Háttértár Rendszerek Mérési Segédlet

Informatikai technológiák laboratórium I.
Összeállította: Tóth Dániel
Méréstechnika és Információs Rendszerek Tanszék

2009. március 25.

1. Háttértár alapok

1.1. Gyakorlati elvárások

A *háttértár* vagy *másodlagos tár* feladata a számítógépes rendszerekben a stabil adattárolás megvalósítása. Az *operatív memóriával*, azaz *elsődleges tárral* szemben a legfontosabb a **nem felejtő** (non-volatile) tulajdonsága, vagyis a rendszer kikapcsolása után is hosszú ideig megőrzi a rajta tárolt adatokat. Fontos jellemzője továbbá a memóriához képest alacsony **fajlagos költsége** és a nagyobb **kapacitása**.

A háttértárakkal szemben támasztott **elsődleges funkcionális követelmény** tehát a nem felejtő adattárolás, ami röviden annyit jelent, hogy a tárolóba írt adatot a későbbiekben pontosan vissza lehet olvasni. A gyakorlatban számos *„nemfunkcionális"* (más szóval *extrafunkcionális*, azaz az alapvető funkcionalitáson túli) elvárást is támasztunk a háttértárakkal szemben. Például:

- **Hozzáférési idő** - mennyi idő telik el egy írás vagy olvasási kérés kiadása és kiszolgálása között.
- **Áteresztőképesség** - adott időegység alatt olvasható illetve írható adatmennyiség.
- **Fogyasztás** - az olvasás- és írásműveletek illetve a tétlen működés energiaigénye.

Háttértárak meghibásodása szigorúan véve azt jelenti, hogy nem képes kiszolgálni olvasás vagy írásműveleteket, illetve az írt adatokat nem képes pontosan visszaolvasni.

Sokszor meghibásodásról beszélünk akkor is, ha a tárolóeszköz nemfunkcionális paraméterei olyan mértékben leromlanak (pl a műveletek nagyon lelassulnak), hogy az a rendszerünk normális működését akadályozza.

A gyártók szokásosan megadják az eszközök várható meghibásodási gyakoriságát, illetve tervezett élettartamát. Fontos megjegyezni, hogy megadott meghibásodási paramétereket mindig a *tervezett élettartamon* (design lifetime) belül kell érteni. Tehát ha a gyártó a *meghibásodások közötti átlagos időre* (MTBF, Mean Time Between Failures) 1000000 üzemórát specifikál, akkor az nem jelenti azt, hogy egy eszköz várhatóan 114 évig üzemképes lesz. A helyes értelmezés szerint az 5 éves tervezett élettartam lejárta előtt várhatóan 22 darabból 1 fog meghibásodni. A könnyebb értelmezhetőség végett néhány gyártó áttért az MTBF-ről az *éves meghibásodási gyakoriság* (annualized failure rate) megadására.

A tárolóeszközök költsége általában eltörpül a rajtuk tárolt adat értékéhez képest, ezért különösen nagy problémát jelentenek az adatvesztéssel járó meghibásodások. Az adatvesztések kockázatát redundáns tárak kiépítésével lehet mérsékelni.

Nagyvállalati környezetben számottevő költséget jelent a **háttértárak karbantartása**. Ez nemcsak a meghibásodott, illetve életciklusuk végére ért eszközök cseréjét jelenti. Az alkalmazások növekvő teljesítmény és tárkapacitás igényei miatt is gyakran szükség van cserére, illetve bővítésre. A karbantartás egyszerűsítésére jelentek meg a központosított tárrendszerek.

1.2. Adattároló eszközfajták és tulajdonságaik

Merevlemez Manapság a legelterjedtebben alkalmazott tárolóeszköz. Az adatokat mágnesesen rögzíti forgó lemeztányérok felületén. A lemez tartalmát egy pozicionálható fejszerelvény olvassa és írja. Alapvetően mechanikus eszköz, ennek következtében a véletlen hozzáférési idők közepesek, tipikusan 5-20 ms nagyságrendűek, az áteresztőképesség is nagyságrendileg közepesen gyors, jellemzően 50-130 MB/s. A nagy sebességgel mozgó alkatrészek, valamint a nagy adatsűrűség miatt a fej és a tányér közötti rendkívül kis légrés igen sérülékennyé teszi. Külső erőhatások, nagy gyorsulás (ütközés), magas (> 50 ℃) vagy éppen alacsony (< 20 ℃) hőmérséklet, illetve a hőmérséklet, légnyomás, páratartalom gyors változása mind adatvesztéssel járó üzemzavart okozhat.

A merevlemezek jellemző paraméterei: lemezek átmérője (jellemző változatok: 3.5”, 2.5”, 1.8”), lemeztányérok fordulatszáma (jellemző értékek: 4200/min, 5400/min, 7200/min, 10000/min, 15000/min), lemezek száma (tipikusan 1-5), lemezenkénti kapacitás (manapság 80GB/lemeztől 500GB/lemezig terjed)

A hozzáférési idők alapvetően két összetevőből állnak össze: fejmozgatási idő és lemez aláfordulási idő. A fejmozgatási idő függ attól, hogy a véletlen elérések mekkora távolságra történnek, az aláfordulási idők azonban 0 és egy teljes lemezfordulat ideje között véletlenszerűen oszlik el. A gyors fejmozgatás nagyobb zajjal és energiafogyasztással jár, továbbá nagyobb teherbírású mechanikát igényel. Az átlagos aláfordulási idő csak a fordulatszám növelésével csökkenthető.

A folyamatos szekvenciális áteresztőképesség a lemeztányérok adatsűrűségétől és fordulatszámától függ. A lemezekre koncentrikus körökben felírt adatsávok kerülete függ a kör sugarától, így az egyes adatsávokon tárolható adat mennyisége is változó. Mivel a lemez fordulatszáma állandó, ezért az áteresztőképesség függ attól, hogy a fej éppen melyik adatsávot olvassa végig (kis kerületű adatsávban egy fordulat alatt kevesebb adat halad el a fej alatt). Ennek a következménye, hogy a lineáris áteresztőképesség egy jellegzetes csökkenő görbét mutat, amit figyelembe kell venni teljesítménytervezéskor.

Optikai lemezek Számos szabványos és nem szabványos optikai adattároló formátum létezik, mára már szinte teljesen egyeduralkodóvá váltak a szabványos, 12 illetve 8 cm-es lemezeket használó megoldások. Kapacitás, illetve az alkalmazott lézer hullámhossza szerint három generációba sorolhatóak: CD (700 MB), DVD (4.5 GB), HD-DVD (15 GB) ill. Blu-Ray (25 GB). Minden generációnak van előregyártott (-ROM), egyszer írható (-R), illetve újraírható (-RW) alváltozata. A merevlemezeknél elmondottakhoz hasonló mechanikai elven működnek, a teljesítményük is hasonlóan írható le, ám a tipikus számértékek lényegesen rosszabbak. A véletlen elérési idő 100ms nagyságrendű, a szekvenciális áteresztőképesség itt is pozíciófüggő, CD-k esetén legfeljebb 1-3 MB/s, DVD-k esetén 2-8 MB/s érhető el.

Kis fajlagos költségéből, de kedvezőtlen teljesítményparamétereiből adódóan optikai adattárolást jellemzően adatok tömeges terjesztésére, illetve archiválásra használnak.

Szalagos adattárolás A merevlemezekhez hasonlóan a szalagos adattárolás is mágneses elven történik. A szalag természetéből adódóan a hozzáférés szekvenciális, véletlen hozzáférés csak a szalag hosszadalmas tekercselésével oldható meg. A kazettánkénti kapacitás (800GB LTO-4) és a szekvenciális áteresztőképesség (120MB/s LTO-4) nagyjából hasonló a merevlemezekéhez, a kazetták fajlagos költsége azonban alacsonyabb, az optikai adattárolókéhoz hasonló. Tulajdonságai miatt szalagos adattárolást manapság kizárólag nagyméretű backup és archiválási feladatokra alkalmaznak. Elérhetőek robotkaros szalagtárak (Autoloader, Tape Library), melyek nagy mennyiségű kazetta tárolását és a meghajtókba automatikus betöltését teszik lehetővé. A legnagyobb tárak összkapacitása elérheti az 5 PetaByte-ot (IBM TS3500T).

