Tallinna Pedagoogikaülikool
Matemaatika – loodusteaduskond
Informaatika osakond

Kalle Tabur

Failisüsteemid

proseminaritöö

Juhendaja: Marek Kusmin

Tallinn 2001

1. Sissejuhatus 4
2. File Allocation table File system 5
2.1. FAT12 5
2.1.1. Alglaadesektor 5
2.1.2. BIOSi Parameetrite plokk 6
2.1.3. Juurkataloog. 7
2.1.4. Kataloogi struktuur: 8
2.1.5. Failinimi ja faililaiend 8
2.1.6. Faili atribuudid 9
2.1.7. Ajatempel 9
2.1.8. Failipaigutustabel 10
2.2. FAT16 failisüsteem 11
2.2.1. BIOS-i parameetrite plokk. 12
2.2.2. Meedia kirjeldaja 12
2.2.3. Laiendatud BIOS Parameetrite Plokk. 13
2.3. VFAT 14
2.4. FAT32 16
3. New technology file system 19
3.1. Master File Table 19
3.1.1. MFT struktuur: 19
3.1.2. Fail ja kataloogid 20
3.1.3. Faili atribuudid: 23
3.1.4. Streams 24
3.1.5. Pakitud vood 24
3.1.6. Krüpteeritud vood 25
3.1.7. Hõredad vood 25
3.1.8. Kõva link 25
3.1.9. Kvoot 26
4. Second extended File System 27
4.1. Plokid ja fragmendid 27
4.2. Grupid 28
4.3. Superplokk 28

4.4. Gruppi kirjeldav andmestruktuur 31
4.5. Bitmap 31
4.6. Inode 32
4.7. Kataloogid 34
4.8. Paigutuste algoritmid 35
4.9. Käitumine vigade korral 35
Kokkuvõte 37
Kasutatud kirjandus: 38

1. SISSEJUHATUS

Käesoleva proseminaritöö eesmärgiks on selgitada tuntumaid failisüsteeme. Töö sobib abimaterjaliks informaatika eriala üliõpilastele ning on kasutatav lisamaterjalina loengusarjades "Sissejuhatus informaatikasse" ning "Operatsioonisüsteemid I” ja ” Operatsioonisüsteemid II". See eeldab mõningaid teadmisi arvutist ja informaatika põhimõistete tundmist. Töö esimeses osas kirjeldatakse erinevaid FAT failisüsteeme, versioonide erinevusi ja ülesehitust. Teises osas käsitletakse NTFS-i ja kolmandas ext2 failisüsteemi ülesehitust. Iga failisüsteemi juures on välja toodud just sellele süsteemile omasemad jooned: ketta ülesehitus, failide paiknemine ja kättesaadavus, põhilisemad atribuudid, kataloogid ja nende struktuur.

2. FILE ALLOCATION TABLE FILE SYSTEM

Failisüsteem File Allocation Table (edaspidi FAT) sai alguse 1977 aastal. See arendati välja Microsoft Corporationi poolt. Seda failisüsteemi hakati laialdaselt kasutama operatsioonisüsteemi (edaspidi opsüsteem) Microsofti Disk Operating System (MS-DOS) koosseisus. Algselt kavandati seda kasutada flopiketastel, seetõttu oli esialgne maksimaalne informatsiooni maht, mida failisüsteem suutis mahutada, 8 MB. Kõvaketaste suurenedes oli vajalik suurendada ka failisüsteemi poolt hallatava informatsiooni mahtu ja seepärast tekkis vajadus failisüsteemi edasi arendada. Loodi FAT16 ja hiljem FAT32. Vahepeal tekkis soov kasutada pikemaid failinimesid algse kaheksa pluss kolm asemel. Kasutusele võeti Virtual FAT (VFAT), mis toetab pikki failinimesid. Algselt välja töötatud FAT-i tuntakse praegu kui FAT12. Viimast failisüsteemi kasutatakse siiamaani flopiketaste juures.

2.1. FAT12

FAT failisüsteemi põhiliseks üksuseks on köide (*volume*). Kasutaja seisukohast vaadates vastab köide kettasalvesti tähisele (*drive letter*) käsureal. Flopiketas sisaldab ühte köidet, samas kõvaketta võib jagada mitmeks köiteks. FAT12 võib köide mahutada kuni 8 MB. Köide on jaotatud neljaks osaks: reserveeritud piirkond, milles sisaldub ka alglaadesektor (*boot sector*), failipaigutustabelid, juurkataloog (*root directory*), andmepiirkond (*data area*).

2.1.1. Alglaadesektor

Alglaadesektor on FAT failisüsteemi süda. See sisaldab kõikvõimalikku informatsiooni, mida opsüsteem vajab ketta draiverite väljakutseks, kui tekib vajadus sektoritele ligi pääseda. Alglaadesektor asub alati salvestusseadme esimeses sektoris.
Järgnev tabel kirjeldab alglaadesektori struktuuri:
Esimene kirje selles tabelis sisaldab instruktsioone alglaadimise jaoks. Kui arvuti käivitatakse, siis lastakse käima madala taseme tegumid, millest viimane saadetakse laetava ketta esimese sektori esimest baiti lugema. Sealt saadetakse tegum alglaaduri koodi (*bootstrap code*) juurde. Alglaadur alustab aadressilt 1Eh ja laiendab ennast kuni sektori lõpuni. Kuna esimene rada on täielikult reserveeritud võib alglaadur enadale hõivata veel sektoreid, kui selleks peaks vajadus tekkima.

Tabel 1 [12]

Osa	Nihe	Pikkus	Kirjeldus
kood	00h	3 baiti	instruktsioon, mis suunab programmi alglaaduri koodi juurde
OS nimi	03h	8 baiti	kasutatud op süsteemi nimi
BIOS-i Parameetrite Plokk	0Bh	2 baiti	baite sektori kohta
	0Dh	1 bait	Sektoreid klastri kohta
	0Eh	2 baiti	Reserveeritud sektorite arv
	10h	1 bait	FAT-ide arv
	11h	2 baiti	Juurkataloogi kirjete arv
	13h	2 baiti	Sektorite arv köites
	15h	1 bait	Media kirjeldaja
	16h	2 baiti	Sektoreid FAT-i kohta
	18h	2 baiti	Sektoreid radade kohta
	1Ah	2 baiti	Lugemis-, kirjutamispeade arv
	1Ch	2 baiti	Varjatud sektorite arv
kood	1Eh	varieeruv	Alglaaduri kood

2.1.2. BIOSi Parameetrite plokk

BIOS Parameter Block: Need väljad sisaldavad informatsiooni, mida vajab BIOS (*Basic Input Output System*), kui ta kutsub välja madalama taseme funktsioonid.

Baite sektori kohta (*Bytes per Sector*): Lubatud on järgmised väärtused: 512, 1024, 2048 või 4096 baiti. Probleeme võib tekitada asjaolu, et suurem osa süsteemidest ei oska käsitseda sektoreid, mille suurus on midagi muud kui 512 baiti.

Sektoreid klastri kohta (*Sectors per cluster*): Võimalikud väärtused on: 1, 2, 4, 8, 16, 32 või 128. Tavaliselt on välja kujunenud kombinatsiooniks
“baite_klastri_kohta” × “sektoreid_klastri_kohta”.

Reserveeritud sektorite arv näitab seda piirkonda, mida alglaadesektor oma vajaduseks kasutab.

FAT-i koopiate arv: Tavaliselt on tegemist kahe FAT-iga. Kuid pole keelatud ka rohkemate FAT-ide arv, kuigi sellist toimingut ei pruugi paljud süsteemid toetada. Mitme FAT-i olemasolu aitab vähendada andmete kaotsiminekut juhul, kui üks FATidest peaks saama kahjustatud.

Juurkataloog: sisestavate kirjete arv on piiratud. FAT12 on see tavaliselt 224 kirjet.

Sektorite arv köites: Sellesse kogusummasse on arvestatud köite kõigi nelja piirkonna kasutuses olevad sektorid.
Meedia kirjeldaja (*Media Descriptor*): See väli võib sisaldada järgnevaid väärtusi:

tabel 2 [12]

Väärtus	Seade	Kirjeldus
F0h	3,5”	2 poolt, 80 rada, 18 sektorit / rajal
F8h	kõvaketas	muutuvate andmetega
F9h	5,25” 3,5”	2 poolt, 80 rada, 15 sektorit / rajal 2 poolt, 80 rada, 9 sektorit / rajal
FAh	5,25” 3,5”	1 pool, 80 rada, 8 sektorit / rajal 1 pool, 80 rada, 8 sektorit / rajal
FBh	5,25” 3,5”	2 poolt, 80 rada, 8 sektorit / rajal 2 poolt, 80 rada, 8 sektorit / rajal
FCh	5,25”	1 pool, 40 rada, 9 sektorit / rajal
FDh	5,25”	2 poolt, 40 rada, 9 sektorit / rajal
Feh	5,25”	1 pool, 40 rada, 8 sektorit / rajal
FFh	5,25”	2 poolt, 40 rada, 8 sektorit / rajal

Sektoreid FAT-i kohta, kirjeldab mitu sektorit üks FAT-i koopia enda alla võtab.

2.1.3. Juurkataloog.

FAT failisüsteem on sarnaselt UNIX-is kasutatavate failisüsteemidega hierarhilise ülesehitusega. Juurkataloog on vanemaks kõikidele failidele ja alamkataloogidele. Iga kirje kataloogi nimistus on 32 baiti pikk. Ainsaks fikseeritud asukohaga kataloogiks on juurkataloog. Juurkataloog on erinev ka seetõttu, et selles asub köite märgend (*volume label*). Juurkataloog on vanemate FAT-i versioonides fikseeritud suurusega, mis on paika pandud BPBs.
Alamkataloogide loomise käigus tehakse läbi järgmine protseduur:
hõivatakse mingi hulk klastreid ja vajadusel tühjendatakse need, seejärel luuakse sellesse kaks kirjet:
“.” – viitab alamkataloogile endale
“..” – viitab vanemkataloogile

Kui hõivatud klastrid on täis, siis haaratakse neid juurde, kuid eelpool kirjeldatud protseduuri enam läbi ei viida.

