INERTNA STRUKTURA UNIX ZDRUŽLJIVEGA 8-BITNEGA OPERACIJSKEGA SISTEMA

INFORMATICA 1/88

Ivan Pepelnjak
Jernej Virant
Marjan Bradeško
Nikolaj Zimic
Fakulteta za elektrotehniko, Ljubljana

UDK 681.3.06

Izvleček :

Razvoj mikroračunalniške tehnologije zahteva uporabo vse zmogljivejših diskovnih enot in tiskalnikov. Takšne zmogljivosti lahko učinkovito izrabimo le v računalniški mreži. Za uspešno implementacijo mreže mikroračunalnikov rabimo operacijski sistem, ki podpira poddirektorije, zaščito datotek ipd. V članku je predstavljen operacijski sistem, ki izpolnjuje vse navedene zahteve, poleg tega je UNIX in CP/M združljiv. Opisana je struktura datotek na disku in interna zgradba operacijskega sistema.

Abstract :

The development of computer technology requires use of more and more sophisticated disk units and printers. This resources can be effectively used only in a local area network. To successfully implement microcomputer local area network, an operating system supporting subdirectories, file protection and users separation is needed. In the paper we present an operating system which meets all of the stated requirements along with UNIX and CP/M compatibility. Disk structure and internal data structures used are presented.

1. UVOD

1.1 Opis problema

Vse hitrejši razvoj mikroračunalniške tehnologije omogoča prodor mikroračunalnikov v vsa področja družbenega življenja. Kakor hitro pa hočemo mikroračunalnik uporabiti v poslovni sferi, se srečamo s količinskimi problemi :

- prisotnost velikih baz podatkov
- velika količina izpisov

ki jih lahko rešimo samo z zahtevno in še vedno drago tehnologijo (trdi diski, hitri tiskalniki). Poleg tega morajo biti podatkovne baze v takšnih okoljih praviloma deljene med več uporabnikov, ki hkrati dosegajo podatke iz podatkovne baze.

Če hočemo rešiti takšen problem je najbolje, da povežemo več mikroračunalnikov v mrežo, znotraj katere si lahko delijo zmogljivosti (trdi disk) in izmenjujejo sporočila. Tako dosežemo večjo hitrost delovanja celotnega sistema (inteligenca je razdeljena med več delovnih mest) in manjšo ceno sistema na enoto zmogljivosti.

1.2 Naša realizacija cenene mikroračunalniške mreže

Mikroračunalniško mrežo lahko s stališča programske opreme realiziramo na več načinov :

- vsi računalniki lahko zahtevajo podatke z vseh računalnikov
- nekateri računalniki so delovne postaje, drugi so strežniki

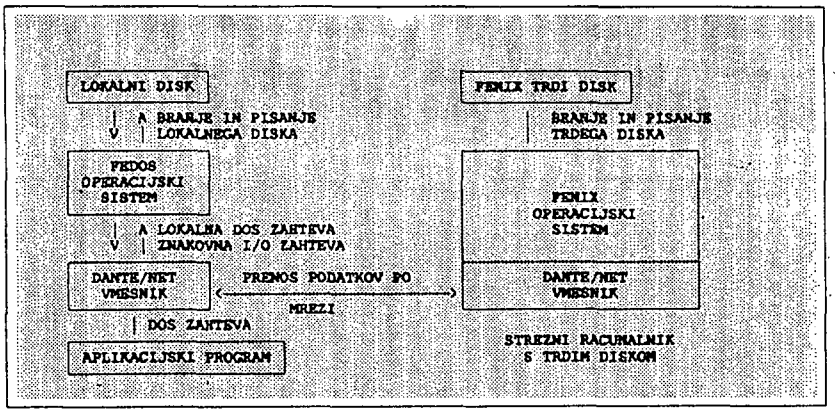
Prvi pristop je zanimiv v primeru, ko imamo na razpolago že obstoječ (dokaj drag) mikroračunalnik s trdim diskom. Ta implementacija ima nekatere slabe lastnosti, saj je operacijski sistem takšnega računalnika zahtevnejši, poleg tega zaradi servisiranja zahtev z drugih vozlišč upade hitrost lokalnega dela.

Drugi pristop je boljši v primeru, ko lahko sami razvijemo svoj mikroračunalnik, saj se lahko odločimo za posebno izpeljanko mikroračunalnika, ki vsebuje samo krmilnik za trdi disk in enostaven zaslonski krmilnik za diagnostiko ter tako dosežemo cenejšo izvedbo mikroračunalnika. Delitev vozlišč na strežna vozlišča in delovne postaje poleg tega prinese še dodatne prednosti, saj se vsako vozlišče ukvarja izključno z eno nalogo, ki jo zaradi tega opravlja bolje in hitreje.

Kot osnovo naše mikroračunalniške mreže smo vzeli mikroračunalnik DIALOG z operacijskim sistemom FEDOS (CP/M združljiv operacijski sistem) in mikroprocesorjem Z80.

