskar och USB-minnen att skriva den faktiska datan. Utöver detta så har man säkerheten av Raid-5, d.v.s. en av diskarna kan krascha och poolen överlever ändå. Till detta projekt använder jag fyra st 8GB Sandisk minnen som tyvärr är relativt långsamma, men som trots allt duger till projektet. Först och främst skapas poolen som vanligt med **zpool create tank raidz /dev/da0 /dev/da1 /dev/da2 /dev/da3**. Därefter skapas ett dataset, också precis som vanligt med **zfs create tank/comp**. Nu aktiveras komprimeringen på datasetet med **zfs set compression=gzip tank/comp**. Här finns ett antal val mellan olika kompressionsalgoritmer, jag väljer gzip här då det är bekant och verkar vara lagom med tanke på datorns hastighet etc. Nu testas först så att själva komprimeringen fungerar genom att skriva en fil med bara zeros som då borde komprimeras väldigt mycket då denna form av data blir enkel att komprimera. Detta görs med att i /tank/comp skriva kommandot **dd if=/dev/zero of=testfile1 bs=1M count=100**. Här går det undan som bara den, den 100MB stora filen skrivs på 0,25 sekunder. En **du -sh** visar att filen bara tar upp 512 bytes på disken. För att verifiera att filen verkligen i sig är 100MB stor kopierar jag över den till den root's hemkatalog som är en vanlig UFS-partition. Jag passar på att ta tiden på överföringen här också för skojs skull. Detta tog 2,8 sekunder. En **du -sh** i roots hemkatalog visar att filen här är 100MB stor. Komprimeringen fungerar alltså. Jag tankar nu hem en ISO-fil på ca 622MB. Hastigheten jag tankar ner i ligger på 10MB per sekund, alltså mer än vad USB-minnena egentligen klarar av. Alltså hjälper både komprimeringen och ZFS till här att snabba upp hastigheten. När väl filen är nedladdad landar den på en storlek på disken som är 411MB stor, komprimeringen har alltså sparat ca 210MB åt oss. Därefter checkar jag ut docs/head från FreeBSD's dokumentation vilket är på ca 218MB. I det komprimerade datasetet blir katalogen istället bara på 160MB, alltså har jag här sparat ca 60MB. En **zfs list** visar nu

```
NAME         USED  AVAIL  REFER  MOUNTPOINT
tank         572M  21.3G  44.9K  /tank
tank/comp    572M  21.3G   572M  /tank/comp
```

D.v.s. datasetet har bara använt 572MB trots att jag har faktiskt har lagrat hela 940MB om man räknar med dummy-filen som endast består av nollor. Räknar man bort den har jag ändå lagrat 840MB "riktig" data i datasetet men bara använt 572MB. Dessutom har komprimeringen snabbat upp minnena en del. Och jag har säkerheten av RAID-5. Med ZFS kan man även kontrollera sin compressions ratio med **zfs get compressratio**. I mitt fall visar detta

```
root@aspire:~ # zfs get compressratio tank/comp
NAME       PROPERTY       VALUE  SOURCE
tank/comp  compressratio  1.90x  -
```

Här hade man säkert kunna pressa ut ännu bättre kompression i utbyte mot prestanda. T.ex. kan man på just gzip sätta kompressionsnivån från 1-9.

3.3 FreeBSD på Rasberry Pi

Jag gör också en del testar med FreeBSD på en Raspberry Pi. Det finns numera en färdig port för FreeBSD på Raspberry Pi redo att tanka hem och installera. Länken till Raspberry porten är https://wiki.freebsd.org/FreeBSD/arm/Raspberry%20Pi. Själva installationen av FreeBSD imagefilen gick enkelt och smärtfritt, bara att använda **dd** för att föra över den till sitt SD-kort och sedan boota upp Rasperryn med SD-kortet. På andra omstarten så utökas också root-partitionen automatiskt till att sträcka sig över hela SD-kortet, i mitt fall 16gb. Därefter försöker jag installera några program men upptäcker snart att det inte finns några som helst färdigkompilerade binärer att tanka hem till ARM-plattformen. Således måste allt byggas från ports. Jag börjar med att tanka hem hela ports-trädet med **portsnap fetch** följt av **portsnap extract**. Och när detta är klart (vilket tar väldigt lång tid) cd:ar jag in till /usr/src och hämtar indexet med **make fetchindex** för att ports-trädet ska bli sökbart.