Szilárdtest adattárolók (*Solid State Drive*, SSD) Az eddig ismertetett adattároló eszközök mind mozgó alkatrészeket tartalmaznak, ami egyrészt sérülékennyé teszi, másrészt korlátozza az elérhető sebességet, különösen a véletlen elérést. Ahol a környezeti hatások miatt különleges tűrőképességű tárolóeszközök kellenek (pl. ipari beágyazott rendszerek, hordozható eszközök), vagy a merevlemezek véletlen elérési idejénél lényegesen gyorsabb hozzáférésre van szükség, ott mozgóalkatrész nélküli, szilárdtest adattárolókat alkalmaznak. Ennek ma két elterjedt fajtája van: a **Flash** memória és az **akkumulátorral táplált RAM** (Battery-Backed RAM). A flash

természeténél fogva nem felejtő, elviekben az adatot éveken át is képes megőrizni és a készenléti állapota sem igényel energiát. A flash alapú SSD-k hozzáférési ideje nagyjából 0.1 ms, a szekvenciális áteresztőképesség 50-200 MB/s, bár párhuzamosítással ennél lényegesen nagyobb sebességek (1500 MB/s Fusion-io ioDrive Duo 640) is elérhetők. Jellemző, hogy az írás lassabb az olvasásnál. További fontos jellemző a flash memóriacellák korlátozott számú újraírási lehetősége (10^4-10^6), a meghibásodásokat többnyire a cellák elhasználódása okozza. A flash SSD-k általában tartalmaznak terheléskiegyenlítő (wear-levelling) allokációs algoritmust, ami a gyakran felülírt blokkokat rendszeresen áthelyezi, hogy megakadályozza a korai meghibásodást. A flash fajlagos költsége nagyjából tízszerese a merevlemezekének, ám ez az arány folyamatosan csökken, így várható, hogy a közeljövőben a merevlemezeket nagy tömegben fogja kiváltani, különösen a notebookokban, ahol fontos a kis fogyasztás. Nagyvállalati IT rendszerekben olyan helyeken alkalmazzák, ahol nincs szükség nagy helyi adattárolásra (csak OS betöltéshez kell) vagy viszonylag kis adathalmazon merevlemezeknél gyorsabb véletlen elérés kell, ahol főleg az olvasásműveletek a dominánsak.

A RAM-alapú SSD-k folyamatos frissítést igényelnek, ezért van szükség beépített szünetmentes tápellátásra. Általában egybe vannak építve merevlemezzel is, hogy áramkimaradás esetén az akkumulátor lemerülése előtt még stabil tárba tudja a memória tartalmát menteni. A készenléti fogyasztásuk és fajlagos költségük is rendkívül magas, összkapacitásuk viszonylag kicsi (1TB 4 egység magas rack méretben $375000 áron!). Csak olyan esetekben alkalmazzák őket, ahol kis adathalmazon rendkívül gyors (< 5 μs) hozzáférési időkre van szükség, például sok tranzakciót kiszolgáló adatbázisok esetében.

Általában elmondható, hogy a korszerű tárolóeszközök rendelkeznek belső hibadetektálással, bizonyos esetekben korlátozott mértékű hibajavítási lehetőséggel is. Meghibásodás esetén általában képesek jelezni a operációs rendszer felé, hogy a kért műveletet nem tudták végrehajtani. Ritka az olyan meghibásodás, amikor egy eszköz hibajelzés nélkül hibás adatot ad vissza.

2. Hibatűrő megoldások, RAID

A RAID (Redundant Array of Independent Disks) olyan megoldások összefoglaló neve, melyek több különálló eszközt kapcsolnak össze a tárkapacitás, teljesítmény illetve hibatűrési tulajdonságok javítása érdekében.

Manapság a következő RAID szinteket használják:

JBOD („just a bunch of disks”), egymás után összefűzi az egyes eszközöket. Hibatűrést nem biztosít, bármely eszköz kiesése hibát okoz.

RAID-0 (striping), az egyes eszközöket blokkokra bontja, a blokkokat sorban szétosztja az egyes eszközök között, így alakítva ki sávokat (stripe). Teljesítmény- és kapacitásnövelő célt szolgál, hibatűrést ez sem biztosít. A JBOD-hoz hasonlóan a RAID-0 is növeli a meghibásodás valószínűségét, mert a tömb bármely tagjának hibája potenciálisan az összes adatot használhatatlanná teheti.[1] Szekvenciális olvasás és írás áteresztőképessége az eszközök számával szorozódik, ha nincs egyéb szűk keresztmetszet a rendszerben. Véletlen elérések akkor párhuzamosíthatóak, ha az egyes műveletek által érintett adat nem nagyobb a blokkméretnél.

RAID-1 (mirroring), tükrözi az adatokat, azaz minden adatot minden eszközre kiír. Elsősorban hibatűrést biztosít. A mai implementációk csak az írásműveleteket többszörözik, olvasást normális esetben csak egy eszköz végzi el, feltételezve, hogy ha az eszköz meghibásodik, akkor a hibát jelzi a tömb vezérlője felé, így nem kell komparálni. Ennek következménye, hogy random olvasás műveleteket vagy több egyidejű szekvenciális olvasást lehet párhuzamosan is végezni. A teljesítmény blokkmérettől független. Leggyakrabban 2 diszkből építenek RAID-1 tömböt.

[1] Gondoljuk arra, hogy ha egy fájlrendszer több eszközre van szétosztva, akkor bármelyik lemez kiesése esetén a fájlrendszernek fontos metaadatai válhatnak elérhetetlenné, amik szükségesek lehetnek a megmaradó eszközön tárolt fájlokhoz való hozzáféréshez is. Legrosszabb esetben a fájlrendszert fel sem lehet csatolni, ha az alatta lévő blokkos eszköz címtartományának jelentős része hiányzik. A gyakorlatban ilyen sérülések után a fájlrendszerekről néha még helyreállítható az adatok egy része, ám ez hosszadalmas, jelentős emberi közreműködést igénylő, bizonytalan kimenetelű procedúra, ezért tervezésnél nem építhetünk rá.

RAID-5 (striping with parity), blokkokra bontás és hibajavítás paritással. Hasonló a RAID-0 esetén ismertetett sávkialakításhoz. Minden sávban az egyik blokk paritást tartalmaz, ami a sáv többi blokkja alapján képződik. Egy eszköz kiesése esetén az adatok helyreállíthatóak a megmaradt adatblokkok és a paritás segítségével. Egynél több eszköz kiesése már adatvesztést okoz. Szekvenciális olvasás és írás áteresztőképessége az eszközszám-1 szeresére skálázódik. A random olvasásműveletek a RAID-0-hoz hasonlóan alakulnak, blokkmérettől erősen függenek. A random írásműveleteknél fontos, hogy minden blokknyi írás összesen 2 blokk, egy adat és a paritás tartalmának frissítését igényli. Mivel a paritást minden írásnál frissíteni kell, ezért fontos, hogy a paritásblokk sávonként más és más eszközre kerül. Leggyakrabban a RAID-5 tömbök 3 vagy 4 diszkből épülnek fel, ennél többet ritkán raknak egy tömbbe.

RAID-6 (striping with dual parity), blokkokra bontás és kétszeres paritás. A RAID-5-től annyiban tér el, hogy kétféle paritás van sávonként, ezáltal a tömb bármely két eszköz kiesését túléli adatvesztés nélkül. Ez viszonylag új RAID szint, csak a legújabb eszközök támogatják. A használata elterjedőben van, aminek a fő oka, hogy a mai kapacitások mellett rendkívül hosszú ideig tart egy RAID-5 tömbben meghibásodás után helyreállítani az adatokat, ez idő alatt a tömbnek még nincs redundanciája.