2.1.4. Kataloogi struktuur:

tabel 3 [12]

Nihe	Pikkus	Kirjeldus
00h	8 baiti	failinimi
08h	3 baiti	faililaiend
0Bh	1 bait	faili tunnused
0Ch	10 baiti	reserveeritud
16h	2 baiti	kellaaeg, millal faili viimati muudeti
18h	2 baiti	kuupäev, millal faili viimati muudeti
1Ah	2 baiti	faili esimene klaster
1Ch	4 baiti	faili suurus

2.1.5. Failinimi ja faililaiend

Failinimi võib olla kuni 8 baiti pikk. Kui failinimi on lühem, täidetakse vabaks jäänud kohad tühikutega (ASCII: 20h). Faililaiendi pikkuseks on 3 baiti. Kui faililaiend on lühem, siis täidetakse ülejäänud kohad tühikutega, samuti nagu failinimede puhul. Failinimedes on lubatud kasutada kõiki tähti inglise keele tähestikust (suured tähed) ja numberid 0..9. Esimene bait failinimes on eriline ja seda koheldakse kindlate reeglite põhjal:

1. väärtust 00h tõlgendatakse viimase kirjena kataloogis
2. väärtus 05h vahetatakse ASCII koodiga E5h
3. Ei tohi sisaldada väärtust 20h ehk tühikut
4. väärtus E5h esitab vaba kirjet

Järgmisi sümboleid ei ole lubatud kasutada ei failinimes ega faililaiendis:

1. kõik sümbolid, mis jäävad kooditabelis 20h ettepoole välja arvatud 05h
2. järgnevad sümbolid: 22h ("), 2Ah (*), 2Bh (+), 2Ch (,), 2Eh (.), 2Fh (/), 3Ah (:), 3Bh (;), 3Ch (<), 3Dh (=), 3Eh (>), 3Fh (?), 5Bh ([), 5Ch (\), 5Dh (]), 7Ch (|)

Et piirata ühilduvusest tingitud probleeme võiks kasutada järgmisi sümboleid:

1. Suurtähti A..Z

2. numbreid 0 ja 1
3. järgmisi sümboleid: #, $, %, &, ', (,), -, @

2.1.6. Faili atribuudid

See väli on ainult ühe baidi pikkune ja kirjeldab failile või alamkataloogile omistatud atribuute. Neid atribuute on kuus:
kirjutamiskaitse (*read only*), varjatud fail (*hidden file*), süsteemne fail (*system file*), köite nimi (*volume name*), alamkataloog (*subdirectory*), arhiveerimis atribuut (*archive bit*) ja kaks reserveeritud atribuuti.
Kirjutamiskaitset kasutatakse failide ja kataloogide juures selleks, et mõni programm neid automaatselt kustutama ei hakkaks, samuti kaitseks oskamatute kasutajate vastu.
Varjatud fail või kataloog on peidetud selliste kasutajate eest, kellel pole nendega midagi pistmist. Süsteemile on sellised failid loomulikult nähtavad, samuti kasutajatele, kes teavad neid kasutada.
Süsteemne atribuut näitab, et tegemist on failisüsteemile tähtsa failiga ja igasuguseid tegevusi, mis selliste failidega on seotud, peaks sooritama ülima ettevaatlikkusega.
Arhiveerimise atribuuti kasutatakse varundamise tegemiseks. Arhiveerimise atribuut lisatakse failidele, mis peale eelmist varundamist on loodud, millel on vahetatud nime või mida on muudetud. Varundusprogrammid vaatavad, kas failil on see atribuut. Kui on, siis lisavad selle faili varundatavate juurde ja eemaldavad selle atribuudi.

2.1.7. Ajatempel

Kellaaeg ja kuupäev näitavad, millal faili viimati muudeti. Ajaformaat on järgmine:

tabel 4 [12]

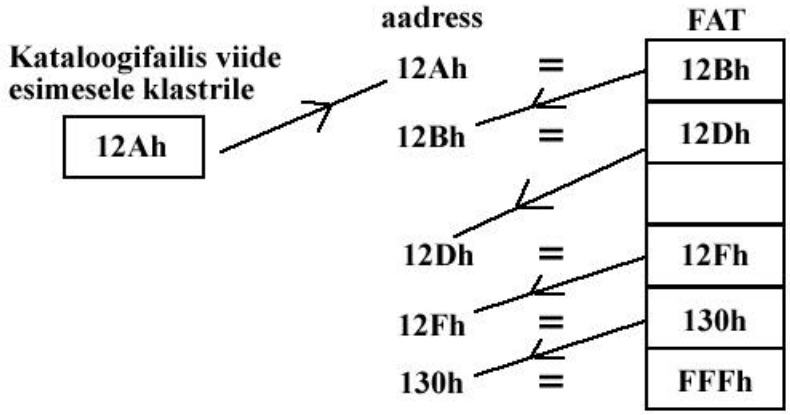
Kellaaeg																
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
tunnid (0-23)					minutid (0-59)						sekundeid (0-29)					0000h

Kuupäev																
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
kuupäev (1-31)					kuu (1-12)				aastad alates 1980 (0-127 -> 1980-2107)							0000h

Sekundite intervall on 2 sekundit, see tähendab, kui on kirjas 29, siis on väärtuseks 2*29 ehk 58 sekundit.
Viide faili esimesele klastrile asub kataloogifailis. Kõik järgmised viited saadakse FAT-ist.

2.1.8. Failipaigutustabel

Kui süsteem soovib mingit faili kasutada, peab ta teadma kus fail asub. Alustatakse klastrite asukohtade kindlaks tegemist. Esimese klastri aadress saadakse kataloogi failist. Järgmiseks saadakse FAT-ist sellelt aadressilt järgmise klastri aadress ja nii edasi kuni klastri järgmise aadressi asemel on faili lõpu tunnus. Et sellist struktuuri paremini kirjeldada, toon näite. Olgu faili suuruseks 20 KB ja klastri suuruseks 4 KB. Kõigepealt vaadatakse kataloogi faili, kus asub viide esimesele klastrile FAT-ist saadakse järgmise klastri aadress – 12Bh. Liigutakse sellele aadressile. Sealt saadakse järgmise klastri aadress. Järgitakse järgmisi viiteid niikaua, kuni ette tuleb faililõputunnus.



Joonis 1

Sellist struktuuri nimetatakse viidetega registriks (*linked list*) ja failipaigutus tabelit viidetega registritabeliks (*linked list table*).
FAT-is on omad kindlad koodid kindlate klastrite jaoks:

tabel 5 [12]

Kood	Tähendus
000h	klaster on vaba
FF0h – FF6h	reserveeritud klastrid
FF7h	klastris on üks või mitu rikutud sektorit
FF8h – FFFh	faili viimane klaster

2.2. FAT16 failisüsteem

See on 16-bitine versioon FAT failisüsteemist. 16-bitine osa kirjeldab, kuidas andmeühikud kettal on jaotatud. FAT16 kasutab 16-bitist numbrit, et identifitseerida iga paigutuslikku ühikut, mida kutsutakse klastriks ja maksimaalselt võib klastreid olla kuni 65536. Klastri suurus defineeritakse ketta alglaadesektoris. FAT16 omab kahte võimalikku failisüsteemi ID numbrit – 04h ja 06h. ID numbrit 04h kasutatakse sellisel juhul, kui kettal on sektoreid vähem kui 65536 (tavaliselt kettasuurus väiksem kui 32 MB), 06h, kui kettal on sektoreid üle 65536. Kuna FAT16 ja FAT12 on üldstruktuurilt üsna sarnased, siis edaspidi keskendun nendele FAT16 osadele, mis erinevad algsest FATi versioonist.

FAT16 failisüsteemi struktuur on identne FAT12-ga ehk siis koosneb järgmisetest osadest: Reserveeritud ala, mis sisaldab ka alglaadesektorit (boot sector), failipaigutusetabelid ehk FAT-id, juurkataloog, andmepiirkond

FAT16 alglaadesektor on sarnase struktuuriga, kuid mitte identne FAT12 omaga. Selguse huvides toon välja terve alglaadesektori struktuuri:

tabel 6 [12]

Osad	Nihe	Pikkus	Kirjeldus
kood	0000h	3 baiti	Kood, mis suunab alglaadekoodi juurde
OS nimi	0003h	8 baiti	Oem ID – vormindatud OS nimi
BIOS Parameetrite Plokk	000Bh	2 baiti	Baiti sektori kohta (*Bytes per Sector*)
	000Dh	1 bait	Sektorit klastri kohta (*Sectors per Cluster*). Tavaliselt on sektor 512 baiti
	000Eh	2 baiti	Reserveeritud sektorid alates ketta algusest
	0010h	1 bait	FAT-i koopiate arv. Tavaliselt on väärtuseks 2
	0011h	2 baiti	Juurkataloogi võimalike kirjete arv - 512 kirjet on soovituslik
	0013h	2 baiti	Vähim sektorite arv – kasutatakse, kui salvestusseade mahutab vähem kui 32 MB
	0015h	1 bait	Meedia kirjeldaja (*Media Descriptor*)
	0016h	2 baiti	Sektoreid FAT-i kohta (*Sectors per FAT*)
	0018h	2 baiti	Sektoreid raja kohta (*Sectors per Track*)
	001Ah	2 baiti	Lugemispeade arv (*Number of Heads*)
	001Ch	4 baiti	Varjatud sektorid (*Hidden Sectors*)
	0020h	4 baiti	Suurim sektorite arv (*large number of sectors*) – kasutatakse, kui salvestusseade mahutab rohkem kui 32 Mb.

