1 Master Boot Record

MBR je 512 bajtů (1 sektor) od začátku disku se čtyřmi pozicemi na oddíly („partišny“). Bootovatelné oddíly začínají dalším MBR se spustitelným kódem (ukázky `objdump(1)`).

Offset		Obsah
0x000	+0	Startovací kód
0x1BE	+446	Oddíl #1 (16 B, níže)
0x1CE	+462	Oddíl #2 (16 B, níže)
0x1DE	+478	Oddíl #3 (16 B, níže)
0x1EE	+494	Oddíl #4 (16 B, níže)
0x1FE	+510	Magická hodnota 0x55AA (LE)

Offset		Obsah
0x00	+0	Boot? + CHS prvního sektoru
0x04	+4	Typ oddílu + CHS posledního sektoru
0x08	+8	LBA prvního sektoru
0x0C	+12	Počet (512 B) sektorů

```
# dd if=/dev/rsd0c of=mbr bs=0x200 count=1
# hexdump -s 0x1FE -n 2 mbr
# objdump -b binary -D -m i8086 mbr
# hexdump -s 0x1BE -n 16 -C mbr
# hexdump -s 0x1CE -n 16 -C mbr
# hexdump -s 0x1DE -n 16 -C mbr
# hexdump -s 0x1EE -n 16 -C mbr
# fdisk sd0
```

Přečtěte první sektor požadovaného oddílu (od LBA start) a ověřte, že jde opět o MBR (bez oddílů):

```
# dd if=/dev/rsd0c of=part3_start \
       bs=0x200 skip=64 count=1
# hexdump -s 0x1FE -n 2 part3_start
# objdump -b binary -D -m i8086 part3_start
```

2 disklabel (BSD/Solaris)

Dříve to býval *úplně první* sektor, na x86 jej odsunuly MBR o kus dále. Celý je jako `struct disklabel` v souboru `/usr/include/sys/disklabel.h`. Slice je popsán tímto:

Offset		Obsah
0x00	+0	Počet sektorů (spodní 4 B)
0x04	+4	První sektor (spodní 4 B)
0x08	+8	První sektor (horní 2 B)
0x0A	+10	Počet sektorů (horní 2 B)
0x0C	+12	Typ (4.4BSD, swap, SW RAID, ...)
0x0D	+13	Fragmentů na blok
0x0E	+14	Válců na cylgroup
0x10	+16	(Další slice)

Slices popisují i ostatní oddíly „namapované“ do disklabel, např. ext2fs či NTFS. Pole „První sektor“ (offset od *začátku disku*) může vést i mimo aktuální MBR oddíl. V této tabulce (druhý sektor, hned za MBR) je celý disklabel:

Offset		Obsah
0x00	+0	Magická hodnota 0x822564557 (LE)
0x04	+4	Typ/subtyp
0x08	+8	Typ (string, 16 B)
0x18	+24	N (string, 16 B)
0x28	+40	Bajtů na sektor
0x2C	+44	Sektorů na stopu
0x30	+48	Stop na válec
0x34	+52	Válců na disku
0x3C	+56	Sektorů na válec
0x40	+64	Sektorů celkem (spodní 4 B)
0x44	+68	Disk UID (8 B)
0x48	+72	„Alternate cylinders“
0x4C	+76	Začátek použitelné oblasti (hi 2 B)
0x4E	+78	Velikost použitelné oblasti (hi 2 B)
0x50	+80	Začátek použitelné oblasti (lo 4 B)
0x54	+84	Velikost použitelné oblasti (lo 4 B)
0x58	+88	Vlajky
0x5C	+92	Informace o disku (20 B)
0x70	+112	Sektorů celkem (horní 2 B)
0x72	+114	Verze (1 == 48bitové adresování)
0x74	+116	Rezervováno (16 B)
0x84	+132	Magická hodnota 0x822564557 (LE)
0x88	+136	Checksum
0x8A	+138	Počet slices
0x8C	+140	Velikost kódu zavaděče v sektoru 0
0x90	+144	Maximální velikost superbloku FS
0x94	+148	Slices (až 22 × 16 B, výše)

```
# dd if=/dev/rsd0c of=part3_disklabel \
       bs=0x200 skip=65 count=1
# hexdump -s 0x8A -n 2 part3_disklabel
# disklabel sd0
# I=0; while [ ${I} -lt 16 ]; do          \
    hexdump -s $((148+${I}*16)) -n 16 -C \
    part3_disklabel | head -n1; I=$(($I+1)); done
```