Ko smo preudarjali, kakšen naj bi bil operacijski sistem strežnega računalnika smo ugotovili naslednje :

- operacijski sistem mora podpirati zaščito datotek med uporabniki
- operacijski sistem mora podpirati drevesno strukturo kazala datotek



Slika 1.1 Struktura mikroračunalniške mreže

V naslednjih razdelkih si bomo zaporedoma ogledali
- Strukturo diska, kot jo podpira FENIX operacijski sistem
- notranjo zgradbo FENIX operacijskega sistema

Če smo hoteli v okviru mreže podpirati standardno CP/M programsko opremo, smo morali vsaj logično obdržati CP/M diskovno strukturo in CP/M diskovne funkcije. Tako smo prišli do naslednje strukture mikroračunalniške mreže :

- v strežnem računalniku teče operacijski sistem FENIX, ki podpira večino funkcij, ki jih mora imeti večuporabniški operacijski sistem (zaščita datotek, drevesna struktura kazala, podpora večih uporabnikov ipd).
- v delovnih postajah teče lokalno operacijski sistem FEDOS (CP/M združljiv) in vmesnik na lokalno mrežo DANTE/NET, ki pošilja zahteve preko lokalne mreže strežnemu računalniku

2. STRUKTURA SISTEMA DATOTEK

2.1 Uvod

Operacijski sistem, ki podpira hkratno ali zaporedno delo večih uporabnikov, mora zadovoljevati sledeče zahteve :

- omogočati mora ločitev datotek različnih uporabnikov, da omogoči vsakemu uporabniku pregled nad njegovimi datotekami in prepreči nehoteno izgubo podatkov zaradi napake v delu enega od uporabnikov

- omogočati mora zaščito datotek pred drugimi uporabniki, da bi preprečili zlorabo podatkov ali namerno povzročeno izgubo podatkov.

Poleg teh zahtev mora operacijski sistem izpolnjevati še naslednje zahteve, ki jih narekuje optimalnost delovanja računalniškega sistema :

- omogočati mora obstoj zelo majhnih datotek, saj je v razvojnem okolju večina datotek majhna (100 bytov - 10K bytov). Osnovna enota dodeljevanja prostora na disku mora biti torej čim manjša, najbolje en fizični sektor.

- omogočati mora obstoj zelo velikih datotek (omejitev mora biti velikost diskovnega pomnilnika), saj podatkovne baze navadno potrebujejo izredno velike datoteke.

- v razvojnem okolju je ugodno, če lahko uporabnik svoje datoteke razdeli v logične skupine.

- omogočati mora hiter naključen dostop do datotek.

Praktično vsi operacijski sistemi imajo dostop do datotek organiziran preko kazala datotek. Le to vsebuje najmanj dva podatka :

- ime datoteke
- informacijo o tem, kje se datoteka nahaja na disku

Informacija o položaju datoteke na disku je lahko spravljena na tri različne načine :

- v kazalu datotek
- v posebnem delu diska
- deli datoteke so med seboj povezani v verigo

Če izberemo tretjo možnost imamo še vedno dve različici :

- informacija o nadaljevanju verige je skrita v vsakem sektorju.
- informacija o nadaljevanju verige je zapisana v posebni tabeli.

Vse tri rešitve imajo svoje dobre in slabe lastnosti :

2.1.1 Zapis položaja datoteke v kazalu datotek

Dobre lastnosti :

- položaj datoteke ugotovimo takoj, ko najdemo ustrezen zapis v kazalu datotek. Tako prihranimo en dostop do diska
- takšen način zapisa je najlažje realizirati

Slabe lastnosti :

- Velikost zapisa v kazalu datotek je velika, zato potrebujemo za iskanje zapisa v kazalu več dostopov do diska.
- Pojavi se problem velikih datotek, ki jih navadno rešujemo z uporabo več zapisov v kazalu datotek, kar še povečuje že tako veliko kazalo

2.1.2 Zapis položaja datoteke v posebnem delu diska

Dobre lastnosti :

- kazalo datotek je majhno, kar omogoča hitro iskanje

Slabe lastnosti :

- dostop do informacije o položaju datoteke zahteva še en dostop do diska in praviloma tudi premik glave.

2.1.3 Povezava v verigo

Dobre lastnosti :

- kazalo datotek je majhno
- v primeru uporabe posebne tabele so vse informacije o položaju datotek zbrane na enem mestu

Slabe lastnosti :

- shranjevanje kazalcev v podatkovnih blokih izredno upočasni naključni dostop do datoteke in ga v bistvu degenirira v sekvenčnega
- uporaba posebne tabele navadno povzroči probleme zaradi njene velikosti (za 40M bytni disk pri velikosti enote dodeljevanja 1K byt ta tabela velika 80K bytov).

2.2 Pregled obstoječih rešitev

Oglejmo si sedaj nekatere obstoječe operacijske sisteme, posebno tiste, ki so primerljivi z našim sistemom.