Laiendatud BIOS Parameetrite Plokk	0024h	1 bait	Ketta number (*Drive Number*) – Kasutatakse mõne alglaadekoodi poolt (näiteks MS-DOS-is)
	0025h	1 bait	Reserveeritud – Kasutatakse Windows NT poolt otsustamaks ketta terviklikkuse üle
	0026h	1 baiti	Laiendatud alglaaduri signatuur (*Extended Boot Signature*) – kasutatakse näitamaks, et kolm järgmist välja on kasutusel
	0027h	4 baiti	Köite seerianumber (*Volume Serial Number*)
	002Bh	11 baiti	Köite tähis (*Volume Label*) – peaks olema identne sellega köite tähisega, mis asub juurkataloogis
	0036h	8 baiti	Failisüsteemi tüüp (*File System Type*) – peaks olema FAT16
kood	003Eh	448 baiti	Alglaaduri kood (*Bootstrap code*)
signatuur	01FEh	2 baiti	Alglaadesektori signatuur (*Boot Sector signature*)

2.2.1. BIOS-i parameetrite plokk.

Vähim sektorite arv: Selle väli esitab salvestusseadmel asuvate sektorite kogusumma. Sellele lisanduvad sektorid, mis on hõivatud kõigi nelja FAT-i piirkonna poolt. See väli on kasutusel, kui salvestusseadmel on vähem mahtu kui 65536 sektorit. Sellisel juhul on *Master Boot Record*-is failisüsteemi ID 04h. Juhul kui salvestusseadme maht on suurem, siis kasutatakse suurimat sektorite arvu ja selle kirje väärtuseks on 0h.

2.2.2. Meedia kirjeldaja

tabel 7 [12]

Väärtus	Mahutatavus	Füüsiline Vormindus
F0h	2.88 MB	3.5-tolli, 2-poolne, 36-sektorit
F0h	1.44 MB	3.5-tolli, 2- poolne, 18- sektorit
F8h	?	kõvaketas
F9h	720 KB	3.5-tolli, 2- poolne, 9- sektorit
F9h	1.2 MB	5.25- tolli, 2- poolne, 15- sektorit
Fah	?	?
FBh	?	?
FCh	180 KB	5.25- tolli, 1- poolne, 9- sektorit
FDh	360 KB	5.25- tolli, 2- poolne, 9- sektorit
Feh	160 KB	5.25- tolli, 1- poolne, 8- sektorit
FFh	320 KB	5.25- tolli, 2- poolne, 8- sektorit

Suurim sektorite arv: Sektorite arv salvestusseadmel. Kasutatakse juhul, kui sektorite arv ületab 65536. Kui seda kasutatakse, siis MBR-is on failisüsteemi ID 06h, kui ei kasutata, siis seatakse väärtuseks 0h.

2.2.3. Laiendatud BIOS Parameetrite Plokk.

See plokk sisaldab informatsiooni, mida kasutatakse ainult FAT16 poolt.
Salvestusseadme number: Esimese flopiseadme puhul on väärtuseks 00h, esimese kõvaketta väärtuseks on 80h. MS-DOS alglaadur kasutab seda väärtust leidmaks õiget ketast.
Reserveeritud: Algselt kasutati seda selleks, et jätta meelde, millisel silindril alglaadesektor asub. Praegu kasutab seda Windows NT kahe lipu hoidmiseks. Kui noorem bit on üks, siis tuleb ketast alglaadimise ajal kontrollida. Kui aga vanem bit on üks, siis tuleb läbi viia ketta pinna kontroll.
Laiendatud alglaade signatuur. Kui see bait omab väärtust 29h, siis tähendab see seda, et järgmised kolm välja on kasutusel.
Köite seerianumber: Selle väärtus on juhuslik 32–bitine number, mis ühendatult köite tähisega teeb võimalikuks jälgida eemaldatavaid salvestusseadmeid ja kontrollida, kas õige meedia on sisestatud.
Köite tähis: See 11–baidine string peaks olema identne sellega, mis on kirjas juurkataloogis.
Failisüsteemi tüüp: Kasutatakse ainult informatiivsel otstarbel.
Alglaaduri kood: See kood varieerub vastavalt opsüsteemile ja versioonidele. Näiteks kui opsüsteem on MS-DOS, siis alglaadur teeb kindlaks, kus asub fail nimega IO.SYS, loeb osa sellest mällu ja liigub selles failis olevale sisendpunktile (*entry point*).
Alglaadesektori signatuur: Sisaldab signatuuri AA55h, mis teeb BIOS-ile selgeks, et see sektor on laetav. Seda kasutatakse ka teistel juhtudel, kui mõni rakendusprogramm soovib teada saada, et kas on laetud õige sektor.

Kataloogi kirje:

tabel 8 [12]

Nihe	Pikkus	Kirjeldus
00h	8 baiti	failinimi
08h	3 baiti	faililaiend
0Bh	1 bait	atribuut
0Ch	1 bait	reserveeritud Windows NT jaoks
0Dh	1 bait	loomisaeg- ajatempel millisekundites
0Eh	2 baiti	loomise kellaaeg
10h	2 baiti	loomise kuupäev
12h	2 baiti	viimati kasutatud
14h	2 baiti	reserveeritud FAT32 jaoks
16h	2 baiti	viimati muudetud (kellaaeg)
18h	2 baiti	viimati muudetud (kuupäev)
1Ah	2 baiti	viide esimesele klastrile
1Ch	4 baiti	Faili suurus baitides

2.3. VFAT

Tegelikult polegi see omaette failisüsteem vaid failisüsteemi alamsüsteem, mida kasutatakse nii FAT12, FAT16 kui ka FAT32 failisüsteemis. VFAT on moodus kasutada FAT piiratud struktuuris pikki failinimesid. Failinimed salvestatakse UNICODE kooditabeli sümbolites. Vastavalt sellele, kui pikk on faili nimi, loob süsteem mingi arvu kehtetuid 8.3 formaadis kirjeid alamkataloogi, mida nimetatakse LFN (Long Filename) kirjeteks. Esimene kirje koosneb 13 sümbolist, teine kirje mahutab sümbolid 14..26 ja nii edasi, kuni failinimi lõpeb. Failinime pikkuseks võib olla 255 sümbolit. LFN kirjed kirjeldatakse eespool reaalsest kataloogi kirjest (tavapärane 8.3 formaat failinimest) ja kirjed salvestatakse vastupidises järjekorras, see tähendab, et LFN-i kõige viimane kirje on tabelis eespool. Kui vaadata sellist alamkataloogi sisu, siis näeks see välja järgmine:

tabel 9 [12]

kirje nr.	ilma LFN kirjeteta	LFN kirjetega
...	...	...
n	tavaline 1	tavaline 1
n+1	tavaline 2	tavalise LFN 2 - osa 3
n+2	tavaline 3	tavalise LFN 2 - osa 2
n+3	tavaline 4	tavalise LFN 2 - osa 1
n+4	tavaline 5	tavaline 2
n+5	tavaline 6	tavaline 3
...	...	...

LFN kirjetel on atribuutidena kasutusel köite nimi, varjatud, süsteemne ja kirjutuskaitse. Seetõttu paljud programmid ei oska neid näidata, kuna kasutusel on köite nimi.

tabel 10 [12]

Nihe	Pikkus	Kirjeldus
00h	1 bait	tavaväli (ordinary field)
01h	2 baiti	Unicode sümbol 1
03h	2 baiti	Unicode sümbol 2
05h	2 baiti	Unicode sümbol 3
07h	2 baiti	Unicode sümbol 4
09h	2 baiti	Unicode sümbol 5
0Bh	1 bait	lipp
0Ch	1 bait	reserveeritud – alati 00h
0Dh	1 bait	kontrollsumma
0Eh	2 baiti	Unicode sümbol 6
10h	2 baiti	Unicode sümbol 7
12h	2 baiti	Unicode sümbol 8
14h	2 baiti	Unicode sümbol 9
16h	2 baiti	Unicode sümbol 10
18h	2 baiti	Unicode sümbol 11
1Ah	2 baiti	alati 0000h
1Ch	2 baiti	Unicode sümbol 12
1Eh	2 baiti	Unicode sümbol 13

Tavaväli on kasutusel selleks, et süsteem teaks, millise LFN kirjega on parajasti tegemist.

tabel 11 [12]

7	6	5	4	3	2	1	0	
kustutatud LFN	viimane LFN	LFN number						0000h

Kontrollsumma arvutamine. Kontrollsumma on vajalik selleks, et saaks kontrollida kas LFN kirje viitab õigele failile. Arvutuste aluseks on 8.3 formaadis failinimi. Kui failinimi on täidetud väärtustega 20h, siis kontrollsumma arvutamiseks kasutatakse ka neid väärtusi. Arvutuse algoritm on järgmine:

1. esimese sümboli ASCII väärtus on põhisummaks
2. pööratakse summa bitikaupa paremale
3. lisatakse järgmise sümboli ASCII väärtus summale
4. läbitakse veelkord etapid 2 ja 3 kuni kõik 11 sümbolit on lisatud.