Protože máme oddíly/slices v řádu MB, GB či TB, offsety rostou do vysokých čísel. Potřebujeme jednoduchou funkci na přečtení n-bitového čísla jako *little endian*.

```
# hexdump -e '"%#x\n"' -s 0x1FE -n 2 mbr
```

Pozor! Nevyužitý zbytek sektoru s disklabel může obsahovat nevynulovaný obsah „od minule“.

3 FFS (`fs(5)`)

Superbloky (`struct fs` v `/usr/include/ufs/ffs/fs.h`) jsou rozmístěny na diskových sektorech z `newfs -N`. Tyto pozice jsou relativní k začátku oddílu/slice a je tedy vhodné pracovat s příslušným zařízením (např. sd0a místo sd0c). Je ve Vašem zájmu zkontrolovat, zda čtení n-tého sektoru ze zařízení s oddílem odpovídá $n+m$-tému sektoru ze zařízení s celým diskem, kde m = LBA začátku oddílu/slice přečtené Vaším skriptem.

Offset		Superblok: Zajímavý obsah
0x0C	+12	Fragment se začátkem cylgroupy (4 B)
0x10	+16	Fragment se začátkem tabulky inod (4 B)
0x14	+20	Fragment prvního datového bloku (4 B)
0x18	+24	Offset cylgroupy ve válci (4 B)
0x20	+32	Timestamp posledního zápisu (4 B)
0x2C	+44	Počet cylinder group (4 B)
0x30	+48	Velikost bloku v bajtech (4 B)
0x34	+52	Velikost fragmentu v bajtech (4 B)
0x38	+56	Fragmentů na blok (4 B)
0x48	+72	Maska k oddělení čísla bloku (4 B)
0x50	+80	$\log_2$ velikosti bloku (4 B)
0x5C	+92	Nejvyšší počet bloků v cylgroupě (4 B)
0x64	+100	FS blok (číslo fragmentu) → sektor (4 B)
0x68	+104	Skutečná velikost superbloku (4 B)
0x78	+120	Počet inod na blok (4 B)
0xB8	+184	Počet inod na cylgroupu (4 B)
0xBC	+188	Počet fragmentů na cylgroupu (4 B)
0x438	+1080	Velikost FS ve fragmentech (8 B)
0x440	+1088	Datových fragmentů celkem (8 B)
0x55C	+1372	Magická hodnota 0x00011954 (LE) (4 B)

```
ufs_sb_bsize() { hexdump -s48 -n4 -e '"%u\n"' $1; }
ufs_sb_fsize() { hexdump -s52 -n4 -e '"%u\n"' $1; }
ufs_sb_fpg() { hexdump -s188 -n4 -e '"%u\n"' $1; }
ufs_sb_ipg() { hexdump -s184 -n4 -e '"%u\n"' $1; }
ufs_sb_ncg() { hexdump -s44 -n4 -e '"%u\n"' $1; }

SBLKS='BEGIN { FS=","; }
    /^ [0-9]/ { for (x = 1; x <= NF; ++x) print $x; }'
for B in `/sbin/newfs -N $1 | awk "${SBLKS}"`; do
        dd if=/dev/r$1 of=sb bs=512 count=4 skip=${B}
        sha256 sb
done

BSIZE=`ufs_sb_bsize sb`; FSIZE=`ufs_sb_fsize sb`;
FPG=`ufs_sb_fpg sb`;IPG=`ufs_sb_ipg sb`;NCG=`ufs_sb_ncg sb`

echo ffs at $1: block/frag size ${BSIZE}/${FSIZE}
echo ${NCG} cylgroups: ${FPG}/${IPG} frags/inodes each
```

Cylinder groupa (`struct cg`) je blok obsahující hlavičku a bitmapy použitých inod/volných bloků (offsety v hlavičce).