Kombinált RAID szintek legjellemzőbb a RAID-10 (RAID0+1 vagy RAID1+0), RAID-50 (RAID5+0) RAID-51[2] (RAID5+1), RAID-60 (RAID6+0). Ezek tömbökből képzett tömbök, amiket különösen nagyszámú (4-nél több, jellemzően 12) diszk esetén alkalmaznak. Az első szám az elsődleges, fizikai eszközökből épülő tömbök jellegét adja meg, a második szám a tömbökből képzett másodlagos tömb típusát. Hálózati tárolóeszközöknél (SAN, illetve NAS hardverek, lásd később) gyakori a 2 egység magas rack házban 12 diszk ami RAID-10, RAID-50 vagy RAID-60 tömbbe konfigurálható.

RAID Szint		**lemezek száma**	**hibatűrés**	**kapacitás**	**szekv. áteresztés (legjobb esetben)**	**véletlen hozzáférés párhuzamosítás (legjobb esetben)**	
						írás	**olvasás**
JBOD	eszközök egymás után kapcsolása	2..N	nincs	N szeres	1 szeres	1 szeres	1 szeres
RAID-0	blokkokra bontás, eszközök között szétosztás	2..N	nincs	N szeres	N szeres	N szeres	N szeres
RAID-1	tükrözés	2..N	$N-1$ meghibásodásig	1 szeres	1 szeres	1 szeres	N szeres
RAID-5	blokkokra bontás, szétosztás, egyszeres paritás	3..N	1 meghibásodásig	$N-1$ szeres	$N-1$ szeres	$N/2$ szeres	$N-1$ szeres
RAID-6	blokkokra bontás, szétosztás, kétszeres paritás	4..N	2 meghibásodásig	$N-2$ szeres	$N-2$ szeres	$N/3$ szoros	$N-1$ szeres

Redundanciával rendelkező RAID tömbben meghibásodás esetén a következő eseménysor zajlik le:

1. Az egyik eszköz hibát jelez, vagy nem válaszol kérésre egy meghatározott ideig (timeout)
2. A RAID vezérlő a kérdéses eszközt hibásnak jelöli és leválasztja a tömbről
3. A tömb így *degradálódott* állapotba kerül. (Típusától és az eszközök számától függően lehetséges, hogy teljes redundanciavesztés lép fel, vagyis a tömb már nem képes további meghibásodást elviselni.) Degradálódott állapotban is képes kiszolgálni írás és olvasás kéréseket, ám a teljesítmény csökkenhet.

[2]Néhány gyártó ezt RAID-53-nak nevezi

4. Egy karbantartó a tömbhöz új diszket rendel hozzá, vagy fizikailag cseréli a hibásat.

5. Megkezdődik a kiesett diszk tartalmának rekonstruálása az új eszközre. Ez egy hosszú folyamat, a teljes diszket tele kell írni, miközben a tömbhöz érkezett kéréseket is ki kell szolgálni.

6. A rekonstrukció végeztével a tömb újra hibátlan állapotba kerül.

A redundanciavesztett állapotban a tömb sérülékeny, ezért minimalizálni kell a redundancia nélküli üzemidőt. Ezt az időt nemcsak a rekonstrukció időtartama határozza meg, hanem az az idő is, ami a meghibásodástól az új eszköz hozzáadásáig eltelik. Mivel a RAID feladata, hogy az operációs rendszer számára elfedje a meghibásodást, ezért a RAID rendszer aktív felügyelete nélkül nem lehet észrevenni a degradálódott állapotot, ami hamis biztonságérzetet okozhat. További nehézség lehet a géphez fizikai hozzáférés biztosítása, esetleg hosszú időbe kerülhet a géphez eljutni. Lehetőség van a gépbe *hideg-* vagy *melegtartalék* eszközt beépíteni, ami vagy kikapcsolt vagy bekapcsolt készenléti állapotban várakozik, hogy átvegye a helyét egy sérült eszköznek a tömbben. Így a kijavítás felügyelet nélkül is automatikusan megkezdődik, a hibás eszköz cseréje később is elvégezhető.

3. Dinamikus tárkiosztás

3.1. Az alapprobléma

A háttértárak lehetséges kapacitását az alkalmazott eszközök határozzák meg. Az alkalmazások tárigénye ettől eltérő, ami nagyon rossz kihasználtságot eredményez. Particionálással részekre lehet bontani az eszközöket az alkalmazások igényeinek megfelelően. A hagyományos, PC-ken alkalmazott particionálási séma rendkívül régi, a mai követelményeknek gyakran nem felel meg. A fő probléma a partíciók statikussága, azaz nem lehet átméretezni őket, hogy kövesse az alkalmazások változó igényeit. [3]

3.2. Logikai kötetkezelés

Korszerűbb, dinamikus particionálást biztosítanak a **logikai kötetkezelő** (LVM, *Logical Volume Manager*) rendszerek. Működési elve hasonló az operációs rendszereknél megismert virtuális memóriakezeléshez.

Az LVM alapfogalmai:

- **Fizikai kötet** (PV, *physical volume*)
- **Kötetcsoport** (VG, *volume group*)
- **Logikai kötet** (LV, *logical volume*)

A fizikai köteteket allokációs egységekre (*allocation unit*, *extent*, ritkán: *partition*) bontja. Az allokációs egységek tetszőleges kettő hatvány méretűek lehetnek, leggyakrabban 1 és 64 MB közötti mérettartományban. Minden fizikai kötet része egy kötetcsoportnak, ami egy nagy közös erőforrás gyűjtőnek (resource pool) tekinthető. A kötetcsoportból lehet allokálni logikai köteteket, ami egész számú allokációs egységből állhat. Fontos megjegyezni, hogy a logikai kötet a kötetcsoportban található bármely fizikai kötet bármely allokációs egységét bármilyen sorrendben megkaphatja, nem feltétlen összefüggő sorozatot. A hozzárendelést a logikai kötetkezelő metaadatai tárolják, melyek a fizikai kötetek fejlécében kapnak helyet. A logikai kötetek tehát egy virtuális lineáris (0-tól a kötet kapacitásáig terjedő, megszakítás nélküli) címtartományból állnak, ami tetszőlegesen lehet leképezve a tényleges fizikai eszközök címeire, akár egynél több különböző fizikai eszközre is szétszórva. A kötetcsoportok bármikor bővíthetőek új fizikai eszköz hozzárendelésével, a logikai kötetek pedig bővíthetőek új allokációs egységek hozzárendelésével, akár működés közben is.

[3] Ugyan léteznek eszközök átméretezésre, ezek azonban mindig teljes partíciók adatmozgatásával járnak, ami hosszú folyamat, és rendszer normális működése közben nem végezhető el.

3.3. A logikai kötetkezelés teljesítményre gyakorolt hatása

Az allokációs egységek nem összefüggő hozzárendelése esetén az egységek határán átnyúló szekvenciális hozzáféréseknél fejpozicionálást kell végezni. Ezért a kötetkezelő törekszik arra, hogy a logikai köteteket összefüggően ossza ki. Ez nyilván akkor lehetetlen, ha meglévő logikai kötetet próbálunk bővíteni és a vége után következő egységet már más logikai kötet elfoglalta. A fejpozicionálások hatása csökkenthető alkalmasan nagy allokációs egység méret megválasztásával. A mai merevlemezek 100MB/s szekvenciális áteresztőképessége mellett egy 64MB-os egység 640 ms alatt végigolvasható. Legrosszabb esetben 640 ms-onként kell egy, átlagosan 20 ms-os fejpozicionálás, ami összesen kb. 6% teljesítményvesztést jelent, ami alig kimutatható. Véletlen hozzáféréseknél a hatás jelentősebb lehet, ha egy kis ugrás a virtuális címtartományban valójában nagy ugrás a fizikai eszközön. Különösen az okozhat problémát, hogy egyes fájlrendszerek optimalizálnak a véletlen hozzáférések ugráshosszának rövidítésére, feltételezve, hogy a címtartományban kis eltérés esetén gyorsabb a pozicionálás, mint nagy eltérésnél. Mindezek ellenére a gyakorlatban az LVM nem okoz észrevehető lassulást, többnyire mert nagy a blokkméret és ritkán kell bővíteni partíciót, így kicsi a fragmentálódás is.