2.4. FAT32

FAT32 on FAT-i 32–bitine versioon. FAT32 kasutab 32–bitist numbrit klastrite identifitseerimiseks, kuid reaalselt on nendest kasutusel 28, neli ülemist bitti on jäetud reservi. Maksimaalne võimalik klastrite arv, mida see süsteem lubab kasutada on 268 435 456. FAT32–s on alglaadesektorit täiendatud paari FAT32 vajaliku tükiga.

tabel 12 [12]

Osa	Nihe	Pikkus	Kirjeldus
kood	0000h	3 baiti	Koodi tükk, mis suunab alglaadekoodi juurde
OS nimi	0003h	8 baiti	Oem ID – vormindatud OS nimi
BIOS Parameetrite Plokk	000Bh	2 baiti	Baiti sektori kohta (*Bytes per Sector*)
	000Dh	1 bait	Sektorit klastri kohta (*Sectors per Cluster*). Tavaliselt on sektor 512 baiti
	000Eh	2 baiti	Reserveeritud sektorid alates ketta algusest
	0010h	1 bait	FAT-i koopiate arv. Tavaliselt on väärtuseks kaks
	0011h	4 baiti	Pole FAT32 kasutusel
	0015h	1 bait	Media kirjeldaja –Sama, mis FAT16, aga FAT32 kasutatakse ainult kõvaketastel, seetõttu kasutatakse väärtust F8h
	0016h	2 baiti	Pole FAT32-s kasutusel
	0018h	2 baiti	Sektoreid raja kohta (*Sectors per Track*)
	001Ah	2 baiti	Lugemispeade arv (*Number of Heads*)
	001Ch	4 baiti	Varjatud sektorid (*Hidden Sectors*)

	0020h	4 baiti	Sektorite arv ketta sektsiooni kohta
	0024h	4 baiti	Sektoreid FAT kohta
	0028h	2 baiti	näitab, kuidas FAT-e koheldakse (*FAT handling flag*)
	002Ah	2 baiti	FAT32 ketta versioon (vanem bait = põhiversioon, noorem bait= vähemtähtis versioon)
	002Ch	4 baiti	Klastri number, mida arvestatakse juurkataloogist
	0030h	2 baiti	Sektorite arv alates ketta sektsiooni algusest, mida kasutatakse failisüsteemi informatsiooni sektori jaoks (*File System Information Sector*)
	0032h	2 baiti	Sektorite arv alates ketta sektsiooni algusest, mida kasutatakse varundamise alglaadesektori jaoks (*Backup Boot Sector*)
	0034h	12 baiti	reserveeritud
Laiendatud BIOS Parameetrite Plokk	0040h	1 bait	Loogilised ketta numbrid (*Logical Drive Number*) – flopi ketastel 00h ja kõvaketastel 80h.
	0041h	1 bait	Windows NT kontroll bait, kas alglaadimise korral on vaja kõvaketast kontrollida
	0042h	1 bait	Signatuur
	0043h	4 baiti	ID – juhuslikult genereeritud seeria number
	0047h	11 baiti	Köite tähis – Sama, mis salvestatud juurkataloogi
	0052h	8 baiti	Süsteemi ID – näitab, et tegemist on FAT32–ga
kood	005Ah	420 baiti	alglaadurikood
Signatuur	01FEh	2 baiti	Käivitatava sektori signatuur – AA55h

FAT käsitsemise lipp: Kasutatakse peegeldamise (mirroring) jaoks. Kui lipp M on seatud, siis mingi muudatuse korral uuendatakse kõiki FAT-e, aga kui M-i ei ole seatud, siis uuendatakse ainult kasutatavaid välju.

tabel 13 [12]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
pole kasutusel								M	pole kasutusel		aktiivne FATi osa					0000h

FATi väärtused: Välja arvatud see, et FAT kasutab 32–bitist numbrit klastrite määramisel, on kõik muu sarnane eelmistele FAT-idele.

tabel 14 [12]

Väärtus	Kirjeldus
00000000h	vaba klaster
00000000h – FFFFFFF5h	võimalik järgmise klastri number
FFFFFFF6h – FFFFFFF7h	üks või mitu rikutud sektorit klastris
FFFFFFFFh	faili lõputunnus

3. NEW TECHNOLOGY FILE SYSTEM

Vormindamist NTFS-i keskkonnas võib vaadata läbi kahe mõiste – partitsioneerimine ehk kettaseadme tükeldamine ja formateerimine ehk failisüsteemi loomine kettaseadmele. Kui ei ole vaja täita erinõudeid, pole tihtipeale vajadust rohkema kui ühe partitsiooni järele. Erinõuded võivad olla tingitud väga paljudest asjaoludest, näiteks kasutaja andmete hoidmine eraldi süsteemile tähtsatest andmetest. Mitut partitsiooni võib minna vaja ka juhul, kui on soovi arvutis kasutada rohkem kui ühte opsüsteemi. NTFS-i failisüsteemi kasutades võib maksimaalne partitsiooni suurus ulatuda kuni 2^{64} baidini (16 eksabaiti) . Kahtlane on, et kellelgi tuleb nii suurte kettaseadmetega lähiajal tegemist, aga enne ketta jaotamist on vaja teada, milliseid piiranguid failisüsteem võib seada. Kettaseadme pind jaotatakse omakorda väiksemateks osadeks – klastriteks. Klaster on väikseim suurusühik, mida kasutatakse andmete salvestamiseks. NTFS-is kasutatakse klastrisuurusi alates 512 B, üks sektor ja kuni 64 kB. Kõige optimaalsem valik on 4 kB suurune klaster, sellisel juhul tundub tasakaal väikeste ja suurte failide vahel balansis olevat. Kuid see, kui suurt klastrit on vaja, sõltub põhiliselt kettaseadme kasutusest.

3.1. Master File Table

Kettaseade jaotatakse NTFSis kahte ossa – Master File Table (MFT) piirkond (~12%) ja andmete piirkond (~88%). MFT piirkonda luuakse MFT tabel. MFT on oma olemuselt spetsiaalne fail. See fail sisaldab 1 KB suuruseid kirjeid ning iga kirje identifitseerib üheselt ühe kettaseadmele salvestatud faili. MFT tabelis 16. esimest faili nimetatakse metafailideks. Metafailide asukohaks on juurkataloog ja nende tunnuseks on $-sümbol failinime ees.

3.1.1. MFT struktuur:

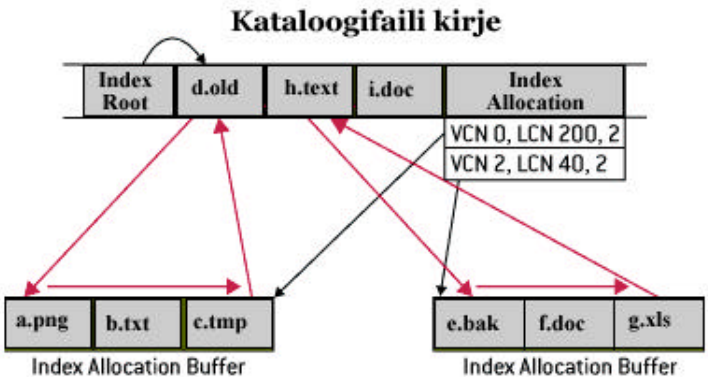
$MFT – Esimene kirje MFT-s ehk esimene kirjeldatav fail on MFT ise. See fail viitab MFT tabelile.
$MFTmirr – MFT sisaldab failisüsteemile kriitilist informatsiooni. Kui kettaseadme selle osaga midagi juhtub, peab olema võimalus seda kiiresti taastada. Sellise juhtumi puhuks on olemas teine, esimest 16 MFT kirjet kopeeriv fail. Seda faili nimetatakse MFT mirror failiks ning see fail asub täpselt kettaseadme keskel.

$LogFile – kasutatakse logiraamatuna (*journalising*). NTFS kasutab transaktsioonide kontseptsiooni tegevuste määratlemisel – kas tegevus viidi edukalt lõpuni või seda pole toimunudki. See metafail peab log-i pooleliolevate tegevuste kohta. Iga tegevus salvestatakse. Kui midagi vahepeal juhtub, siis süsteem vaatab, millised tegevused lõpetamata jäid ja enne kui ülejäänud tööd jätkatakse, taastab tegevuse eelse olukorra, mis stabiliseerib süsteemi. Selline võte on väga veakindel, see tähendab, et süsteemi ei käivitata kunagi ebastabiilses olukorras.
$Volume – kõikvõimalik informatsioon olemasoleva kettaseadme kohta.
$AttrDef – nimekiri standardfailide atribuutidest.
$ – juurkataloog
$Bitmap – kettaseadme vaba pinna kujutis.
$Boot – laadesektor. See metafail on olemas ainult laetavas partitsioonis.
$BadClus – vigased klastrid kettaseadmel.
$Quota – kasutaja õigused kettaseadme kasutamisel.
$UppCase – Unicode sümbolite tabel.
MFT piirkond on süsteemile peaaegu et ligipääsematu. Ainult erandjuhtumil võib süsteem saada luba sinna tavaandmeid kirjutada. See võib juhtuda sellisel puhul, kui andmete osa on täis saanud; siis vähendatakse MFT piirkonda ja kirjutatakse ka sinna andmeid. See, kui suure osa võrra seda vähendatakse, oleneb süsteemist. Kui olukord andmeosal hakkab stabiliseeruma, siis taastatakse endine olukord.