Offset		Cylinder Group: Zajímavý obsah
0x04	+4	Magická hodnota 0x00090255 (LE)
0x08	+8	Timestamp posledního zápisu (4 B)
0x0C	+12	Kolikátá cylgroupa je to? (4 B)
0x10	+16	Počet válců (2 B)
0x12	+18	Počet bloků s inodami (2 B)
0x14	+20	Počet bloků s daty (4 B)
0x34	+52	Počty dostupných fragmentů (32 B)
0x5C	+92	Pozice mapy s použitými inodami (4 B)
0x60	+96	Pozice mapy s volnými bloky (4 B)

```
ufs_sb_cgstart() { hexdump -s12 -n4 -e '"%u\n"' $1; }
ufs_sb_fpb() { hexdump -s56 -n4 -e '"%u\n"' $1; }
ufs_cg_dblocks() { hexdump -s20 -n4 -e '"%u\n"' $1; }
ufs_cg_ino_bitmap() { hexdump -s92 -n4 -e'"%u\n"' $1; }
ufs_cg_blk_bitmap() { hexdump -s96 -n4 -e'"%u\n"' $1; }

FPB=`ufs_sb_fpb sb`;
G=0; CG=`ufs_sb_cgstart sb`; while [ $G -lt $NCG ]; do
        dd if=/dev/r$1 of=cg  \
            bs=${FSIZE} count=${FPB} skip=${CG}
        BMP_INODES=`ufs_cg_ino_bitmap cg`
        BMP_BLOCKS=`ufs_cg_blk_bitmap cg`
        BPG=`ufs_cg_dblocks cg`

        echo cylgroup $G @frag $CG: $BPG data blocks

        hexdump -C -s ${BMP_INODES} -n $((${IPG}/8)) cg
        hexdump -C -s ${BMP_BLOCKS} -n $((${BPG}/8)) cg

        CG=$((${CG}+${FPG})); G=$((${G}+1));
done
```

Jak najít inodu podle čísla v tabulce za její cylgrupou:

```
ufs_ino() {
        ICG=$((${2}/${IPG})); POS=$((${2}%${IPG}));
        CG=$((`ufs_sb_cgstart sb`+${ICG}*${FPG}));
        echo inode $2 in cylgroup $ICG @frag $CG

        INOSIZE=128; INOTBL=$((${CG}+${FPB}))
        FCNT=$((${IPG}*${INOSIZE}/${FSIZE}));
        dd if=/dev/r$1 of=inotbl        \
            bs=${FSIZE} count=${FCNT} skip=${INOTBL}
        dd if=inotbl of=ino_$2.ino \
            bs=${INOSIZE} count=1 skip=${POS}
}

ufs_ino wd0a 2
ufs_ino wd0a 50000
```

Inody jsou `struct ufs1_dinode` z `/usr/include/ufs/ufs/dinode.h`. Typ souboru (a další) v manuálu `stat(2)`.