Jag försöker nu få ZFS att fungera som ett kul experiment även om Raspberry egentligen har för lite RAM för ZFS (och dessutom delar buss till USB och ethernet). Dock fungerar detta inte alls och jag får bara felmeddelandet *failed to initalize ZFS library*. Jag söker lite på det på nätet och hittar en post om att det ska gå att få tillbaks ZFS-stödet genom att kompilera om kärnan med *WITH_ZFS="YES"* i /etc/make.conf. Så nästa steg blir att hämta hem källkoden till kärnan och userland med SVN. Subversion finns inte installerat per default och finns inte heller som ett paket så här blir det återigen dags att installera via ports. För att kunna gå ifrån terminalen utan att behöva svara på alla dialogrutorna under kompileringens tid så kör jag först **make config-recursive**. Efter att alla frågor är besvarade kör jag **make install clean** som vanligt i katalogen (/usr/ports/devel/subversion). När kompileringen är klar av Subversion och alla dependencies så checkar jag ut källkodsträdet med **svn checkout svn://svn0.eu.FreeBSD.org/base/releng/10.0 /usr/src/**. Detta tar väldigt lång tid, alldeles för lång tid för att vara normalt. Jag undersöker därför lite närmre och upptäcker snart att det är något galet med nätverksdrivrutinen. Hur jag än gör kommer jag inte över en hastighet av ca 130kb/s. När källkodsträdet väl är hämtat kompilerar och installerar jag **screen** först så att jag kan deatcha terminalen över natten medans kompileringen sker.

Jag går nu vidare med att sätta *WITH_ZFS="YES"* i /etc/make.conf och kompilerar sedan kärnan på vanligt vis och installerar den som vanligt. Jag blir lite osäker på om kärnan installerarats som den skulle då ARM-porten inte har den vanliga boot-screenen men en **uname -a** visar att jag faktiskt kör min egna kärna, kallad *MYPI*. Dock så fungerar fortfarande inte ZFS i FreeBSD under Raspberry Pi vilket var lite tråkigt. Men det var iaf ett kul projekt att köra FreeBSD på Raspberryn.

4 Reflektioner

Detta har varit ett riktigt äventyr med många spännande moment och många aha-upplevelser. Att skicka patchar till FreeBSD's doc-projekt var nog det mest spännande av allt. Jag hoppas att mina bidrag blir accepterade, men i vilket fall som helst så har man blivit en lärdom rikare. Att skicka in patchar till FreeBSD krävde en hel del läsande innan det var dags att skrida till verket. Man fick lära sig lite DocBook, hur man bygger dokumentationen från "källkoden" som i det här fallet är XML DocBook, man fick lära sig lite om SVN, hur man skapar en diff och hur man sedan faktiskt på rätt sätt skickar in sin patch via send-pr. Det var en hel del men som trots allt gick ganska lätt att sätta sig in i. Snapshot och quota var busenkelt att sätta upp jämfört med hur det var i Linux när vi labbade med quota där. I Linux krävdes det mycket mer meckande och dessutom installation av extra programvara, i FreeBSD finns allting redan där direkt från början. Och dessutom mycket enklare och smidigare konfig-filer än vad Linux har. Även att skapa krypterade diskar var förvånansvärt enkelt. Under hela denna kursen har jag blivit förvånad över hur enkelt och smidigt allt fungerar. Konfig-filer, kommandon, moduler, kärn-kompilering och allt känns väldigt straight-forward jämfört med Linux där mycket känns som "fulhack" av olika slag.

Just när det gäller detta projektarbetet var det svårt att avgränsa sig på ett specifikt ämne då det fanns så mycket jag ville "prova på" som man annars inte hade tagit sig tid till. Jag hoppas att det var ok att istället testa och fördjupa sig inom flera olika ämnen istället för bara ett enda.