3.1.2. Fail ja kataloogid

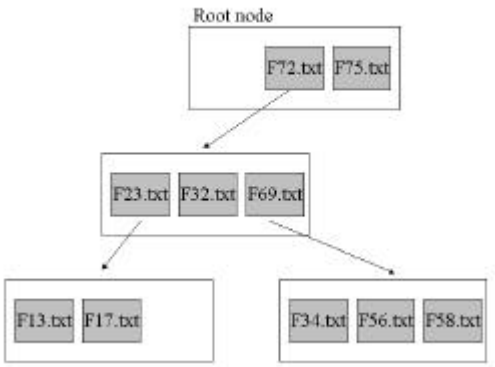
NTFS-i jaoks on kõik süsteemi komponendid failid. Iga fail on identifitseeritud kirjena MFT tabelis. Seetõttu, kui MFT tabelis on kirje faili kohta, siis see fail eksisteerib. Ülejäänud info, mis käib faili kohta, on mittekohustuslik.
Kataloogifail NTFS-is on eriline fail, milles on salvestatud viited failidele ja alamkataloogidele. Niimoodi luuakse hierarhiline struktuur. Kataloogifail on jagatud plokkideks, millest igaüks sisaldab failinime, faili põhiatribuute ja viiteid MFT tabelisse. Kirjed kataloogifailis sorteeritud tähestiku järjekorras ja hoitakse B+ puu kujul. B+ puu on üks kahendpuu vormidest – igas puu sõlmpunktis hoitakse mitmeid kirjeid. Kuna on alust arvata, et kõik kirjed kataloogifaili ära ei mahu, siis paigutatakse kirjed indeksite puhvrisse (*index allocation buffer*). Kataloogifaili jääb märge selle kohta, kus neid kirjeid tuleb otsida.

B+ puu hoitakse tasakaalus. Kataloogifailis hoitakse faile, mille järgi süsteem saab otsida kirjeid, mis kataloogi ära ei mahu. Olgu näiteks on kataloogikirjes üheksa failikirjet. Need jaotatakse ära järgnevalt:
kolm neist jäetakse kataloogikirjesse ja ülejäänud kuus kahte indeksite puhvrisse. Jagatakse sorteeritud järjekorra alusel, et võetakse järjest neli kirjet, millest viimane jäetakse kataloogikirjesse ja ülejäänud paigutatakse indeksite puhvrisse. Indeksite paigutuse kirjesse (*index allocation*) pannakse kirja, kus kohas indeksite puhver asub. Kui hakatakse mingit faili kataloogist otsima, siis kõigepealt käiakse kataloogikirje läbi ja võrreldakse failinimesid. Oletame, et otsitakse faili nimega e.bak. Sellisel juhul hakatakse failinime võrdlema esimese kirjega d.old. Võrdlemisel selgub, et e.bak peab asuma tagapool seda kirjet. Minnakse järgmise kirje juurde ja võrdlemisel selgub, et fail peab asuma nende kahe kirje vahel. Nüüd päritakse indeksite paigutuse kirjest infot. Kirjete siseselt kasutatakse klastrite määramiseks virtuaalsete klastrite numbreid (*virtual cluster number – VCN*). Küsitakse infot VCN 2 (fail h.text) kohta ja saadakse teada indeksite puhvri klastri number. Sealt leitakse otsitav fail.



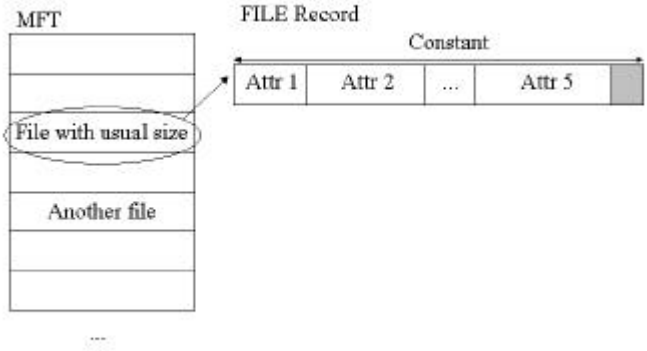
Joonis 2 [11]

Selline käsitlus on palju kiirem ja optimiseeritum kui näiteks failidepaigutustabeli puhul (FAT). FAT-is on failid reastatud loomise järjekorras. Sellisel juhul peab kõik failid järjest läbi uurima.



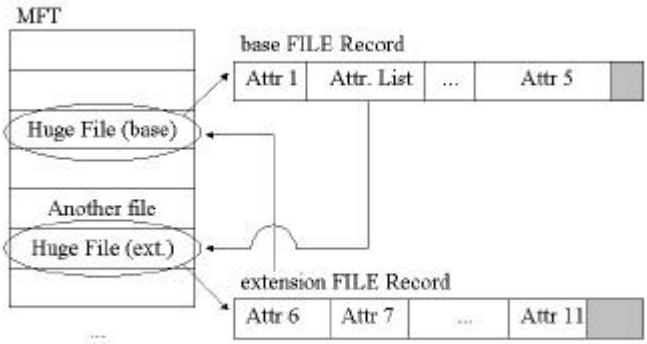
Joonis 3 [8]

Failide kandmisel MFT kirjesse pannakse sinna ka kirja faili atribuudid ja nimelised vood, ligipääsuõigused.

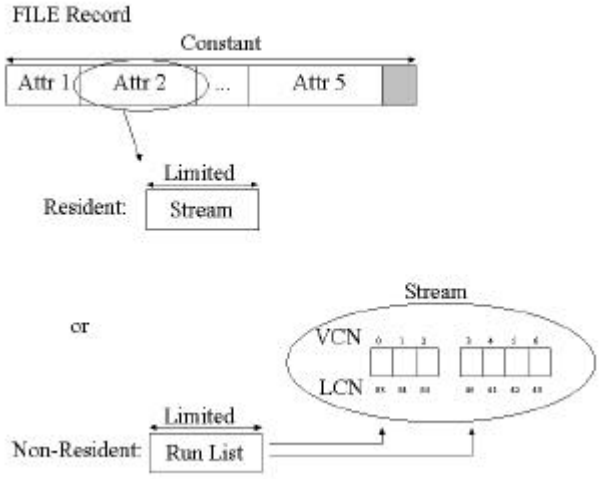


Joonis 4 [8]

Aga sellisel juhul on võimalik, et informatsioon ei mahu kõik sellesse kirjesse, siis võetakse kasutusele veel kirjeid, et see informatsioon ära paigutada. Sellisel juhul on esmases kirjes juures üks lisaatribuutide nimekiri, et oleks võimalik järgnevad kirjed üles leida. Ka on faili juures ära märgitud, kas fail on alaline (*resident*) või mittealaline (*nonresident*)



Joonis 5 [8]



Joonis 6 [8]

Võib ette tulla ka vastupidine situatsioon, et fail on väga väike ja samuti informatsiooni on sellel failil vähe, siis on suur võimalus, et terve see väike fail mahub MFT kirjesse. See kiirendab tublisti süsteemi tööd, sest failiga tegelemisel selle faili MFT kirje loetakse mällu.

3.1.3. Faili atribuudid:

Standardinformatsioon – tavalised failiatribuudid, nagu *read-only*, *hidden*, *system*, ajatempel ja lingi luger;

Nimi – faili või kataloogi nimi Unicode-is. Failil võib olla mitmeid nime attribuute sellisel juhul, kui on olemas 8.3 failinime kuju või kui failil on linke;

Security descriptor – andmestruktuur, mis on kirjeldab kasutajate õigusi
Andmed – faili sisu. Täpsemalt on tegu nimetu andmevooga, mida NTFS käsitleb tavalise failiatribuudina. Kataloogifailidel see atribuut puudub;
Nimeline andmevoog – mittekohustuslik faili atribuut, mis identifitseerib teised andmevood. Sellised vood võivad olla suunatud ka kataloogifailidesse;
Index Root, index allocation, bitmap – kasutusel suurte kataloogofailide puhul failinimede indekseerimisel;

3.1.4. Streams

Faili NT failisüsteemis võib tinglikult vaadata kui kanalite süsteemi. Vaikimisi on failil ainult üks kanal. Neid võib juurde teha, kuigi ilma spetsiaalsete programmideta võib see olla keerukas. Igas sellises kanalis talletatakse andmeid ning selliseid kanaleid nimetatakse voogudeks (*streams*). Uue faili loomisel võetakse kasutusele üks kanal – nimetu voog (*unnamed stream*). Nimetu voog sisaldab andmeid, millest saab aru enamus programmidest ning soovi korral on neid võimalus transportida teistesse failisüsteemidesse. Ülejäänud voogude loomine ja kasutamine on konkreetsete programmide ülesanne. Need vood on kindlate nimedega, et rakendusprogrammid neid ära tunneksid. Kuna enamus programme neid kasutada ei oska, on nimega vood (*named streams*) väljaspool NT failisüsteemi kasutud. Kui transportida faili, mis omab peale nimetu voo veel teisigi voogusid, siis ühest süsteemist üleminekul teise jäetakse alles info ainult nimetu voo kohta.
Näiteks oletame, et mingi firma X tegeleb pilditöötlusprogrammide tootmisega. Neil on vaja programmile lisada võimalus, et koos pildiga salvestatakse ka väiksemas formaadis vaade (*thumbnail view*). Sellise olukorra annaks lahendada voogude abil. Võetakse kasutusele kaks kanalit. Nimetu voog sisaldab endas pilti, aga teine voog thumbnail view. Info failis olevate voogude kohta salvestatakse faili päises.
Nimega vooge on võimalik kasutada ka kataloogide juures. Nt. desktop.ini.

3.1.5. Pakitud vood

Pakkimine (*compression*) on NTFS-i osa olnud juba algusest peale. Kui pakkimisel poleks omi nõrku külgi, siis võiks NT-failisüsteemis kõik andmed eksisteerida pakitud kujul. Nõrgaks küljeks ja suureks miinuseks on see, et selliste algoritmide täitmised kasutavad palju protsessori aega ning seda lennult ja kõikide andmetega ette võtta on liiga aja kulukas. Enne

pakkimist jaotatakse voog loogilisteks osadeks. Selliste osade pikkuseks on 16 klastrit (suuruseks 64 KB, juhul kui klastri suuruseks on 4 KB). Iga tükk loetakse eraldi mällu, kasutatakse pakkimisalgoritme ja saadud tulemus kirjutatakse tagasi kettaseadmele. Kui pakkimine õnnestus ja tagasikirjutatav osa hõlmab vähem kettapinda, siis tagastatakse vabaks jäänud klastrid süsteemile. Võib juhtuda, et ühte osa polnudki võimalik pakkida. Sellisel juhul kirjutatakse see osa samamoodi kettaseadmele tagasi. Kui fail oli sellise pikkusega, et jaotatud tükkidest viimane ei andnud 16 klastri mõõtu, siis seda osa mällu ei loeta ja pakkimist selle osa puhul ette ei võeta.