2.2.1 CP/M

Operacijski sistem CP/M ima vse informacije o datotekah zbrane v skupnem kazalu datotek. Informacija o položaju datoteke je zapisana kar v kazalu datotek, za velike datoteke uporablja CP/M več zapisov v kazalu datotek.

Osnovna enota dodeljevanja je odvisna od velikosti diska in se giblje od 1K byt do 8K bytov. Velikost kazala datotek je omejena, torej je omejeno tudi skupno število datotek na celotnem disku.

Zaščita datotek je omejena na Read-Only atribut, ki ga lahko nastavlja vsak uporabnik. Delno je torej preprečeno nenamerno brisanje datotek, namerne poškodbe podatkov ne moremo preprečiti.

Operacijski sistem podpira 16 uporabnikov, vendar ne nudi nobene zaščite med uporabniškimi področji.

2.2.2 MS-DOS

Operacijski sistem MS-DOS podpira drevesno strukturo kazala, vendar je velikost glavnega kazala fiksna. Ne podpira uporabnikov, prav tako ne podpira zaščite datotek.

Osnovna enota dodeljevanja je odvisna od velikosti diska in se giblje od 1K do 4K. Skupno število datotek na disku ni omejeno.

Informacija o položaju datoteke na disku se nahaja v posebni tabeli na začetku diska (FAT), posamezni elementi v tej tabeli so povezani s kazalci. Vsak element predstavlja en cluster (enoto dodeljevanja prostora).

2.2.3 FILES-11 struktura (RSX-11 in VMS)

FILES-11 struktura je sorazmerno precej zahtevna diskovna struktura. Omogoča drevesno strukturo datotek, zaščito datotek med uporabniki, omogoča neomejeno število uporabnikov. Vsi parametri diskovne strukture (število datotek, velikost datoteke ipd.) so neomejeni, omejitev je le fizična velikost diskovne enote.

Informacija o položaju datotek na disku je zapisana v posebnih sektorjih (FILE HEADER), ki so zbrani v datoteki (0,0)INDEXF.SYS, kar poenostavi kontrolo pravilnosti diskovne strukture in omogoča iskanje izgubljenih datotek. Operacijski sistem edini med naštetimi podpira verzije datotek.

2.3 Naša izbira strukture datotek

Ob izbiri strukture datotek smo se skušali izogniti večini slabih strani vseh sistemov. Tako smo dobili strukturo, ki dokaj optimalno izpolnjuje večino ciljev, ki smo si jih zadali v začetku tega poglavja, vendar je omejena z uporabljeno tehnologijo (Z80 procesor in 64K bytov pomnilnika), ki nam je onemogočala realizacijo bolj zahtevne in bolj optimalne strukture.

Velikost sektorja na disku je 1024 bytov, velikost logičnega sektorja, kot ga vidi uporabniški program pa je 128 bytov, da ohranimo združljivost s CP/M operacijskim sistemom. Ves dostop uporabniškega programa do datotek poteka preko logičnih sektorjev velikosti 128 bytov.

Informacija o imenu datotek je zbrana v kazalu datotek. Kazalo datotek je datoteka tako kot vsaka druga datoteka, zato je lahko kazalo neomejeno veliko. Operacijski sistem podpira drevesno strukturo kazala, vsako vozlišče v drevesni strukturi je spet datoteka. Tako smo dosegli enoten pristop do vseh datotek v operacijskem sistemu, kar poenostavi pisanje sistemskih programov a tudi samega operacijskega sistema. Poleg tega obstajata v vsakem kazalu posebni datoteki . in .. (trenutno in predhodno kazalo), kar še olajša sistematski pristop do drevesne strukture kazala. Za definicijo poti do posameznega vozlišča je uporabljena UNIX sintaksa (npr. /USR/SAMPLE/TEST).
Kot vidimo, je v kazalu datotek zapisano samo ime datoteke, njena zaščita in položaj ostalih informacij o datoteki. S tem dosežemo, da je velikost enega zapisa samo 16 bytov, torej spravimo v en fizičen sektor (1K byt) 64 zapi-

```
Type Directory_Record = Record
        File_Name        : String (8)
        File_Type        : String (3)
        First_Descriptor : Integer
        Protection       : Array (1..4) of nibble
        Reserved_Byte    : Byte
```

Slika 2.1 Struktura zapisa v kazalu datotek

sov, kar je dovolj za večino primerov, torej lahko celotno kazalo večinoma dosežemo z enim samim dostopom do diska, kar bistveno vpliva na hitrost odpiranja datotek.

Zaščita datoteke je razdeljena v štiri dele :

- uporabnik (USER_ID datoteke je enak USER_ID uporabnika)
- uporabniška skupina (GROUP_ID datoteke je enak GROUP_ID uporabnika)
- ostali svet (USER_ID in GROUP_ID datoteke in uporabnika se razlikujeta)
- privilegirani sistemski uporabnik (USER_ID = 0). Za takšnega uporabnika sistem ne testira zaščite datotek.