Offset		UFS Inode: Zajímavý obsah
0x00	+0	File mode (2 B)
0x02	+2	Počet linků (2 B)
0x04	+4	Velikost (8 B)
0x28	+40	Ukazatele na datové bloky (12×4 B)
0x58	+88	Nepřímý ukazatel (4 B)
0x5C	+92	Dvojitě nepřímý ukazatel (4 B)
0x60	+96	Trojitě nepřímý ukazatel (4 B)
0x64	+100	Vlajky (4 B)
0x68	+104	*Celkem* zabraných *sektorů* (4 B)

```
DIRECT=$((12*${FPB}));  FPOINTERS=$((${BSIZE}/4*${FPB}))
INDIR1=$((${DIRECT} + ${FPOINTERS}))
INDIR2=$((${INDIR1} + ${FPOINTERS}*${FPOINTERS}))

ufs_ino_blk() {
        if [ $3 -lt ${DIRECT} ]; then
                POS=$((40+${3}/${FPB}*4))
                hexdump -s${POS} -n4 -e'"%u\n"' $2;
        elif [ $3 -lt ${INDIR1} ]; then
                IND1=`hexdump -s88 -n4 -e'"%u\n"' $2;`
                [ ! -f ind1 ] && dd if=/dev/r$1 of=ind1 \
                    bs=${FSIZE} count=${FPB} skip=${IND1}
                POS=$(((${3}-${DIRECT})/${FPB}*4))
                hexdump -s${POS} -n4 -e'"%u\n"' ind1;
        elif [ $3 -lt ${INDIR2} ]; then
                echo "doubly indirect!"
        else    echo "triply indirect!"
        fi
}
ufs_ino_size() { hexdump -s4 -n8 -e'"%u "' $1 |
    { read HI LO; echo $(((${HI}<<32)|${LO})); } }

ufs_ino_save() {
        SIZE=`ufs_ino_size ino_$2.ino`
        TOTAL=$(((${SIZE}+${FSIZE}-1)/${FSIZE}))
        rm -f ind1 ino_$2.dat*
        B=0; while [ ${B} -lt ${TOTAL} ]; do
                BLK=`ufs_ino_blk ino_$2.ino $B`
                dd if=/dev/r$1 of=blk count=${FPB} \
                    bs=${FSIZE} skip=${BLK}
                cat blk >> ino_$2.datx
                B=$((${B}+${FPB}));
        done
        # bash: truncate -s [ufs_ino_size] ino_$2.dat
        dd if=ino_$2.datx of=ino_$2.dat count=1 bs=${SIZE}
        rm -f blk ind1 ino_$2.datx
}
ufs_ino_save wd0a 2
```

Adresáře jsou soubory obsahující následující „dirent“ záznamy jeden za druhým. Stejné v ext2/3/4.

Offset		Directory Entry: Obsah
0x00	+0	Číslo inode
0x04	+4	Délka záznamu včetně hlavičky (2 B)
0x06	+6	Délka názvu (1 B)
0x07	+7	Typ souboru (1 B)
0x08	+8	Název souboru

Typ souboru je též uveden v jeho inodě, v poli „file mode“. Symbolické odkazy mohou mít cíl v 60 bytech s ukazateli na bloky. (zabraných sektorů je rovno 0)

```
dir_ino() { hexdump -s$(($2+0)) -n4 -e'"%u\n"' $1; }
dir_len() { hexdump -s$(($2+4)) -n2 -e'"%u\n"' $1; }
dir_type() { hexdump -s$(($2+6)) -n1 -e'"%u\n"' $1; }
dir_nlen() { hexdump -s$(($2+7)) -n1 -e'"%u\n"' $1; }
dir_name() { hexdump -s$(($2+8)) -n$3 -e'"%.'$3's\n"' $1; }

eval `stat -s $1`
OFF=0; while [ ${OFF} -lt $st_size ]; do
        INO=`dir_ino $1 ${OFF}`
        LEN=`dir_len $1 ${OFF}`
        NLEN=`dir_nlen $1 ${OFF}`
        TYPE=`dir_type $1 ${OFF}`
        NAME=`dir_name $1 ${OFF} ${NLEN}`

        echo "$INO ($TYPE) $NAME"

        OFF=$((${OFF}+${LEN}))
done
```