3.1.6. Krüpteeritud vood

Krüpteerimine (*encryption*) võimaldab oma andmeid kaitsta teiste kasutajate eest. See ei kaitse faile, aga see kaitseb sisu. Krüpteeritud faile on võimalik avada, aga midagi arusaadavat ei ole võimalik sealt välja lugeda. NTFS kasutab võtmepaaride loomiseks CryptoAPI-t. Võtmeid säilitatakse arvutis, aga neid ei kirjutata füüsiliselt kettaseadmele, seetõttu ei ole võimalik neid kettalt varastamise eesmärgil välja lugeda. Loomulikult võib kasutada spetsiaalseid kiipkaarte krüpteerimisvõtmete hoidmiseks. Paraku on juba selliste asjade puhul kombeks, et ei olda hoidmisel ettevaatlikud ja seetõttu võib juhtuda, et salajane võti kaob. Selliseks juhuks on välja mõeldud tagavara võtmesüsteem. NTFS ei luba kasutada krüpteerimist, kui taoline süsteem puudub. Tagavaravõtmed genereeritakse automaatselt. Domeenis defineeritakse need võtmed domeeni kontrollerisse ja nendele võtmetele pääseb ligi ainult administraatori õigustega isik.

3.1.7. Hõredad vood

NTFS toetab hõredate voogude (*sparse streams*) kasutamist failide juures. Faili kirjutamine toimub jadana. Kui tahaks faili kirjutada 50. kirjet, tuleb enne seda faili kirjutada kirjed 1. kuni 49. Hõredate voogude juures oleks võimalik see, et kirjutatakse ainult see kirje, mida on vaja kirjutada. Kui vaja faili kirjutada ainult 20005. kirje, siis ainult see kirjutataksegi ja muud kirjed võivad jääda puutumata.

3.1.8. Kõva link

Kõva lingi korral võib ühel failil olla mitmeid failiteid. Selliste failide korral viidatakse ühele ja samale MFT kirjele, seetõttu jagavad kõik need failid ühtesid ja samu failiatribuute

(ajatempel, turvalisus, vood). Piiranguks on, et sellised lingid kehtivad failidele. Kataloogifailidele linke teha ei saa. Sellised lingid toimivad ainult ühe kettaseadme piires. MFT kirjes on spetsiaalne luger, mida suurendatakse, kui tekitatakse uus link ja vähendatakse kui link eemaldatakse. Kui luger saab väärtuseks nulli, siis kirje kustutatakse MFT tabelist.

3.1.9. Kvoot

Kvoodi tugi NTFS-is aitab administraatoril jälgida ja piirata kasutajatele määratud kettaseadmel salvestatava pinna suurust. Kvootide toetuse tõttu on administraatoril parem ülevaade kettaseadme kasutatavusest. Administraator võib lihtsalt jälgida kasutajate kvoote, aga ta võib ka neid piirata. Kui nüüd kasutaja püüab ületada talle määratud pinda, siis süsteem teatab, et ketas on täis.
Kvootide puhul arvestatakse faili loogilist suurust. Kui faili tegelik suurus on 10 MB, aga ta on kokku pakitud 8 MB peale, siis loetakse kvoodi poolt faili suuruseks 10 MB. Kui faili suurus on 0 B, aga sinna on suunatud voog, mille suurus on 10 MB, siis kvoot arvestab faili suuruseks 10 MB. Selline suuruse arvestus on loodud selleks, et saaks võrrelda erinevate kettaseadmete kvoote võrdsetel alustel.

4. SECOND EXTENDED FILE SYSTEM

ext2 FS on praegu enam kasutatavamaid failisüsteeme Linux keskkonnas. ext2 FS, nagu sellele eelnenud ext FS on välja töötatud Remy Card'i, Theodore Ts'o ja Stephen Tweedie poolt. See failisüsteem toetab standardseid UNIX-i failitüüpe: tavafailid, kataloogid, seadmefailid ja sümboolsed lingid. Ext2 saab hakkama üsna suure mahuliste kettaseadmetega. Kui tuuma koodi tõttu oli algselt piiranguks 2 GB, siis nüüd on piire nihutatud 4 TB. Ext2 toetab ka pikki failinimesid. Kasutades muutuva pikkusega kataloogikirjeid, võib failinimi maksimaalselt olla 255 sümboli pikkune. Vajaduse korral, kui sellest ei piisa, on võimalik failinime piiriks määrata 1012 sümbolit.

4.1. Plokid ja fragmendid

Kettapind jagatakse kindla suurusega plokkideks. Kasutuses on kahte sorti plokid – füüsilised ja loogilised plokid. Füüsiline plokk asub füüsiliselt sellel salvestataval meedial ja on alati kindla suurusega. Loogiline plokk luuakse failisüsteemi loomise ajal. Suurus on valitav, kas 1024, 2048 või 4096. Loogiline plokk jaotatakse veel väiksemateks osadeks. Selliseid osasid nimetatakse fragmentideks. Kuna fail ei pruugi olla fikseeritud suurusega, siis jääb üldjuhul mingi osa plokist kasutamata. Fragmentide kasutamine annab võimaluse vähendada ketta pinna raiskamist. Kui mingi plokk täidetakse osaliselt, siis jagatakse see plokk fragmentideks. Vabad fragmendid eemaldatakse sellest plokist ja liidetakse teiste vabade fragmentidega uuteks plokkideks.

Plokkide arvestamine algab 0 plokist, mis on alglaadimisplokk. Failisüsteemi loomise hetkel reserveeritakse mingi osa plokkidest *superuser*-i käsutusse. See informatsioon salvestatakse struktuuri, mis kuulub superploki juurde – *s_r_blocks_count*. Tavapäraselt on selleks reserveeritud mahuks 5% plokkidest. Kui ketta vaba mahtu on järgi reserveeritud plokkide jagu, siis ei lubata tavakasutajal enam midagi kettale salvestada ja ainult *superuser* saab uusi plokke kasutusele võtta. Ilma sellise ettevaatusabinõuta oleks võimalik tekitada olukord, kus süsteemi ei suudaetaks enam käivitada. Reserveeritud plokkide puhul on garanteeritud vähemalt mingi vaba ruum, mida saaks alglaadimiseks kasutada ja mis annaks võimaluse *superuser*-il kettalt mittevajaliku osa eemaldada.

4.2. Grupid

Plokid ühendatakse kokku ühtseteks gruppideks. Iga selline grupp omab failisüsteemile vajalikku informatsiooni. Sellisel kujul info kordamisega tehakse süsteem veakindlamaks.

tabel 15 [6]

Boot Sector	Block Group 1	Block Group 2	... ...	Block Group N

Grupp koosneb järgmistest osadest:

superplokk (*Super Block*);

gruppi kirjeldav andmestruktuur (*FS descriptors*);

gruppi kuuluvate plokkide *bitmap* (*Block Bitmap*);

grupi *inode*-ide bitmap (*Inode Bitmap*);

grupi *inode*-ide tabel (*Inode Table*);

gruppi kuuluvad andmeplokid (*Data Blocks*);

tabel 16 [6]

Super Block	FS descriptors	Block Bitmap	Inode Bitmap	Inode Table	Data Blocks

4.3. Superplokk

Superplokk sisaldab failisüsteemile olulist informatsiooni. Seetõttu on sellele ligipääs lubatud ainult *superuser*-ile. Superplokk on järgmise struktuuriga:

```
struct ext2_super_block {
 unsigned long s_inodes_count;        //failisüsteemis esinevate inode-ide
                                      arv
 unsigned long s_blocks_count;        //failisüsteemis olevate plokkide arv
 unsigned long s_r_blocks_count;      //plokid, mis on reserveeritud
                                      superuser-ile
 unsigned long s_free_blocks_count;   //vabade plokkide arv
 unsigned long s_free_inodes_count;   //vabade inode-ide arv
 unsigned long s_first_data_block;    //esimese andmeploki asukoht
 unsigned long s_log_block_size;      //arvutatakse loogilise ploki suurus
                                      baitides
 long          s_log_frag_size;       //arvutatakse loogilise fragmendi
                                      suurus baitides
```

```
 unsigned long s_blocks_per_group;    //grupis sisalduvate plokkide arv
 unsigned long s_frags_per_group;     //grupis sisalduvate fragmentide arv
 unsigned long s_inodes_per_group;    //grupis sisalduvate inode-ide arv
 unsigned long s_mtime;               //aeg, millal failisüsteemi
                                      ühendamine viimati toimus
 unsigned long s_wtime;               //aeg, millal superplokki
                                      failisüsteemi poolt viimati kirjutati
 unsigned short s_mnt_count;          //mitu korda on failisüsteem
                                      ühendatud ilma vigade kontrollita
 short          s_max_mnt_count;      //arv, mitu korda võib failisüsteemi
                                      ühendada ilma, et peaks tegema vigade
                                      kontrolli
 unsigned short s_magic;              //maagiline number, mis lubab
                                      failisüsteemi identifitseerimist
 unsigned short s_state;              //failisüsteemi seisund. Sellel on
                                      kaks võimalikku väärtust:
                                      EXT2_VALID_FS (0x0001) – viimane
                                      failisüsteemi ühendamine toimus
                                      vigadeta või EXT2_ERROR_FS (0x0002) –
                                      tuuma poolt on avastatud vigu
 unsigned short s_errors;             //osutab, millist operatsiooni läbi
                                      viia juhul, kui avastatakse vigu
 unsigned short s_pad;                //pole kasutusel
 unsigned long s_lastcheck;           //aeg, millal viimati failisüsteemi
                                      kontrolliti
 unsigned long s_checkinterval;       //maksimaalne võimalik aeg, mis võib
                                      olla failisüsteemi kahe kontrollimise
                                      vahel
 unsigned long s_reserved;            //pole kasutusel
};
```

Aja arvestamine toimub sekundites alates 00:00:00 GMT, January 1, 1970 aastast.