Vsaka skupina uporabnikov (OWNER, GROUP, WORLD) ima v dveh zlogih zaščite prirejene 4 bite, ki predstavljajo dovoljenja posamezne skupine uporabnikov :

- READ -- uporabnik lahko datoteko bere
- WRITE -- uporabnik lahko datoteko piše
- EXECUTE -- uporabnik lahko datoteko izvaja
- DELETE -- uporabnik lahko datoteko briše

Za kazala, ki so po definiciji tudi datoteke, je pomen posameznih bitov nekoliko spremenjen :

- READ -- uporabnik lahko bere kazalo
- WRITE -- uporabnik lahko kreira ali briše datoteke
- EXECUTE -- uporabnik lahko odpira datoteke
- DELETE -- uporabnik lahko briše kazalo

Četrta skupina štirih bitov je neuporabljena za zaščito datotek in služi kot opis dodatnih lastnosti datoteke :

- DIRECTORY -- datoteka je kazalo
- SYSTEM -- datoteka je sistemska datoteka

Z implementacijo zaščite nad kazali datotek lahko uvedemo nekatere zanimive izvedbe zaščite datotek :

- uporabnik lahko samo pregleduje kazalo, vendar ne more odpreti datoteke v kazalu (READ)
- uporabnik lahko odpre datoteko, če pozna njeno ime, vendar ne more izpisati imen datotek (EXECUTE)
- uporabnik lahko samo lista kazalo in odpira datoteke (READ in EXECUTE)
- uporabnik lahko samo kreira nove datoteke, ko je datoteko enkrat kreiral, je ne more več brati (WRITE). Preprečevanje brisanja datotek dosežemo z ustrezno zaščito datotek pri kreiranju.

Informacija o položaju datoteke na disku je zbrana v dveh podatkovnih strukturah :

- Za prvih 256K bytov datoteke v opisu datoteke (FILE DESCRIPTOR) -- en sektor. V tem sektorju so zbrane tudi vse ostale informacije o datoteki.
- Za podatke preko 256K bytov datoteke v razširjenem opisu datoteke (EXTENDED FILE DESCRIPTOR) -- en sektor za vsakih 512K bytov

```
Type File_Descriptor = Record
       Open_Count         : Byte
       User_ID            : Byte
       Group_ID           : Byte
       Last_Block         : Double Integer
       Access_Date        : Date
       Update_Date        : Date
       Create_Date        : Date
       Backup_Date        : Date
       Descriptor_Pointer : Array (0..127) of Integer
       Data_Pointer       : Array (0..255) of Integer
     End ;
```

Slika 2.2 Struktura opisa datoteke

Polja USER_ID in GROUP_ID imajo znan pomen. Polje OPEN_COUNT služi za štetje števila dostopov do datotek, ki jih hkrati odpre več uporabnikov. Polje LAST_BLOCK pove številko zadnjega logičnega bloka (128 bytov) v datoteki. V opisu datoteke hrani operacijski sistem datum zadnjega dostopa, spreminjanja, kreiranja in arhiviranja. Te datume lahko uporabljamo za kontrolo dostopa do datoteke ali za delno arhiviranje podatkov na disku (arhiviramo samo podatke kjer je UPDATE_DATE > BACKUP_DATE). Datum je natančen do sekund.

DESCRIPTOR_POINTER je tabela kazalcev na sektorje, ki vsebujejo razširjen opis datoteke. DATA_POINTER je tabela kazalcev na prvih 256 podatkovnih sektorjev. S takšno zasnovo opisa položaja datoteke na disku smo dosegli naslednje omejitve :

- velikost diska največ 64M bytov
- velikost datoteke največ 128M bytov, torej več od velikosti diska, kar pomeni, da je velikost datoteke dejansko omejena z velikostjo diska.

Ker ima velika večina danes dostopnih diskov, primernih za vgraditev v mikroračunalnik, kapaciteto manjšo od 64M bytov, ta omejitev velikosti diska ne bi smela predstavljati resne ovire. Če želimo uporabljati večji disk, ga lahko razdelimo na particije velikosti 64M bytov in ga uporabljamo kot več logičnih diskov velikosti 64M bytov.

Zasedenost diska je zapisana v posebni datoteki (/BITMAP.SYS). Z uporabo posebne datoteke dosežemo bistveno manjši zagonski čas sistema v primerjavi s sistemi, ki ob zagonu računalnika gradijo tabelo zasedenosti diska v spominu (CP/M). Poleg tega je gradnja takšne tabele v pogojih drevesne strukture kazala praktično nemogoča, saj bi zasedla preveč spomina. Edina slabost takšne zasnove bitne tabele se pokaže ob izpadu sistema, saj je takrat tabela poškodovana in potrebujemo poseben program, ki jo popravi nazaj v korektno stanje.