3.4. A logikai kötetkezelés egyéb szolgáltatásai

A dinamikus allokáción túl a legtöbb LVM megvalósítás egyéb szolgáltatásokkal is kiaknázza az indirekt allokáció adta lehetőségeket:

- Atomi pillanatkép készítés a teljes logikai kötetről (snapshot), majd a változások transzparens követése. Ennek egy alkalmazása lehet például teljes rendszer gyors visszaállítása egy elmentett állapotra, illetve működés közben konzisztens mentés (backup) készítés a teljes kötetről.
- Logikai kötetek működés közbeni mozgatása eszközök között.
- Redundancia az allokációs egységek tükrözésével. Ez a RAID-1-hez hasonlóan több példányban, különböző fizikai kötetekre írja ki a logikai kötet összes allokációs egységét.
- Teljesítménynövelés allokációs egységek sávos szétosztásával. A RAID-0-nál megismert elvet követi.

A tükrözés és sávos szétosztás olyan esetekben bír nagy jelentősséggel, ha egyébként nincs RAID lehetőség a rendszerben.

A pillanatkép készítése és fenntartása a copy-on-write működési elvet követi. A pillanatkép számára egy külön logikai kötetet kell létrehozni, tetszőleges mérettel (legfeljebb az eredeti kötet mérete állítható be). Az új logikai kötet eleinte üres, ám minden[4] az eredeti köteten végzett írás művelet előtt a módosítani kívánt allokációs egységről készül egy másolat a pillanatkép kötetre. Az eredeti kötet tartalmának módosulásával tehát egyre több adat kerül a pillanatképre. Amennyiben a pillanatkép számára kijelölt méret kisebb az eredeti köteténél, a pillanatkép kötet betelhet a módosítások hatására. Emiatt a pillanatkép méretét az alapján kell megválasztani, hogy várhatóan mennyi ideig lesz szükség rá, illetve, hogy ez idő alatt mekkora módosítások várhatóak. Egy backup elkészítésének viszonylag rövid idejére az eredeti kötet méretének akár 5-10%-a is elegendő lehet.

Fontos kiemelni az allokációs egység méretének hatását. Mind a sávos szétosztás, mind pedig a pillanatkép készítés esetén az allokációs egység az alapvető blokkméret, amivel a kötetkezelő az említett műveleteket elvégzi. Pillanatképnél minden módosítás - érintsen az bármilyen kis adategységet - egy teljes allokációs egység másolását fogja kiváltani. Ha nagy allokációs egységek használatakor sok elszórt kis módosítás történik, akkor lényegesen több adatról fog másolat készülni, mint amennyi változott, így hamar elfogyhat a pillanatkép kötet helye, továbbá a másolások sok ideig tartanak, ezzel számottevően csökkentve a teljesítményt. Ez a legfontosabb oka annak, hogy az LVM megvalósítások nemcsak nagy (akár 1GB-os), hanem egészen kicsi (akár 4 kB-os) allokációs egységek használatát is lehetővé teszik.

[4] Pontosabban csak az olyan allokációs egységekről, amelyekről korábban még nem készült másolat a pillanatkép kötetre.

4. Központosított tárrendszerek

4.1. Problémák az elosztott lokális tárolással

Sok szervert tartalmazó nagyvállalati rendszerekben, jelentős adminisztratív és karbantartási terhet jelent, ha minden gépben lokális tárhely van.

Költség szempontból sem optimális, mert igen gyakori, hogy egy szervernek csak kis tárhelyre van szüksége, ám fontos a hibatűrés vagy a nagy teljesítmény. Az eszközök ára nem áll egyenes arányban a kapacitással, ezért sok kis tárhely lényegesen drágább egy nagy tárhelynél. Bővítési szempontból is nehézségek merülnek fel, mert a gépekbe szerelt kis lokális táreszközök gyakran csak cserével bővíthetők, ami leállással és hosszadalmas adatmozgatással járhat.

Hibatűrő vagy terheléselosztó fürtök (cluster) esetén szükség van közösen használt tárhelyre, ami szintén nem oldható meg csak lokális tárakkal.

4.2. Tárhálózatok

A fenti problémákra kínálnak megoldást a központosított, hálózati tármegoldások. Két fajtáját különböztetjük meg: a hálózati fájlrendszereket és a hálózati blokkos eszközöket. Ezek megoldhatóak szoftveresen, de kaphatóak dedikált hardverek is, melyek hálózati tárhelyet biztosítanak. A fájlrendszer megosztást kínáló eszközök neve *Network Attached Storage* (NAS), a blokkos eszközöket nyújtó megoldások neve *Storage Area Network* (SAN). Központi hálózati tárakkal lehetőség nyílik egy nagy közös pool-ból dinamikusan, az aktuális igényeknek megfelelően hozzárendelni tárhelyet az egyes szerverekhez. A nagy központi tár hibatűrését könnyű megoldani, továbbá optimális fajlagos költségű eszközökből állhat. A központ tár karbantartása lényegesen kevesebb emberi erőforrást igényel a sok elosztott tár felügyeletéhez képest.

Most elsősorban a SAN megoldásokkal foglalkozunk. Számos különböző SAN protokoll van, melyeket eltérő helyen alkalmaznak:

- **FibreChannel** Optikai összeköttetés, típustól függően 1-20 GBit/s sebességgel. Legegyszerűbb esetben pont-pont összeköttetés, bonyolultabb topológiák esetén speciális kapcsolóelemeket (switch) igényel. SCSI utasításkészletet használ, bár lehetséges ATM vagy IP hálózatot is kiépíteni vele. Rendkívül drága.

- **iSCSI** TCP kapcsolatot épít ki a kliens (*kezdeményező, initiator*) és a szerver (*célpont, target*) között, és efelett a SCSI protokoll parancsait és üzenetformátumát használja. Viszonylag olcsó (létezik teljesen szoftveres megvalósítás is), szabványos és széles körben támogatott. Általában dedikált gigabit ethernet hálózatot építenek ki iSCSI számára.

- **HyperSCSI** Közvetlenül ethernet keretekbe ágyazza be a SCSI parancsokat és üzeneteket. Teljesítmény szempontból ez jobb lehet az iSCSI-nál, viszont nehézkes konfigurálni. Kevéssé támogatott protokoll.

- **ATA over Ethernet** a HyperSCSI-hoz hasonló, de ez ATA parancskészletet használ a SCSI helyett. Nagyvállalati környezetben nem használják, kis olcsó asztali SAN dobozok alkalmazzák ezt a protokollt.

Hálózati fájlrendszerek általános képessége, hogy többen csatlakozhatnak hozzá egy időben és meg tudja oldani az elosztott hozzáférések között a megfelelő kölcsönös kizárásokat. Hasonló megoldás SAN-ok esetében is lehetséges, azonban olyan fájlrendszer kell, amely az alacsony szintű struktúráin képes lekezelni, hogy egyszerre többen módosítják a tartalmát. Ilyen fájlrendszerek pl a VMWare Virtual Machine File System v3 (VMFS3), Oracle Cluster File System (OCFS) vagy a RedHat Global File System (GFS).

5. Blokkos eszközök kezelése Linux alatt

A UNIX-alapú rendszerek hagyományos filozófiáját követve Linux alatt is a fájlok formájában férhetünk hozzá a hardver eszközökhöz. Itt megjegyzendő, hogy a fájlokra úgy kell tekinteni, mint hierarchikus névtérbe szervezett egyedi azonosító névvel ellátott objektumokra, amik egy jól meghatározott interfészt implementálnak. A fájl API legfontosabb metódusai a `read()` és a `write()`. Egyszerű, ún. reguláris fájlok egy változó hosszúságú byte tömböt implementálnak, amiben pozicionálni is lehet a `seek()` metódussal. Speciális eszköz fájl esetén azonban a `read()` és `write()` metódusokat egy-egy kernel meghajtó (*driver*) implementálja teljesen egyedi módon, ez azt jelenti, hogy a hagyományos jelentésétől akár teljesen eltérő dolgot is csinálhat.