Projít bitmapu inod/bloků znamená z cylgroupy `${G}` přečíst, co ukazují příkazy `hexdump` v sekci o cylgroupách.

```
BYTES=$((${IPG}/8))
dd if=cg of=inobm bs=1 off=${BMP_INODES} count=${BYTES}

B=0; while [ ${B} -lt ${BYTES} ]; do
        BYTE=`hexdump -s $B -n 1 -e '"%u\n"' inobm`;
        b=0; while [ ${b} -lt 8 ]; do
                INODE=$((${G}*${IPG} + ${B}*8 + ${b}));
                [ $((${BYTE}&(1<<${b}))) -ne 0 ] && \
                    echo inode $INODE is marked;
                b=$((${b}+1));
                done
        B=$((${B}+1));
done
```

4 ext2

Alokační i adresační jednotkou je 1K/2K/4K blok (Linux má velikost FS bloku limitovanou velikostí stránky).
Superbloky (`struct ext4_super_block`, `fs/ext4/ext4.h`) jsou rozmístěny na logických (FS) blocích z `mke2fs -n`; primární je vždy 1024 B od začátku oddílu, další dle `http://git.kernel.org/cgit/fs/ext2/e2fsprogs.git/tree/lib/ext2fs/closefs.c#n36`.

Offset		Superblok: Zajímavý obsah
0x00	+0	Počet inod (4 B)
0x04	+4	Počet bloků (4 B)
0x14	+20	První datový blok (4 B)
0x18	+24	$\log_2$ velikosti bloku /1024 (4 B)
0x1C	+28	$\log_2$ velikosti fragmentu /1024 (4 B)
0x20	+32	Počet bloků ve skupině (4 B)
0x24	+36	Počet fragmentů ve skupině (4 B)
0x28	+40	Počet inod ve skupině (4 B)
0x38	+56	Magická hodnota 0xEF53 (LE)
0x3A	+58	Stav FS (clean == 0) (2 B)
0x58	+88	Velikost inody (2 B)
0xFE	+254	Velikost popisovače skupiny (2 B)

```
ext2_sb_bsize() {
        LOG_BSIZE=‘hexdump -s24 -n4 -e’"%u\n"’ $1‘
        echo $((1024 * 1<<${LOG_BSIZE}));
}
```

Popisovače skupin (`struct ext4_group_desc`) následují hned za příslušnou kopií superbloku, na hranici FS bloku. Typicky jsou 32 B velké, není-li ext4 vytvořen jako 64bitový.

Offset		Popisovač skupiny: Zajímavý obsah
0x00	+0	Blok s bitmapou alokovaných bloků
0x04	+4	Blok s bitmapou alokovaných inod
0x08	+8	Začátek tabulky inod

Inody (`struct ext4_inode`) jsou staticky alokované v tabulkách, na které ukazují popisovače skupin; vzestupné číslování od 1 přesně určuje skupinu, ve které inode je.

Offset		ext2/3/4 Inode: Zajímavý obsah
0x00	+0	File mode (2 B)
0x04	+4	Velikost (spodní 4 B)
0x1A	+26	Počet hardlinků (2 B)
0x1C	+28	Počet bloků (spodní 4 B)
0x20	+32	Vlajky (4 B)
0x28	+40	Ukazatele na datové bloky (12×4 B)
0x58	+88	Nepřímý ukazatel (4 B)
0x5C	+92	Dvojitě nepřímý ukazatel (4 B)
0x60	+96	Trojitě nepřímý ukazatel (4 B)
0x6C	+108	Velikost (horní 4 B)

5 ext4

Je-li nastavena u inody vlajka `EXT4_INODE_EXTENTS` (0x80000), potom místo dvanácti přímých ukazatelů na bloky je uzel stromu extentů:

Offset		Hlavička uzlu stromu
0x00	+0	Magická hodnota 0xF30A (LE)
0x02	+2	Počet položek za hlavičkou (2 B)
0x04	+4	Maximální počet položek (2 B)
0x06	+6	Hloubka tohoto uzlu ve stromě (2 B)
0x08	+8	Generace uzlu (Lustre) (4 B)