A Bilagor

A.1 Bilaga 1, första patchen

```
Index: en_US.ISO8859-1/books/handbook/jails/chapter.xml
===================================================================
--- en_US.ISO8859-1/books/handbook/jails/chapter.xml (revision 44074)
+++ en_US.ISO8859-1/books/handbook/jails/chapter.xml (working copy)
@@ -352,15 +352,22 @@

       <step>
  <para>For each jail listed in <varname>jail_list</varname>, a
-  group of &man.rc.conf.5; settings, which describe the
-  particular jail, should be added:</para>
+   group of &man.jail.conf.5; settings, which describe the
+   particular jail, should be added to /etc/jail.conf:</para>

-<programlisting>jail_<replaceable>www</replaceable>_rootdir="/usr/jail/www"     # jail's root directory
-jail_<replaceable>www</replaceable>_hostname="<replaceable>www</replaceable>.example.org"  # jail's hostname
-jail_<replaceable>www</replaceable>_ip="192.168.0.10"           # jail's IP address
-jail_<replaceable>www</replaceable>_devfs_enable="YES"          # mount devfs in the jail
-jail_<replaceable>www</replaceable>_devfs_ruleset="<replaceable>www_ruleset</replaceable>" # devfs ruleset to apply to jail</programlisting>
+<programlisting>
+<replaceable>www</replaceable> {                                    # Name of the jail from jail_list in /etc/rc.conf
+     host.hostname = <replaceable>www</replaceable>.example.org;        # Hostname
+     ip4.addr = 192.168.0.10;                # IP address of the jail
+     path = "<replaceable>/usr/jail/www</replaceable>";                 # Path to the jail
+     devfs_ruleset = <replaceable>4</replaceable>;                      # DevFS ruleset
+     mount.devfs;                            # Mount devfs inside the jail
+     exec.start = "/bin/sh /etc/rc";         # Start command to run
+     exec.stop = "/bin/sh /etc/rc.shutdown"; # Stop command to run to stop the jail
+}
+</programlisting>

+
  <para>The default startup of jails configured in
    &man.rc.conf.5;, will run the <filename>/etc/rc</filename>
    script of the jail, which assumes the jail is a complete
```

A.2 Bilaga 2, uppdaterad patch

```
Index: en_US.ISO8859-1/books/handbook/jails/chapter.xml
===================================================================
--- en_US.ISO8859-1/books/handbook/jails/chapter.xml (revision 44074)
+++ en_US.ISO8859-1/books/handbook/jails/chapter.xml (working copy)
@@ -373,6 +373,25 @@
    <para>For a full list of available options, please see the
      &man.rc.conf.5; manual page.</para>
  </note>
+
+<note>
+<para>From FreeBSD 9.1 and above the recommended way is to place the jail settings in <filename>/etc/jail.conf</filename> (see &man.jail.conf.5;).
+For example a jail config similar to above would look like this in <filename>/etc/jail.conf</filename>.
+<programlisting>
+<replaceable>www</replaceable> {                                    # Name of the jail from jail_list in /etc/rc.conf
+     host.hostname = <replaceable>www</replaceable>.example.org;        # Hostname
+     ip4.addr = 192.168.0.10;                # IP address of the jail
+     path = "<replaceable>/usr/jail/www</replaceable>";                 # Path to the jail
+     devfs_ruleset = <replaceable>www_ruleset</replaceable>;            # DevFS ruleset
+     mount.devfs;                            # Mount devfs inside the jail
+     exec.start = "/bin/sh /etc/rc";         # Start command to run
+     exec.stop = "/bin/sh /etc/rc.shutdown"; # Stop command to run to stop the jail
+}
+</programlisting>
+</para>
+</note>
+
+
       </step>
     </procedure>
```

A.3 Bilaga 3, patch till disks-avsnittet ang. geli

```
Index: en_US.ISO8859-1/books/handbook/disks/chapter.xml
===================================================================
--- en_US.ISO8859-1/books/handbook/disks/chapter.xml    (revision 44144)
+++ en_US.ISO8859-1/books/handbook/disks/chapter.xml    (arbetskopia)
@@ -3164,6 +3164,20 @@
          the <command>geli</command> provider from the kernel before
          the system shuts down.</para>

+       <note><para>
+         During boot the system will prompt for the passphrase to
+         attach the provider. If the correct password is given the
+         system will attach the provider. But, if the provider is not
+         mounted by an entry in <filename>/etc/fstab</filename> the
+         provider will be detached again.
+         So, in order to keep the provider attached an entry in
+         <filename>/etc/fstab</filename> is necessary. See
+         &man.fstab.5; for more information.
+       </para></note>
+
+
+
+
        <para>More information about configuring
          <filename>rc.d</filename> is provided in the
          <link linkend="configtuning-rcd">rc.d</link> section of the
```