Néhány speciális eszköz:

- `/dev/null` - üres write művelet, a read mindig fájl vége jelzést (*EOF*) ad vissza, ez gyakorlatilag egy mindent eldobó nyelő
- `/dev/zero` - üres write, a read minden hívása 0-értékű byte-okat ad vissza, gyakorlatilag egy végtelen 0 forrás.
- `/dev/full` - a write művelet mindig tele jelzést ad, a read a `/dev/zero`-hoz hasonló
- `/dev/random` és `/dev/urandom` - üres write, a read véletlen adatfolyamot állít elő.

A tömegtároló eszközök is ilyen fájlokon keresztül érhetőek el, bár ezek viselkedése hasonlít a reguláris fájlokéhoz annyiban, hogy ezek is egy perzisztens adattároló tömbként jelenítik meg az eszközt, de a méret ez esetben fix.

- `/dev/sda`, `sdb`, `sdc`, stb. - SCSI API-val elérhető merevlemezek. A legutóbbi kernelkiadásokban már nem csak a valódi SCSI merevlemezek, hanem a SATA, PATA merevlemezek, illetve USB-storage (külső merevlemez, pendrive) eszközök meghajtóprogramjai is ezek alatt a nevek alatt jelennek meg. A neveket inicializálási sorrendben kapják, tehát változhat, hogy melyik eszköz melyik név alatt jelenik meg.
- `/dev/sr0`, `sr1` stb. - SCSI, újabban SATA és PATA CD/DVD írók. Inicializálási sorrendben számozódnak.
- `/dev/hda`, `hdb`, `hdc`, stb. - Régebbi kerneleknél az IDE (főleg PATA) merevlemez és CD/DVD meghajtók eszköznevei. Busz pozíció szerint kötött nevük van: a primary master a `hda`, primary slave a `hdb`, secondary master a `hdc`, stb.
- `/dev/loop0`, `loop1`, stb. - különleges eszközök, nincs valódi hardver alattuk, reguláris fájlokat lehet hardver eszközként kezelni velük. Ezzel lehet például .iso CD image fájlokat felcsatolni anélkül, hogy CD lemezre kéne kiírni.

A merevlemezek particionálhatóak és általában particionáltak is, ezeket a kernel az eszköz inicializálásakor deríti fel az ún. superblock (az eszköz első blokkja, ami gyakran metainformációkat tartalmaz) beolvasásával. A partíciók számára külön számozott eszközök jönnek létre, pl. `/dev/sda1`, `sda2`, stb. A partíciók számozása követi a partíciós tábla kiosztását, tehát előfordulhat, hogy nem folytonos a számozás. A partíció fájl az eredeti eszközfájlon belül egy korlátozott címtartományt fed le. A partíció tartalma a partíció fájlban és a teljes eszközt reprezentáló fájlban egyidejűleg látható.

A tömegtároló eszközök szinte mindig blokkos szervezésűek, ami azt jelenti, hogy valójában nem byte-ok, hanem blokkok tömbjeként épülnek fel. A blokk a legkisebb elemi írás vagy olvasás művelet adatmérete. A pozíciók címzése is blokk egységekben történik. A kernel átszámolja a címzést és puffereli a fájlokon végzett műveleteket, ezért azoknak nem kell feltétlenül a blokkméretet követni, byte pontosságú címzést használhatnak. Fontos azonban megjegyezni, hogy az eszköz blokkméretének egész számú többszörösével végzett adatműveletek lényegesen

gyorsabbak és kevesebb processzorterheléssel járnak. A merevlemezek blokkmérete szinte mindig 512 byte [5], ez a CD-k és DVD-k esetén 2048 byte. A flash eszközök különlegesek, mert kétféle blokkméretet is használnak, olvasás és üres területre írás egy kicsi, általában 4096 byte-os blokkmérettel (ún. lap, page) történik, ám a már megírt területek felülírás előtti törlése lényegesen nagyobb egységekben, nem ritkán 128 kB-os vagy 256 kB-os blokkokban történik. Emiatt a beépített blokk transzlációs algoritmus nélküli flash eszközök (Linux alatt *Memory Technology Devices, MTD*) kezelése külön speciális interfészt igényel. A transzlációs algoritmus feladata többek között a szemétszedés (garbage collection), vagyis a használt, érvénytelennek jelölt adatok által foglalt hely visszanyerése. Ennek működése gyakran a törölni kívánt blokkban található még érvényes adatok új helyre mozgatásával jár, ezek a járulékos írásműveletek az SSD eszközöknél számottevő teljesítménycsökkenést okoznak. Ez sajnos egy szükségszerű következménye a kétféle blokkméret használatának.

Nagyméretű blokkos eszközökön adatmanipulációt a `dd` programmal végezhetünk el. Ez lényegében egy egyszerű másolóprogram, aminek megadhatjuk, hogy milyen blokkméretet használjon, illetve a bemeneten mettől meddig olvasson, a kimenetre pedig milyen pozíciótól kezdődően írjon.
Paraméterezése:

Paraméterezése: `dd if=`*bemenő fájl* `of=`*kimenő fájl* `bs=`*blokkméret* `count=`*másolandó blokkok száma* `skip=`*bemenet kezdőpozíciója* `seek=`*kimenet kezdőpozíciója*

Minden paraméter opcionális, a bemenet alapértelmezetten a standard input, a kimenet a standard output, a blokkméret 1, a bemeneti illetve kimeneti pozicionálás egyaránt 0. A felsoroltakon kívül természetesen van még számos, kevésbé gyakran használt paramétere.

Például 1 MB-os üres fájl létrehozása 1 kB-os blokkok használatával:

```
dd if=/dev/zero of=./zerofile bs=1024 count=1024
```

6. Fájlrendszerek létrehozása, átméretezése

Ahhoz, hogy a blokkos eszközökön vagy partíciókon reguláris fájlokat tudjunk tárolni szükség van valamilyen fájlrendszerre. A fájlrendszereket nemcsak partícióra, hanem particionálatlan teljes eszközökre is lehet rakni, bár általában a szokásos megoldás egy partíció létrehozása még akkor is, ha a teljes eszközt használni akarjuk.

A fájlrendszer is rendelkezik egy mérettel, ami alapértelmezésben a partíció teljes mérete, de lehet kisebb is annál. Ez akkor fordul elő, ha a partíciót átméretezzük. Méret csökkentésnél először a fájlrendszert kell zsugorítani, majd utána lehet a tároló eszköz méretét hozzáigazítani, növelésnél előbb az eszközt kell megnövelni és utána lehet a fájlrendszert hozzánöveszteni, hogy kitöltse a rendelkezésre álló helyet.

Linux alatt számos fájlrendszerfajtát lehet használni, a jelen példában a legelterjedtebben használt *ext3fs* fájlrendszer kerül ismertetésre.

6.1. Ext3 létrehozása

A fájlrendszer létrehozására (közkedvelt nevén *formattálás*) az `mkfs.ext3` segédprogrammal történik, alapértelmezett esetben:

`mkfs.ext3` *fájlnév*

A fájl lehet eszköz, de akár egy előre lefoglalt reguláris fájl is, amin belül kialakításra kerülnek a fájlrendszer alacsony szintű adatstruktúrái.

6.2. Felcsatolás

A fájlrendszer felcsatolható a `mount` paranccsal. Fájlrendszert alapértelmezetten csak a *root* felhasználó csatolhat fel, egyszerű felhasználó csak abban az esetben, ha azt a `/etc/fstab` konfigurációs fájlban engedélyeztük.

[5] Egyes SCSI merevlemezek esetén ez átkonfigurálható az 512 byte 2 hatvány szorosára

`mount` [`-o` [`loop`],[`ro`]] *fájlnév csatolási pont*

A csatolási pont egy könyvtárnév a már felcsatolt könyvtárhierarchiában. Ez a könyvtár lesz a most felcsatolt fájlrendszer gyökéreleme. Gyakori opció a `loop`, amit akkor kell használni, ha reguláris fájl tartalmát akarjuk felcsatolni. A mount parancs ilyenkor automatikusan használatba veszi a következő szabad `/dev/loop` eszközt. Másik gyakori opció a `ro`, ami azt jelöli, hogy csak olvasható üzemmódban csatolja fel.