Superplokk loetakse mällu. ext2 tuuma kood arvutab selle põhjal täiendatavat informatsiooni ja hoiab seda struktuuris *ext2_sb_info*:

```
struct ext2_sb_info {
 unsigned long s_frag_size;           //fragmendi suurus baitides
 unsigned long s_frags_per_block;     //mitu fragmenti on plokis
 unsigned long s_inodes_per_block;    //inode-ide arv inode-ide tabeli
                                      plokis
 unsigned long s_frags_per_group;     //fragmentide arv grupis
```

```
 unsigned long s_blocks_per_group;    //plokkide arv grupis
 unsigned long s_inodes_per_group;    //inode-ide arv grupis
 unsigned long s_itb_per_group;       //inode-ide tabeli plokkide arv
                                      grupis
 unsigned long s_desc_per_block;      //gruppi kirjeldavate
                                      andmestruktuuride arv ploki kohta
 unsigned long s_groups_count;        //gruppide arv
 struct buffer_head * s_sbh;          //superplokki mälus sisaldav puhver
 struct ext2_super_block * s_es;      //viit superplokile puhvris
 struct buffer_head * s_group_desc[EXT2_MAX_GROUP_DESC];    //viidad gruppi
                                      kirjeldavaid andmestruktuure
                                      sisaldavatele puhvritele
 unsigned short s_loaded_inode_bitmaps;   //kasutusel olevate inode-ide
                                      bitmap-i vahemälu kirjete arv
 unsigned short s_loaded_block_bitmaps;   //kasutusel olevate plokkide
                                      bitmap-i vahemälu kirjete arv
 unsigned long s_inode_bitmap_number[EXT2_MAX_GROUP_LOADED];      //osutab,
                                      millisesse gruppi inode-i bitmap
                                      puhvris kuulub
 struct buffer_head * s_inode_bitmap[EXT2_MAX_GROUP_LOADED];      //inode-i
                                      bitmap-i vahemälu
 unsigned long s_block_bitmap_number[EXT2_MAX_GROUP_LOADED];      //osutab,
                                      millisesse gruppi ploki bitmap
                                      puhvris kuulub
 struct buffer_head * s_block_bitmap[EXT2_MAX_GROUP_LOADED];      //ploki
                                      bitmap-i vahemälu
 int s_rename_lock;                   //lukustus, kasutatakse vältimaks
                                      failisüsteemis kaht samaaegset
                                      ümbernimetamise operatsiooni
 struct wait_queue * s_rename_wait;   //ootejärjekord ümbernimetamiste
                                      operatsiooni täitmise ootel
 unsigned long s_mount_opt;           //administraatori poolt määratud
                                      ühendamise valikud
 unsigned short s_mount_state;        //enamus nendest väärtustest on
                                      arvutatud superploki poolt
};
```

Linuxi ext2fs haldavad programmid kasutavad vahemälu (*cache*) *inode*-ide ja plokkide *bitmap*-idele ligipääsuks. Seda sorti vahemälu on puhvrite register, kus järjestatud viimati

kasutatud puhvrid. ext2 haldavad programmid peaksid kasutama samu või sarnaseid meetodeid *bitmap*-ide vahemällu salvestamiseks, et kiirendada ketta poole pöördumise aega.

4.4. Gruppi kirjeldav andmestruktuur

Gruppi kirjeldav andmestruktuur (*group descriptor*) järgneb koheselt superplokile ja kõikidel andmestruktuuridel on järgnev sisu:

```
struct ext2_group_desc
{
 unsigned long bg_block_bitmap;       //viidad grupi plokkide bitmap-i
                                      plokile
 unsigned long bg_inode_bitmap;       //viidad grupi inode-ide bitmap-i
                                      plokile
 unsigned long bg_inode_table;        //viidad inode-ide tabeli esimesele
                                      plokile
 unsigned short bg_free_blocks_count; //grupi vabade plokkide arv
 unsigned short bg_free_inodes_count; //grupi vabade inode-ide arv
 unsigned short bg_used_dirs_count;   //grupi kataloogidesse jaotatud
                                      inode-ide arv
 unsigned short bg_pad;               //padding ehk sisutu märgijada, mis
                                      lisatakse andmetele nõutava vormingu
                                      saavutamiseks
 unsigned long bg_reserved[3];        //reserveeritud
};
```

Gruppi kirjeldav andmestruktuur kirjeldab ainult seda konkreetset gruppi, milles see asub.

4.5. Bitmap

Bitmap ehk *bit pattern* – igale andmeelemendile vastab üks või mitu bitti mälus. ext2 failisüsteem kasutab *bitmap*-e jaotatud plokkide ja *inode*-ide jälgimiseks. Iga grupi ploki *bitmap* viitab plokkide piirkonnale alates grupi esimesest plokist kuni viimaseni. Kui on vajalik mingile kindlale plokile ligipääsemine, peab kõigepealt järgi uurima, millisesse gruppi see plokk kuulub ja siis uurima selle grupi plokkide *bitmap*-i. Tegelikult viitab ploki *bitmap* vähimale jaotuslikule osale, mis on failisüsteemi poolt toetatud, nimelt fragmendile. Plokk

koosneb alati mingist kindlast arvust fragmentidest, seetõttu tegeletakse failisüsteemi tasemel ploki jaotamise korral alati mingi arvu fragmentidega.

Gruppide *inode*-ide *bitmap* viitab *inode*-i piirkonnale alates grupi esimesest *inode*-ist kuni viimaseni. Kui on vajalik ligipääs kindlale *inode*-ile, siis samamoodi nagu plokkidegi puhul tuleb kõigepealt kindlaks teha, millisesse gruppi see *inode* kuulub ja seejärel uurida vastava grupi *inode*-i *bitmap*-i. Samasugune tegevusjärjekord tuleb läbi teha, kui on vajalik saada informatsiooni *inode*-i tabelist.

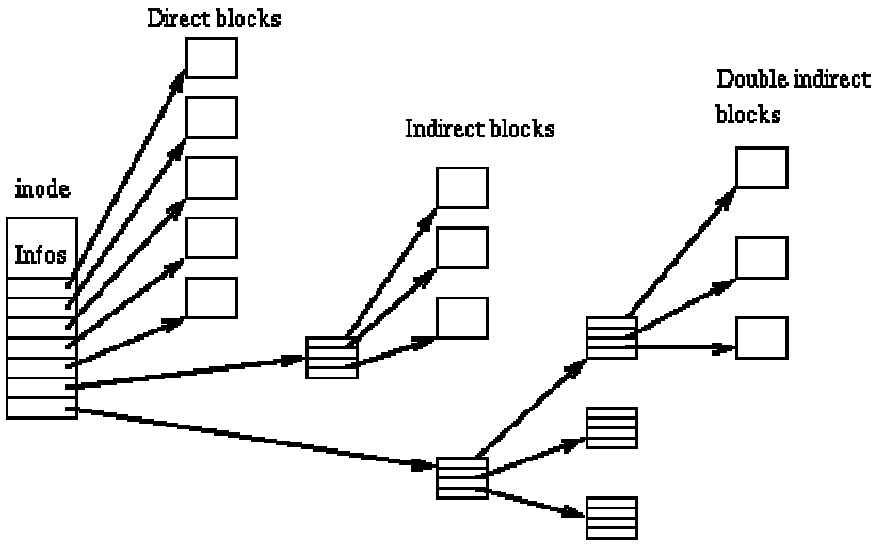
4.6. Inode

On erinevaid võimalusi, kuidas faile kirjeldada. Üks võimalus on failid indekseerida. *Inode*-i võiks defineerida kui indekseeritud sõlmpunkti. Igal failil on oma unikaalne *inode* ja sellel on järgmine struktuur:

```
struct ext2_inode {
 unsigned short i_mode;              //määratakse ära failitüüp (char-
                                     tüüpi, link) ja juurdepääsu õigused
 unsigned short i_uid;               //faili omaniku identifikaator
                                     (userID)
 unsigned long i_size;               //faili loogiline suurus baitides
 unsigned long i_atime;              //aeg, millal faili viimati kasutati
 unsigned long i_ctime;              //aeg, millal faili inode-i
                                     informatsiooni viimati muudeti
 unsigned long i_mtime;              //aeg, millal faili sisu viimati
                                     muudeti
 unsigned long i_dtime;              //aeg, millal see fail kustutati
 unsigned short i_gid;               //faili grupi identifikaator
                                     (groupID)
 unsigned short i_links_count;       //sellele failile viitavate linkide
                                     arv
 unsigned long i_blocks;             //selle faili poolt haaratud plokkide
                                     arv, arvutuslikuks ühikuks 512 baiti
 unsigned long i_flags;              //lipp, mis näitab, kas failiga on
                                     seotud mingeid lisaomadusi
 unsigned long i_reserved1;          //reserveeritud
 unsigned long i_block[EXT2_N_BLOCKS];    //viidad plokkidele
```

```
 unsigned long i_version;            //faili versioon, mida kasutab
                                     Network File System (NFS)
 unsigned long i_file_acl;           //faili ligipääsuloend – access
                                     control list (ACL). Pole veel
                                     kasutusel
 unsigned long i_dir_acl;            //kataloogi ACL
 unsigned long i_faddr;              //plokk, kus asub faili fragment
 unsigned char i_frag;               //fragmendi number plokis
 unsigned char i_fsize;              //fragmendi suurus
 unsigned short i_pad1;              //sisutu märgijada
 unsigned long i_reserved2[2];       //reserveeritud
};
```

Nagu näha sisaldab *inode* (EXT2_N_BLOCKS) viitasid plokkidele. Esimesed EST2_NDIR_BLOCKS (12) on otsesed viidad (*direct pointers*) andmetele. Järgmised kirjed ehk kaudsed viidad (*indirect points*) viitavad plokkidele, kus asuvad viidad andme plokkidele. Järgmised kirjed viitavad plokkidele, kus on viidad plokkidele, kus asuvad viidad andmeplokkidele (*double indirection*). Veel kasutatakse ka kolmekordseid viitasid (*triple indirection*).