Za vzpostavitev vseh sistemskih struktur potrebujemo še dva kazalca :

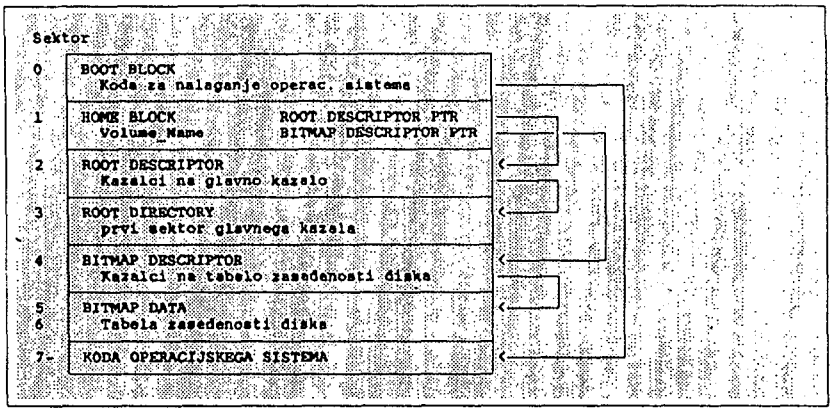
- kazalec na opis datoteke / (vrh drevesne strukture kazala)
- kazalec na opis datoteke /BITMAP.SYS (za lažje delo, čeprav bi ga lahko dobili iz datoteke /).

Ta dva kazalca sta zapisana v HOME BLOCK sektorju diska (sektor 1 v 0. stezi), ki ima sledečo strukturo

```
Type Home_Block = Record
       Volume_Name       : String (16)
       Root_Descriptor   : Integer
       Bitmap_Descriptor : Integer
       Bitmap_Size       : Integer
     End ;
```

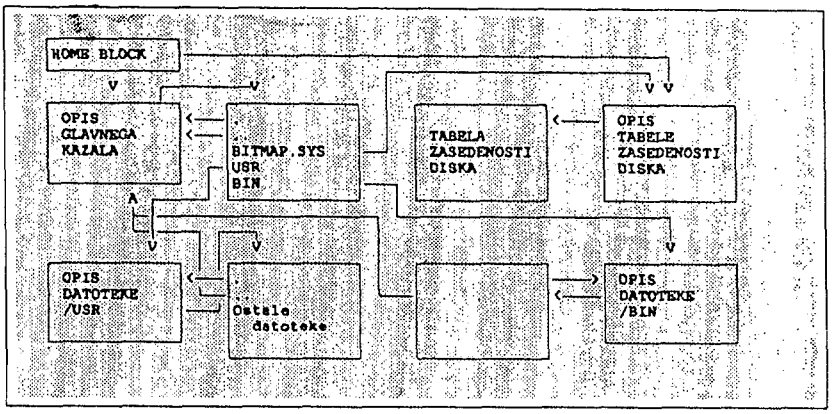
Slika 2.3 Struktura HOME BLOCK sektorja

Sektor 0 v 0. stezi je rezerviran za program, ki naloži operacijski sistem ob zagonu računalnika. Tako je standardna oblika prve steze diska sledeča :

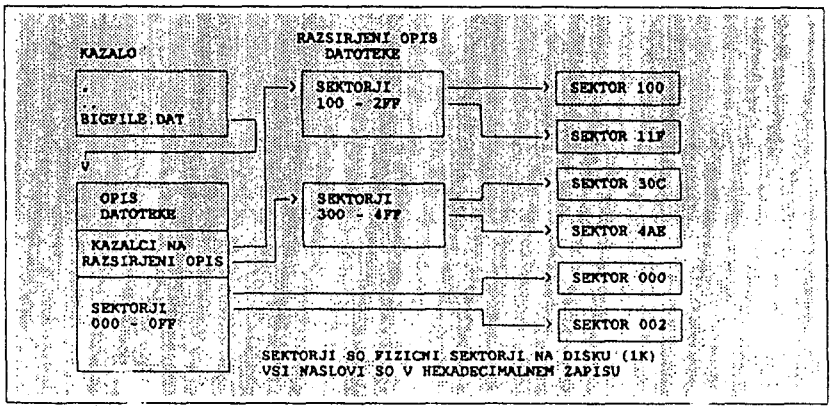


Slika 2.4 Struktura prve steze diska

Za ilustracijo zgoraj navedenih pojmov si oglejmo še primer strukture diska, ki ima v glavnem kazalu samo dva poddirektorija /BIN in /USR. Prikazane so vse povezave med podatkovnimi strukturami na disku in način kako operacijski sistem doseže določene podatkovne strukture.



Slika 2.5 Primer diskovne strukture



Slika 2.6 Primer zelo velike datoteke

Lastnost	CP/M	MS-DOS	FILES-11	FENIX
velikost kazala	omejena	omejena za glavno kazalo	64K datotek za RSX-11	neomejena
velikost zapisa v kazalu	32 bytov	32 bytov	16 bytov	16 bytov
drevesna struktura kazala	NE	DA	NE za RSX-11 DA za VMS	DA
zapis polozaja datoteke na disku	v kazalu	veriga (FAT)	v FILE HEADER	v opisu datoteke
maksimalna velikost datoteke	32M bytes	velikost diska	velikost diska	64M bytes
globina kazala	---	neomejeno	2 za RSX-11	neomejeno
stevilo uporabnikov	16	---	64K za RSX-11 neomejeno VMS	255
zascita datotek	NE	NE	DA	DA
casovne oznake	DA (2)	DA (2)	DA (4)	DA (4)
osnovna enota dodeljevanja prostora	2-4K	1-4K	0.5K	1K

Opomba : Oznaka neomejeno pomeni, da ima parameter tako veliko vrednost, da je za naso primerjavo neomejeno velika.