Egy már felcsatolt fájlrendszer lecsatolható a következő paranccsal:

`umount` *eszköznév vagy csatolási pont*

Ha valamilyen folyamat éppen nyitott fájlt tart az adott eszközön (pl. a shellben éppen a felcsatolt eszközön lévő könyvtárban vagyunk), akkor hibaüzenetet kapunk, a lecsatolás nem fog sikerülni. Különösen figyeljünk arra, hogy felcsatoláskor ne fedjük el a `/dev` illetve a `/proc` könyvtárakat, mert ezzel megakadályozzuk, hogy a fel- és lecsatolás folyamatához szükséges speciális fájlokhoz hozzáférjen a rendszer, így a későbbiekben már nem tudjuk lecsatolni őket. Ilyenkor többnyire csak újraindítással lehet helyreállítani a rendszer normális működését.

6.3. Átméretezés

Az ext3 fájlrendszer lehetővé teszi, hogy felcsatolt állapotban is megnövelhessük a méretét (*online grow*), azonban méret csökkentést csak lecsatolt állapotban lehet végezni. Az átméretezés a következőképpen zajlik:

`resize2fs` *eszköz új méret*

A méret megadható fájlrendszer blokkokban (létrehozáskor paraméterrel megadható), 512 byteos (`s` végződés) kB egységekben (`K` végződés), MB egységekben (`M` végződés).

Minden fájlrendszerhez saját segédprogramkészlet tartozik és eltérő átméretezési képességekkel bírnak.

7. Szoftveres RAID Linux alatt, md

Általában RAID tömböket csak azonos méretű és típusú merevlemezekből építenek. A Linux szoftver RAID megvalósítása, az *md* (*multiple devices*) nem köt ki semmit, létrehozhatóak teljesen vegyes tömbök is, nemcsak merevlemezekből, hanem tetszőleges blokkos eszközből. Teljesítmény szempontból azonban előnyösebb törekedni arra, hogy azonos eszközöket használjunk.

7.1. Particionálási megfontolások

A felépített és aktív RAID tömbök a `/dev/md0` `/dev/md1`, stb fájlok alatt érhetőek el, amik szintén létrehozási sorrendben számozódnak.

A régebbi kernel verziók nem támogattak partíciókezelést md tömbökön belül, ezért a gyakorlat az volt, hogy nem a teljes merevlemez eszközökből építettek raid tömböt, hanem egyformán particionálták azokat és a megfelelő partíció csoportokból építettek fel raid-ezett partíciókat.

Az újabb verziókban lehetőség van particionált raid tömbök kezelésére is, ezek jellemzően `/dev/md0p0`, `md0p1` stb. elnevezési sémát követő eszközök alatt érhetőek el.

Ennek ellenére a korábbi gyakorlat megmaradt, ugyanis lehetővé teszi, hogy a merevlemez különböző partícióit különböző RAID szintű tömbökbe szervezzük. Ez különösen a boot partíciók esetén fontos, mert a bootloaderek nem tudnak raid tömböket kezelni, így a boot partíciókat csak RAID-1 tömbbe szabad szervezni, aminek minden tagja a tömbtől függetlenül, külön-külön is működőképes.

Egy másik gyakran alkalmazott megoldás a raid tömbök particionálására a későbbiekben részletes ismertetésre kerülő logikai kötetkezelés, ami a hagyományos particionálási sémát egy lényegesen dinamikusabb megoldással váltja fel.

7.2. Műveletek RAID tömbökkel

A raid tömböket kezelő segédprogram az `mdadm`. Tömb létrehozás általános szintaktikája:

`mdadm --create` *eszköz* `--level=`*raid szint száma* `--raid-devices=`*tagok száma eszközök felsorolása space-el elválasztva*

példa:

`mdadm --create /dev/md0 --level=raid1 --raid-devices=2 /dev/sdb /dev/sdc`

A létrehozás folyamata megfigyelhető a következőképpen:

`cat /proc/mdstat`

A létrehozott tömbök tagjainak metaadatait megnézhetjük így:

`mdadm --examine` *tömb valamely tagja*

A teljes létrehozott tömb adatait pedig így jeleníthetjük meg:

`mdadm --query` *tömb*

`mdadm --detail` *tömb*

Tömb leállítása (az md eszköz megszűnik):

`mdadm --stop` *tömb*

Már létrehozott tömb elindítása:

`mdadm --assemble` *tömb*

Az összes bekonfigurált tömb elindítása:

`mdadm --assemble --scan`

Tartalék (hot spare) eszköz hozzáadása:

`mdadm --add` *tömb eszköz*

Eszköz hibásnak jelölése (soft hibainjektálás[6]):

`mdadm --fail` *tömb eszköz*

Hibás vagy tartalék eszköz kivétele a tömbből:

`mdadm --remove` *tömb eszköz*

Tömb teljes leállítása:

`mdadm --stop` *tömb*

Ahhoz, hogy az egyszer létrehozott tömbök leállítás után újraindíthatóak legyenek, illetve az operációs rendszer újraindítása után automatikusan felépüljenek egy konfigurációs fájlba (`/etc/mdadm.conf`) kell beleírni a tömb adatait, mindenekelőtt az egyedi azonosítóját (UUID). Egy bejegyzés így néz ki:

`ARRAY /dev/md0 level=raid1 num-devices=2 UUID=5e7e8a70:503417e0:25c6f0a3:1617215c`

A következő utasítás automatikusan előállítja a bejegyzéseket az éppen aktív raid tömbökről, ezeket csak hozzá kell fűzni a `/etc/mdadm.conf` végéhez:

`mdadm --detail --scan`

A már nem létező tömbökhöz tartozó bejegyzéseket kézzel el kell távolítani vagy ki kell kommentezni.

A konfigurációs fájlban definiált összes tömb automatikus elindítása:

`mdadm --assemble --scan`

A konfigurációs fájlban megadható riasztás is, ami a raid tömbök állapotváltozásairól értesítést küld. Ez történhet e-mailben:

`MAILADDR` *e-mail cím*

Vagy tetszőleges program meghívásával:

`PROGRAM` *a meghívandó program teljes elérési útvonala*

A program lehet akár shell script is, így tetszőleges monitorozó rendszerhez könnyen illeszthetőek az md riasztásai. Fontos, hogy végrehajtási jogosultság legyen a fájlon. A program három paramétert kap, amiben szerepel az esemény jellege, az érintett tömb, illetve az érintett eszköz neve.

[6] Nem feltétlenül teszi ténylegesen elérhetetlenné az eszköz tartalmát, csak hibásnak jelöli, hogy az md rendszer kezdje el a helyreállítást. Ez azt jelenti, hogy pl. egy RAID-0 tömbből egy eszköz hibásnak jelölése hatástalan lesz.

8. Logikai kötetkezelés Linux alatt, LVM2

A logikai kötetek kezelésére az `lvm` parancs szolgál. Az `lvm` további alparancsokat tartalmaz az egyes műveletek elvégzésére, ezeket az `lvm help`-el kérdezhetjük le. Minden alparancs további paraméterezést igényel, ezekről segítséget a következőképpen kaphatunk: `lvm alparancs --help`.

Az alábbiakban csak néhány alapvető parancs kerül ismertetésre, az LVM2 ezeknél lényegesen több szolgáltatást és paraméterezési lehetőséget nyújt.

Listázások:

- `pvs` - fizikai kötetek listázása
- `vgs` - kötetcsoportok listázása
- `lvs` - logikai kötetek listázása

Részletes információ lekérdezése:

- `pvdisplay` - fizikai kötet lekérdezése
- `vgdisplay` - kötetcsoport lekérdezése
- `lvdisplay` - logikai kötet lekérdezése

Létrehozás:

- `pvcreate` *eszköz* - fizikai kötet létrehozása
- `vgcreate` *kötetcsoport neve fizikai kötet eszközök* - kötetcsoport létrehozása. Opcionálisan megadható az allokációs egység (extent) mérete a `--physicalextentsize` *méret* paraméterrel.
- `lvcreate` *kötetcsoport neve* - logikai kötet létrehozása. Meg kell adni a kívánt méretet: `--size` *méret*. Opcionálisan megadható a logikai kötet neve: `--name` *név*, enélkül egy alapértelmezett nevet kap.