Joonis 7

Nagu üleval pool juba märgitud sai, saab failile lisada mingeid omadusi.

`EXT2_SECRM_FL 0x0001` – *secure deletion* ehk turvaline kustutamine. Kui failil on see lipp seatud, siis peale faili kustutamist kirjutatakse faili poolt kasutatud plokkidesse suvaline andmejada

`EXT2_UNRM_FL 0x0002` –*undelete* ehk ennistatav fail. Selle lipu olemasolu korral faili kustutamise järel jäetakse teatud ulatuses failist piisavalt informatsiooni järgi, et seda oleks vajaduse korral võimalik taastada

`EXT2_COMPR_FL 0x0004` – *compress file* ehk pakitud fail. Faili hoitakse pakitud kujul. Kui selle failiga sooritatakse mingeid operatsioone, peab failisüsteem kasutama teatud pakkimis- ja lahtipakkimisalgoritme

`EXT2_SYNC_FL 0x0008` – *synchronous updates* ehk sünkroonne uuendamine. Faili kujutamine kettal peab olema sünkroniseeritud tema mälupildi (*core*) kujutisega. Asünkroonne I/O sellise faili puhul pole võimalik. Sünkroniseeritakse ainult *inode*-i ennast ja kaudseid (*indirect*) plokke. Andmeplokke kirjutatakse kettale ikkagi asünkroonselt.

Mõnedel *inode*-idel on oma kindel tähendus:

`EXT2_BAD_INO 1` – fail, mis sisaldab loendit failisüsteemis sisalduvate defektsete plokkide (*badblock*) kohta

`EXT2_ROOT_INO 2` – failisüsteemi juurkataloog

`EXT2_ACL_IDX_INO 3 –` ACL *inode*.

`EXT2_ACL_DATA_INO 4` – ACL *inode*.

`EXT2_BOOT_LOADER_INO 5` – fail, mis sisaldab alglaadurit. Ei ole hetkel kasutusel

`EXT2_UNDEL_DIR_INO 6` – ennistatav kataloog ehk kataloogi on võimalik taastada peale kustutamist

`EXT2_FIRST_INO 11` – *inode*, millel pole oma kindlat tähendust

4.7. Kataloogid

Kataloogid on spetsiaalsed failid. Nende kaudu kindlustatakse ligipääs failidele. Ei tohiks jätta tähelepanuta, et *inode*-il võib olla mitmeid ligipääsuteid. Kataloogid on failisüsteemile väga olulised, seetõttu on neil kindel struktuur. Kataloogifail omab järgnevas formaadis kirjete loendit:

```
struct ext2_dir_entry {
 unsigned long inode;              //viide faili inode-le
 unsigned short rec_len;           //kirje pikkus
```

```
 unsigned short name_len;           //failinime pikkus
 char      name[EXT2_NAME_LEN];     //failinimi
};
```

4.8. Paigutuste algoritmid

Ext2-le on ette nähtud kindlad paigutuslikud algoritmid (*allocation algorithms*). Tänapäeval kasutatakse ühe arvuti peal palju erinevaid opsüsteeme ja mitmed opsüsteemid kasutavad partitsioone, millel on failisüsteemiks ext2. Seetõttu peavad nad ka samu algoritme kasutama. Selleks on kindlaks määratud reeglid, mida tuleb kasutada. *Inode*-ide paigutamiseks ettenähtud reeglid:
uue faili *inode* tuleb paigutada samasse gruppi, kus tema vanem kataloogi *inode*;
inode-id tuleb paigutada gruppide vahel võrdselt

Uute plokkide paigutamiseks ettenähtud reeglid:
uus plokk paigutatakse samasse gruppi, kuhu kuulub ka tema *inode*
plokid tuleb paigutada järjestikustesse plokkidesse

Siiski võib ette tulla olukordi, kus nende reeglite järgimine on võimatu. Sellisel juhul peavad failisüsteemi haldavad programmid paigutama plokid ja *inode*-id kuhugi mujale.

4.9. Käitumine vigade korral

Superplokk sisaldab kahte parameetrit, mille läbi ta korraldab käitumist vigade korral (*error handling*). Esimene *s_mount_opt* kuulub superploki mälu struktuuri. Selle väärtus arvutatakse spetsiaalsete valikute järgi failisüsteemi ühendamise ajal:
`EXT2_MOUNT_ERRORS_CONT` – jätkatakse, isegi juhul, kui vigu tuvastatakse
`EXT2_MOUNT_ERRORS_RO` – failisüsteemi ümberühendamine olekus, kus andmeid saab ainult lugeda (*read-only*)
`EXT2_MOUNT_ERRORS_PANIC` – vea korral katkestab tuum tegevuse
Teine *s_errors* on superploki struktuuri osa, mis asub kettal. Väärtused sellel võivad olla järgmised:
`EXT2_ERRORS_CONTINUE` – vea tuvastamise järel jätkatakse tööd

`EXT2_ERRORS_RO` – failisüsteemi ümberühendamine olekus, kus andmeid saab ainult lugeda (*read-only*)

`EXT2_ERRORS_PANIC` – vea korral katkestab tuum tegevuse

`EXT2_ERRORS_DEFAULT` – kasutatakse vaikimisi käitumist

KOKKUVÕTE

Käesoleva proseminaritöö eesmärgiks oli selgitada tuntumaid failisüsteeme ja luua abimaterjal informaatika eriala üliõpilastele. Erialaseid materjale on raske leida ja tihtipeale on tegemist erineva ülesehitusega artiklitega. Seetõttu on hea seda kasutada lisamaterjalina. Kirjeldatud on failisüsteemide üldist struktuuri ja ülesehitust. Iga failisüsteemi juures on välja toodud just sellele süsteemile omasemad jooned.

Oma töös olen keskendunud failisüsteemide ülesehituse kirjeldamisele ja ei ole tegelenud analüüsiga. Need failisüsteemid on oma arhitektuurilt piisavalt erinevad, et saaks hakata konkreetseid paralleele tõmbama. Kui üldistada, siis NTFS ja ext2 on üsna sarnased selle poolest, et nad on skaleeritavad, see tähendab, et nende ülesehitus võimaldab neid kohaldada uutele oludele vähese vaevaga ja ilma suuremate struktuurimuutusteta. FAT seevastu on ajast maha jäänud ega vasta tänapäeva nõuetele.

Käesolev materjal on ka suhteliselt teoreetiline, mis on tingitud proseminaritöö mahust, seetõttu puuduvad praktilised näpunäited ja tulemuste analüüs.

Käesolev proseminaritöö on kättesaadav kõigile veebiaadressil:
http://www.tpu.ee/~kallet/proseminar/prosem.doc

KASUTATUD KIRJANDUS:

1. Septer, A., Liikane, L., Inglise-eesti-inglise seletav arvutisõnastik, Tallinn, Estada kirjastus OÜ, 2000, 479 lk.
2. Agur, U., Hanson, V., Kull, R., Inglise-vene-eesti arvutisõnastik. A – L, Tallinn : Eesti Majandusjuhtide Instituut, 1991, 192 lk.
3. Agur, U., Hanson, V., Kull, R., Inglise-vene-eesti arvutisõnastik. M – Z, Tallinn : Eesti Majandusjuhtide Instituut, 1991, 396 lk.
4. Norton, P., Mueller, J. P., Peter Norton's Complete Guide to Windows NT 4 Workstation, United States of America, Sams Publishing, 1996
5. Dubeau, Louis-Dominique, Analysis of the Ext2fs structure, http://step.polymtl.ca/~ldd/ext2fs/ext2fs_toc.html
6. Card, Rémy, Ts'o, Theodore, Tweedie, Stephen, Design and Implementation of the Second Extended Filesystem, http://web.mit.edu/tytso/www/linux/ext2intro.html
7. Hinner, Martin, Filesystems HOWTO, http://www.linuxdoc.org/HOWTO/Filesystems-HOWTO.html
8. Linux-NTFS Project, http://linux-ntfs.sourceforge.net
9. Richter, Jeffrey and Cabrera, Luis Felipe, A File System for the 21st Century: Previewing the Windows NT 5.0 File System, http://www.microsoft.com/msj/1198/ntfs/ntfs.htm
10. Mikhailov, Dmitrey, NTFS file system, http://www.digit-life.com/articles/ntfs/
11. Russinovich, Mark, Inside Win2K NTFS, Part 1, http://msdn.microsoft.com/library/en-us/dnw2kmag00/html/NTFSPart1.asp?frame=true
12. MAVERICK OS, http://www.maverick.subnet.dk/Mainmenu_NoFrames.html