Slika 2.7 Primerjava diskovne strukture sistema FENIX z drugimi operacijskimi sistemi

3. Struktura operacijskega sistema FENIX

3.1 Uvod

Operacijski sistem FENIX je samostojen program, ki teče v glavnem računalniku mreže DANTE/NET. Njegova edina naloga je odgovarjati na zahteve drugih računalnikov v mreži. Glavna zanka operacijskega sistema je zato zasnovana tako

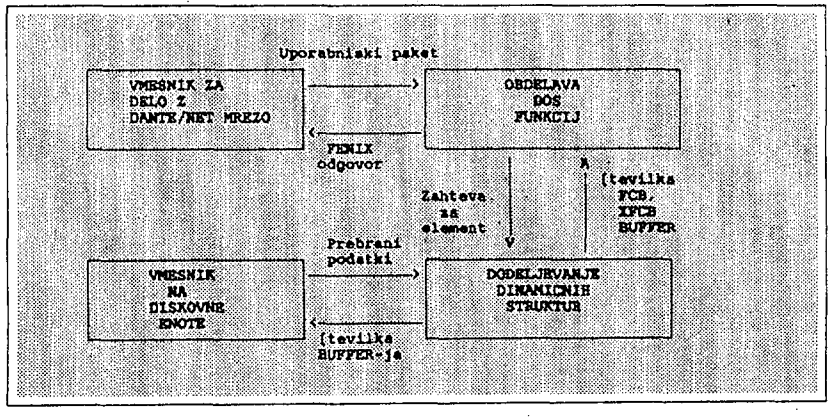
```
Initialize System
Repeat
      Get network message
      Process message
      Send network message
Until Shutdown_Request
Shutdown system
Halt CPU
```

Slika 3.1 Glavna zanka operacijskega sistema

Če bi hoteli doseči večjo efektivnost dela, bi morali implementirati paralelnost procesov PROCESS_MESSAGE, kar bi pomenilo, da operacijski sistem hkrati streže več dostopom do diska.

Operacijski sistem lahko razdelimo v več modulov :

- komunikacija z DANTE/NET mrežo
- izvajanje DOS funkcij
- dodeljevanje in vzdrževanje vmesnikov
- krmilnik diskovnih enot



Slika 3.2 Razdelitev operacijskega sistema *FENIX*

3.2 Opis posameznih modulov

3.2.1 Komunikacija z DANTE/NET mrežo

Naloga tega modula je sprejem in oddaja paketa v DANTE/NET mrežo. Mreža sama je collision-avoidance mreža tipa Ethernet, uporabljen protokol je SDLC. Paket sam je sestavljen iz več delov :

- krmilne informacije (sprejemnik, oddajnik)
- opis DOS funkcije
- parametri za DOS funkcijo
- kontrolni seštevek

Ko modul za komunikacijo z DANTE/NET mrežo sprejme paket, zasede v pomnilniku 256 bytov velik blok, ki ga uporablja za shranjevanje paketa z mreže in za shranjevanje vseh sistemskih spremenljivk, ki jih potrebuje DOS za obdelavo te DOS funkcije. Ko je paket uspešno sprejet, ga modul preda modulu za obdelov DOS funkcij. Po izvršeni DOS funkciji modul prejme izhodni paket, ki ga pošlje računalniku, ki je zahteval DOS funkcijo.
Parametri za DOS funkcijo so lahko :

- FCB ID -- številka FCB - ja, ki opisuje datoteko, s katero hoče uporabnik delati. FCB ID je sestavljen iz dveh bytov : številka FCB - ja in zaporedna številka dostopa do tega FCB-ja, s pomočjo katere se kontrolira pravilnost dostopa in preprečuje branje datotek drugih uporabnikov
- Številka zapisa v datoteki, ki ga hočemo brati ali pisati
- Celoten FCB, kot ga podpira CP/M (16 bytov)
- Zastavice v CP/M FCB-ju
- DMA področje (vsebina zapisa, ki ga beremo ali pišemo)

Poleg teh parametrov lahko vrne operacijski sistem FENIX uporabniku še :

- status DOS funkcije

3.2.2 Izvajanje DOS funkcij

Modul skrbi za izvajanje vseh DOS funkcij, ki jih drugi računalniki zahtevajo po mreži.
V inicializaciji podatkovnih struktur modul prepiše podatke o uporabniku v začasne spremenljivke ter določi logični disk, s katerim uporabnik dela in njemu pripadajoči fizični disk in direktorij.

```
Get network packet
Initialize data structures
If error then Send reject packet
        else Process DOS function
             Send result packet
```

Slika 3.3 Osnovna struktura modula za izvajanje DOS funkcij

Po končani inicializaciji modul preko tabele kliče ustrezno DOS funkcijo, ki obdela uporabnikovo zahtevo in zgradi paket, ki ga modul preko komunikacijskega modula vrne uporabniku.