Törlés:

- `pvremove` *eszköz* - fizikai kötet fejléc eltávolítása az eszközről
- `vgremove` *kötetcsoport* - kötetcsoport törlése
- `lvremove` *kötetcsoport/logikai kötetnév* - logikai kötet törlése

Átméretezések, elemek hozzáadása, elvétele:

- `pvresize --setphysicalvolumesize` *méret eszköz* - fizikai kötet átméretezése
- `vgextend` *kötetcsoport fizikai kötetek* - új fizikai kötet hozzáadás a kötetcsoporthoz
- `vgreduce` *kötetcsoport fizikai kötetek* - Fizikai kötet eltávolítása a kötetcsoportból
- `lvresize --size` *méret kötetcsoport/logikai kötetnév* - logikai kötet átméretezése
- `lvextend`, `lvreduce` - az `lvresize` speciális esetei (növesztés, zsugorítás)

A legegyszerűbb LVM felépítési műveletsor a következő:

1. Fizikai kötetek létrehozása
2. Kötetcsoport létrehozása a fizikai kötetből
3. Logikai kötetek létrehozása a kötetcsoportban.

A logikai kötetek a `/dev/mapper` könyvtár alatt találhatóak meg *kötetnév-kötetcsoportnév* formában, ugyanúgy használhatóak, mint bármely más partíció.

Ellentétben a hagyományos particionálási szokással, ahol a partíciók általában a rendelkezésre álló helyet teljes egészében kitöltik és a szabad hely a fájlrendszerben jelenik meg, logikai kötetkezelésnél szokásos a kötetcsoportban mindig valamennyi allokálatlan helyet kihagyni, szükség esetén a logikai kötetek méretét növelni. Ügyelni kell azonban arra, hogy a fájlrendszerek hajlamosak töredezésre (jelentős teljesítménycsökkenéssel járhat), ha nagy telítettség mellett próbálunk műveleteket végezni rajta. A logikai kötetek és a rajtuk lévő fájlrendszerek méretét még azelőtt ajánlatos megnövelni, hogy az 90% feletti telítettséget érne el.

9. Hibainjektálás

A hibainjektálás elsősorban szoftverfejlesztés, ezen belül is a tesztelés egyik fontos eszköze. Célja, hogy szándékosan és reprodukálható módon tudjunk hibás működést előidézni, hogy a hibakezelésért felelős kódrészletek viselkedését ellenőrizhessük. Fontos, hogy a számos - főleg hardver eredetű - hibajelenséget *szimulálni* kell, mert a tényleges hibaok kiváltása nehéz, esetleg az eszköz fizikai tönkretételével lenne csak lehetséges.

A Linux kernel is rendelkezik beépített hibainjektálási lehetőséggel, melyet fordítási időben kell engedélyezni, és a fordítást debug szimbólumok megtartásával kell végezni. Ilyen módon fordított kernelt éles rendszerekben nem találunk, mert mérete lényegesen nagyobb egy normális módon fordított kernelnél, a teljesítménye is rosszabb lehet.

A hibainjektálási beállításokat egy speciális (*debugfs*) fájlrendszerből végezhetjük. Ez, hasonlóan a *proc* vagy *sys* fájlrendszerhez, olyan speciális fájlokból áll, amik valamilyen kernelen belüli adatszerkezetet tesznek kívülről elérhetővé. Nincs mögötte fizikai tárolóeszköz (ezért a `none` kulcsszó). A debugfs-t a következőképpen lehet felcsatolni:

`mount -t debugfs none` *könyvtár neve*

Blokkos eszközök működésébe történő hibainjektálás a *fail_make_request* interfész használatával valósítható meg. Ez az adott eszközön végzendő minden rendszerhívást érinti, minden esetben hibajelzés lesz a visszatérési érték. Szelektív hibázást (pl csak `read()` vagy `write()`), vagy megválasztott típusú hibajelzést, esetleg hibajelzés nélküli *„csendes”* hibajelenséget csak felhasználói módú hibainjektorral (Systemtap[7]) lehet megvalósítani, ezt itt most nem tárgyaljuk. Blokkos eszköz működésébe hiba injektálása a következő értékek beállítását igényli (feltételezve, hogy a debugfs-t a `/debug` könyvtár alá csatoltuk):

- `/debug/fail_make_request/interval` - beállítja, hogy hány műveletenként térjen vissza hibával. Alapértelmezetten 1, minden műveletnél hibázik.
- `/debug/fail_make_request/probability` - az előfordulási valószínűséget állítja százalékban 0-100 között. Az előzővel együtt használva széles tartományban lehet állítani a hibák gyakoriságát.
- `/debug/fail_make_request/times` - meghatározza, hogy összesen hányszor injektáljon hibát rendszerhívásba. -1-es érték azt jelenti, hogy korlátlan sokszor.
- `/debug/fail_make_request/verbose` - minden egyes hiba injektálásakor a kernel logba kerülő üzenet részletességét állítja: 2 - teljes hívási lánc, 1 - egysoros üzenet, 0 - semmi

Hogy a hibainjektor mely eszközök esetén működjön a következő fájlba írt „1” értékkel választhatjuk ki:

`/sys/block/`*blokkos eszköz neve*`/make-it-fail`

[7] http://sourceware.org/systemtap/index.html

10. iSCSI target konfigurálása Linux alatt

Linux alatt a legelterjedtebb iSCSI target (célpont, szerver) implementáció az iSCSI Enterprise Target (*iet*). Ennek feladata, hogy helyi blokkos eszközökhöz nyújtson hozzáférést hálózaton keresztül csatlakozó initiatorok (kezdeményezők, kliensek) számára. Három fő komponensből áll:

- Kernel módú szerver - alacsony szintű blokkos IO műveletek gyorsítására
- Felhasználói módú szerver - a hálózati kapcsolatok kezelése, iSCSI protokoll implementáció
- Felhasználói adminisztrációs segédprogram (`ietadm`) - ennek segítségével konfigurálható, akár működés közben is

Legegyszerűbb esetben a `/etc/ietd.conf` alatti konfigurációs állományban adjuk meg az iet által kiajánlott iSCSI célpontokat, az `iscsi-target` indító scriptje ez alapján konfigurálja fel a szervert. Itt csak az alapszintaxis kerül ismertetésre, ami a target kiajánláshoz elegendő, a többi - elsősorban teljesítmény és hozzáférési jogosultságokat szabályozó - paraméterről az iet dokumentációja részletes ismertetést tartalmaz.

Minden target kell, hogy rendelkezzen egy kvalifikált névvel, aminek a szintaxisa hasonlít a domain nevekéhez. Például:

`iqn.2009-03.local.ftslab.host:sandboxvolume`

1. Az első tag kötelezően `iqn` az *iSCSI qualified name* rövidítése.
2. Ezt egy dátum követi, ami regisztrációs időpontot jelöl. Külvilággal nem érintkező zárt hálózatban ez tetszőleges lehet.
3. A gép domain neve, a DNS-nél megszokotthoz képest fordított sorrendben, az általánostól a konkrét felé haladva. Megjegyzendő, hogy nem feltétlenül kell ennek összhangban lennie a gép tényleges DNS-ben megadott domain nevével, de egyszerűsíti az azonosítást.
4. Kettőspont után a gépen belüli célpont azonosítója. Egy gépen belül tetszőlegesen sok célpont is lehet definiálva. A célpontot is lehet hierarchikus, pontokkal elválasztott kvalifikált névvel ellátni.

Fontos még megjegyezni, hogy egy célponton belül további logikai egységek (*LUN - Logical Unit Number*) definiálhatók. Minden célpontnak legalább egy logikai egységet kell tartalmaznia. Egy logikai egységhez a következőket kell megadni:

- A logikai egység száma (LUN)
- A kiajánlott blokkos eszköz vagy fájl elérési útvonala. Az itt megadott fájl tartalmát fogják a célponthoz csatlakozó kezdeményezők olvasni, írni.
- A hozzáférés mód.
- Opcionális SCSI sorozatszám (*serial number*. Lényegében egy tetszőleges string lehet. Célszerű minden célpontnak egy egyedi sorozatszámot megadni.[8]

A hozzáférési módok lehetnek:

- `fileio` - az iet egyszerű fájl API-t használja. Reguláris fájlok esetén mindenképpen ezt kell használni, blokkos eszközöknél is használható. Szerver oldali cache-elést biztosít, szekvenciális átvitel gyors, viszont a véletlen elérések némileg lassabbak.