Hloubka uzlu > 0 znamená 12bytové ukazatele „hlouběji“:

Offset		Vnitřní uzel stromu extentů
0x00	+0	První FS blok této části stromu
0x04	+4	Blok s nižší úrovní (spodní 4 B)
0x06	+6	Blok s nižší úrovní (horní 2 B)

Je-li hloubka uzlu rovna 0, následují 12bytové listy:

Offset		List stromu extentů
0x00	+0	První logický (FS) blok extentu
0x04	+4	Počet FS bloků v extentu (2 B)
0x06	+6	Fyzický FS blok začátku (horní 2 B)
0x08	+8	Fyzický FS blok začátku (spodní 4 B)

Více na `https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout`

V praxi pozor na *flexible* skupiny (bitmapy a tabulky inod pro více skupin již u té první): např. offsety v popisovačích 1-15 se vztahují k začátku skupiny 0.

6 Vysvětlivky a rady

- IPG znamená „inodes per group“, NCG „number of cylgroups“, FPB „fragments per block“, ...
- Uvědomte si rozdíl mezi sektory a bloky FS.
- Slepým přepisováním se nic nenaučíte (proto je to po sekcích).
- Nepište si do systému, který skriptem procházíte.
- Můžete psát víc modulů .sh/.awk a volat je navzájem, pokud si to předem rozmyslíte.
- dumpfs(8) je dobrý pro zpětnou kontrolu. Nepoužívat k získávání dat!
- Rychlost a efektivitu neřešte do detailů (to se dá vždy napsat v C/C++ lépe).
- Nemusíte programovat dvojitě a trojitě nepřímé adresování (extra body za elegantní řešení!).
- `flex_bg` nemusíte podporovat (`mkfs.ext4 -G 1`).

Našli jste chybu? Pošlete ji na `pelikan@storkhole.cz`!

7 mdraid

RAID superblok (`include/uapi/linux/raid/md_p.h`: `mdp_super_t`) je posledních 64 kB na spodním zařízení.

8 LVM2

Physical volumes mají v druhém sektoru (0x200) struktury `label_header` a `pv_header` a v šestém `mda_header`.

Offset		`label_header`, začátek na 0x200
0x00	+0	Magická hodnota, ASCII "`LABELONE`"
0x08	+8	Číslo sektoru tohoto labelu (8 B)
0x10	+16	CRC zbytku tohoto sektoru (4 B)
0x14	+20	Offset obsahu od začátku labelu (4 B)
0x18	+24	Magická hodnota, ASCII "`LVM 001`"
0x20	+32	PV UUID, ASCII bez pomlček (32 B)

Zobrazí je `pvdisplay(8)`. Hodnota `FMTT_MAGIC` je toto:

```
$ sudo hexdump -C -s 0x1004 /dev/loop1
00001004  20 4c 56 4d 32 20 78 5b  \
 35 41 25 72 30 4e 2a 3e  | LVM2 x[5A%r0N*>|
```

Offset		`mda_header`, začátek na 0x1000
0x00	+0	Checksum zbytku hlavičky (4 B)
0x04	+4	Magická hodnota, `FMTT_MAGIC`
0x14	+20	Verze (4 B)
0x18	+24	Začátek této hlavičky v bytech (8 B)
0x20	+32	Velikost ring bufferu metadat (8 B)
0x28	+40	Seznam raw_locations

Volume groups a logical volumes mají od dalšího sektoru metadata v textovém formátu. `extent_size` v sektorech.

```
# dd if=/dev/loop0 of=vg.txt bs=512 skip=9 count=42
# less vg.txt
název_VG { [...] physical_volumes { [...] }
[...] logical_volumes { [...] } [...] }
```

Do ring bufferu s metadaty připíše LVM vždy na hranici sektoru nový textový layout s vyšším `seqno` a aktualizuje checksum v `mda_header` – změna je tedy atomická a standardně se děje na začátku každého PV (`pvcreate(8) --metadatacopies`).