3.2.3 Dodeljevanje in vzdrževanje vmesnikov

Modul skrbi za dodeljevanje in vzdrževanje vseh dinamičnih sistemskih podatkovnih struktur :

- vmesniki za delo z diskom
- opis odprte datoteke (FCB)
- opis večkratno odprte datoteke (XFCB)

Natančen algoritem dodeljevanja teh struktur je opisan v razdelku SISTEMSKE PODATKOVNE STRUKTURE.

3.2.4 Krmilnik trdega diska

Modul skrbi za povezavo ostalega operacijskega sistema z diskovnimi enotami. Modul kliče modul za dodeljevanje in vzdrževanje vmesnikov v sledečih primerih :

- ko zahteva branje novega sektorja z diska
- ko zahteva zapis sektorja na disk v primeru, ko zmanjka vmesnikov
- ko zahteva zapis sektorja na disk ob periodičnem čiščenju sistemskih vmesnikov

3.3 Sistemske podatkovne strukture

Operacijski sistem FENIX potrebuje za svoje delovanje naslednje podatkovne strukture :

- vmesnike za delo z diskom
- opis odprtih datotek (FCB)
- opis večkratno odprtih datotek (XFCB)
- opis uporabnikov (USER)
- opis diskovnih enot

Vse dinamične podatkovne strukture (vmesniki za delo z diskom, FCB in XFCB) se dinamično dodeljujejo in odvzemajo ob prezasičenosti sistema. Operacijski sistem vodi za te tri dinamične strukture statistiko njihove uporabe (LRU algoritem), tako, da lahko po potrebi odvzame najmanj uporabljan element.
LRU algoritem je realiziran s pomočjo vrste :

- ob kreiranju se element postavi na začetek vrste
- ob vsakem dostopu operacijski sistem premakne element na začetek vrste
- LRU element je element na koncu vrste

Opisan algoritem je enostaven za realizacijo in daje dovolj dobre rezultate.

3.3.1 Opis odprtih datotek

```
Type File_Control_Block = Record
     Disc_Drive          : Byte
     File_Name           : String (8)
     File_Type           : String (3)
     Sequence_Number     : Integer
     First_Descriptor    : Integer
     First_Buffer        : Byte
     Current_Descriptor  : Integer
     Current_Buffer      : Byte
     FCB_Semaphore       : Byte
     Next_Pointer        : Integer
     Prev_Pointer        : Integer
     Protection          : Byte
     Node_ID             : Byte
     FCB_Flags           : Byte
     Access_Count        : Byte
End ;
```

Slika 3.4 Opis odprte datoteke

Opis posameznih polj :

Disc_Drive — diskovna enota, na kateri je odprta datoteka

File_Name — ime datoteke

File_Type — tip datoteke

Sequence_Number — zaporedna številka uporabe tega elementa. Ko operacijski sistem odvzame FCB uporabniku, ki ga že dolgo ni uporabljal, poveča sequence number in tako onemogoči ponovno uporabljanje odvzetega FCB-ja

First_Descriptor
Številka sektorja na disku, ki vsebuje opis datoteke

First_Buffer
Številka vmesnika, ki vsebuje opis datoteke ali 0, če opisa datoteke ni v spominu

Current_Descriptor
Številka sektorja na disku, ki vsebuje razširjeni opis datoteke za zadnji prebrani sektor. To polje se uporablja le če beremo datoteke, ki so večje od 256K bytov, drugače je enako First_Descriptor

Current_Buffer
Številka vmesnika, ki vsebuje sektor, na katerega kaže Current_Descriptor. Če tega sektorja ni v spominu, vsebuje to polje 0.

FCB_Semaphore neuporabljeno

Next_Pointer
Prev_Pointer Kazalci za implementacijo LRU algoritma

Protection bit 0 -- 3
zaščita datoteke, ki je odvisna od zaščite datoteke v opisu datoteke in uporabnika in grupe računalnika, ki je odprl datoteko.
bit 4 -- 7
dodatni atributi datoteke

Node_ID Številka računalnika, ki je odprl datoteko

FCB_Flags Zastavice, ki povedo stanje FCB-ja
bit 7 FCB je zaklenjen v spominu
bit 6 polje First_Descriptor je OK
bit 5 FCB je neveljaven
bit 4 FCB je uporabljen

Access_Count Števec dostopov do FCB-ja. Števec se poveča ob vsakem OPEN ukazu in zmanjša ob vsakem CLOSE ukazu. Ko doseže vrednost števca 0, operacijski sistem izvede dokončno CLOSE datoteke.