[8] Egyes iSCSI initiatorok, pl. a VMware ESX Server úgy értelmezi a sorozatszámot, hogyha több azonos sorozatszámú LUN-t lát, akkor feltételezi, hogy ezek mind valójában egyazon tárhelyet jelölik, csak több különböző útvonalon érhetőek el (multipath hibatűrés). Ha nincs megadva sorozatszám vagy több LUN azonos sorozatszámmal rendelkezik, az helytelen működést eredményezhet.

- `blockio` - alacsony szintű blokkos hozzáférést használ. Csak blokkos eszközöknél alkalmazható. Megkerüli a kernel cache alrendszerét, ennek eredményeképpen az elszórt (tehát rosszul cache-elhető) véletlen elérések felgyorsulnak.
- `nullio` - nem tárol el semmit, olvasásnál véletlen adatokat ad vissza. Tesztelési célokat szolgál.

Egy célpont definíciója tehát így nézhet ki:

```
Target iqn.2007-09.local.ftslab.host:sandboxvolume
       Lun 0 Path=/dev/mapper/StoreVG-SandboxVol,Type=fileio,ScsiSN=SERIAL-00007
```

Ügyelni kell arra, hogy a LUN definíciójába ne helyezzünk el üres (whitespace) karaktereket a vesszők közé, továbbá ne törjük meg a sort sehol. Sajnos az iet nem ad hibaüzenetet hibás szintaktikájú fájl esetén, figyelmen kívül hagyja a hibás sorokat.

11. iSCSI initiator konfigurálása Linux alatt

A hálózati tárhelyet igénybevevő eszközön belül az iSCSI initiator feladata csatlakozni a célpontokhoz és az általa rendelkezésre bocsátott tárhely, használatát biztosítani. A Linux alatt eredetileg több egymástól függetlenül fejlesztett iSCSI initiator implementációkat néhány éve egyesítették egy közös projektbe, melynek neve *open-iscsi*.

Az open-iscsi is több komponensből áll:

- kernel módú iSCSI driver - ez felelős a blokkos eszközökért és az iSCSI protokoll adatforgalmat megvalósító részéért
- felhasználói módú daemon (`iscsid`) - az iSCSI kapcsolatok karbantartásáért, felderítéséért felelős szerver folyamat
- felhasználói módú konfigurációs segédprogram (`iscsiadm`) - ennek segítségével konfigurálható

Az `iscsid` folyamatnak futnia kell mielőtt az `iscsiadm`-ot használnánk, ugyanis valójában az `iscsid` végez minden műveletet, az `iscsiadm` csak egy parancssoros felületet biztosít ehhez.

Egy iSCSI kötet felcsatolása két lépésből áll:

1. Az iSCSI szerveren definiált célpontok felderítése
2. Egy kiválasztott logikai egységre (LUN) belépés, melynek során létrejön a lokális gépen egy új blokkos eszköz

A belépés és felderítés a következőképpen végezhető el. (a portszámot meg kell adni, alapértelmezetten az iscsi-target portja 3260):

`iscsiadm --mode discovery --type sendtargets --portal` *IP cím*`:`*portszám*

A felderített célpontok listáját megtekinthetjük:

`iscsiadm --mode node`

Belépés egy célpontra:

`iscsiadm --mode node --targetname` *célpont kvalifikált neve* `--portal` *IP*`:`*port* `--login`

Kilépés a célpontról:

`iscsiadm --mode node --targetname` *célpont kvalifikált neve* `--portal` *IP*`:`*port* `--logout`

A célpont részletes adatainak megjelenítése:

`iscsiadm --mode node --targetname` *célpont kvalifikált neve* `--portal` *IP*`:`*port*

Fontos megjegyezni, hogy a felderítés egy lokálisan tárolt listát hoz létre a célpontokról, tehát akkor is szükség van egyszer a felderítés lépésre, ha enélkül is tudjuk a célpont nevét. A belépés után a `dmesg` paranccsal tekinthető meg a kernel log tartalma, aminek a végén látható, hogy újonnan megjelent egy blokkos eszköz, ami a soron következő SCSI diszk nevet (`/dev/sda, sdb` stb.) kapta meg.

12. iSCSI initiator konfigurálása Windows alatt

Különféle Windows operációs rendszerek is felkészíthetőek iSCSI kezdeményező szerepre. Ehhez egy külön meghajtóprogramot kell telepíteni a Microsoft oldaláról [9].

A telepítés után a iSCSI kezdeményező konfigurációs felülete elérhetővé válik a start menübe települt alkalmazás ikon vagy a vezérlőpult megfelelő elemének indítása után. Egy iSCSI célpontra becsatlakozás és a tárhely használatbavételének a lépései a következők:

- Control panel alatt `iSCSI initiator` megnyitása
- A `General` fülön állítható az initiator neve, nem szükséges megváltoztatni.
- A `Discovery` fülön a `Target Portals` sorolja fel a célpontokkal rendelkező gépeket. Ide kell felvenni az iSCSI célpont IP címét az `Add` gombbal.
- A felvétel után a `Targets` fülön megjelennek az imént felvett gépen definiált célpontok, egyelőre inaktív állapotban.
- A `Log On...` gomb segítségével léphetünk be a célpontra. Az `Automatically Restore this Connection when the System Starts` jelölőnégyzettel állítható be, hogy erre a célpontra minden rendszerindításkor automatikusan be akarunk-e lépni. Ha kiválasztottuk, akkor az előző ablak `Persistent Targets` füle alatti listába bekerül. A belépés végeztével a célpont connected állapotba kerül.
- Végezetül ki kell jelölnünk a kapcsolódott célpontot helyi kötetnek, hogy a Windows meghajtóként kezelni tudja, ezt a `Bound Volumes/Devices` fül alatt tehetjük meg. Ez legegyszerűbben a Bind All gombbal végezhető el.

  Az iSCSI célpontot ezáltal meghajtóként kezelhetjük, ám ahhoz, hogy meghajtó betűjelet kapjon, illetve fájlokat tárolhassunk rajta particionálni kell. Windows alatt erre a MMC (Microsoft Management Console) egyik beépülő modulja szolgál. A particionálás lépései:

- Az `mmc` elindítása (`Run...` menüből)
- `File` menü alatt `Add/Remove Plugins...`
- `Add...` majd a listából a `Disk Management` kiválasztása
- A kiválasztás elfogadása `Ok`-val.

Jobb oldalt a `Disk Management (Local)` pontra navigálva a lemezkezelő eszköz kezelőfelületéhez érkezünk. Az új "lemez" várhatóan még inicializálatlan állapotban lesz, tehát a partíciós táblák adatszerkezete még nincs kialakítva az első blokkjában. Az inicializálást a jobb gombbal felbukkanó menüből választhatjuk ki. Ezután létrehozhatunk rajta új partíciót, az üres területen végzet jobb kattintással felbukkanó menüből indítva. Az új partíció varázslóban betűjelet rendelhetünk hozzá, és egyben a fájlrendszert is létrehozhatjuk rajta. Fontos a `Quick Format` jelölőnégyzet szerepe, ha nincs bejelölve akkor a fájlrendszer létrehozása előtt az eszköz teljes tartományának írás és olvasás ellenőrzése megtörténik, az adathordozó hibáinak felderítése érdekében. Ez igen hosszú folyamat, iSCSI virtuális meghajtó esetén eltekinthetünk tőle. A folyamat végeztével az kijelölt betűjellel ellátott meghajtó éppen úgy használható fájlok tárolására, mint egy fizikai merevlemez.

[9] A segédlet írásakor ezen a címen volt letölthető: http://www.microsoft.com:80/downloads/details.aspx?familyid=12cb3c1a-15d6-4585-b385-befd1319f825&displaylang=en