3.3.1.1 Dodeljevanje FCB-jev

Operacijski sistem zahteva nov FCB ob DOS funkciji, ki mora za svoje delo odpreti novo datoteko (OPEN, MAKE, DELETE, RENAME, GET FILE SIZE ipd.). Če je na razpolago prost FCB, ga modul za dodeljevanje vmesnikov zaseže, vpiše v FCB_FLAGS bit 7 in 4 ter ga vrne DOS modulu.

Če operacijski sistem ne more najti prostega FCB-ja, poišče zadnji FCB v listi, ki ni zaklenjen v spomin, zapre datoteko, ki ji ta FCB pripada in ga vrne DOS modulu.

Da bi preprečili dostop do tujih podatkov in zlorabo operacijskega sistema, poveča operacijski sistem ob vsakem OPEN klicu polje SEQUENCE_NUMBER v FCB-ju. Če hoče uporabniški program brati ali pisati datoteko, mora poleg številke FCB-ja v paketu, ki ga pošlje preko mreže podati še zaporedno številko v FCB-ju. Če se ti dve številki ne ujemata, je bila datoteka medtem zaprta in uporabniški program mora ponovno izvršiti OPEN klic. Ta protokol je implementiran že v mrežnem delu CP/M operacijskega sistema na ostalih računalnikih v mreži in je tako za končnega uporabnika neviden.

Ker bi lahko uporabnik uganil pravilno zaporedno številko v FCB-ju primerja operacijski sistem poleg zaporedne številke še številko vozlišča s katerega je bila datoteka odprta in številko vozlišča s katerega je prišla zahteva za branje oz. pisanje

Operacijski sistem vzdržuje v vsakem FCB-ju polje Access_Count ki šteje število OPEN klicev izvršenih na tem FCB-ju. Ko je polje Access_Count pri CLOSE klicu enako 0, lahko operacijski sistem izvrši dokončen CLOSE datoteke. Operacijski sistem tedaj sprosti FCB, ki je pripravljen za naslednji OPEN klic.

Če več uporabnikov hkrati odpre isto datoteko, vrne operacijski sistem prvemu uporabniku številko FCB-ja, s katerim je odprl to datoteko, za vse naslednje uporabnik pa uporabi operacijski sistem novo podatkovno strukturo XFCB. Takšen algoritem omogoča vodenje evidence o odpiranju datotek in preprečuje izgubo podatkov zaradi brisanja in hkratnega branja oz. pisanja datoteke (če bi vodili le evidenco o prvem vozlišču, ki je odprlo datoteko, bi lahko zbrisali datoteko, ki jo dosega več uporabnikov ali pa bi preprečili brisanje datoteke, ki jo je en uporabnik večkrat odprl). S stališča uporabniškega programa se XFCB ne razlikuje od FCB-ja, razlika je le v tem, da ima XFCB zaporedno številko od 128 do 255.

Opomba : Dodeljevanje FCB-jev je še oteženo zaradi slabega sloga programiranja v večini CP/M programov, ki praviloma ne zapirajo datotek, ki jih ne rabijo več, tako, da mora operacijski sistem hevristično določati katere datoteke program še uporablja in katere lahko zapre. Napake v tem hevrističnem procesu privedejo do ponovne uporabe zaprte datoteke in ponovnega OPEN klica, ki je, kot smo že poudarili, neviden za uporabniški program.

3.3.2 XFCB -- opis večkrat odprte datoteke

Slika 3.5 Struktura XFCB - ja

Operacijski sistem kreira enega ali več XFCB za vsako datoteko, ki jo hkrati odpre več kot en uporabnik. Prvi uporabnik uporablja za dostop do datoteke številko FCB-ja, vsi naslednji uporabniki uporabljajo za dostop do datoteke številko XFCB-ja, ki kaže na pravilni FCB. S takšnim pristopom je omogočeno vodenje natančne evidence o odpiranju datotek, poleg tega je število uporabnikov, ki lahko hkrati odprejo eno datoteko neomejeno.

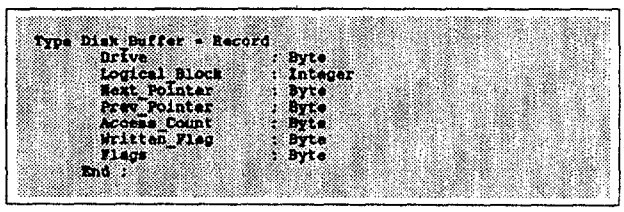
3.3.2.1 Dodeljevanje XFCB :

Operacijski sistem dodeli nov XFCB vsakič ko uporabnik poskuša odpreti datoteko, ki jo je že odprl drug uporabnik. Če modul za dodeljevanje vmesnikov ugotovi, da ne more dobiti prostega XFCB-ja, odvzame enega od uporabljenih XFCB-jev, zapre FCB, ki ga je ta XFCB naslavljal (in tako zmanjša ACCESS_COUNT) in vrne nov XFCB modulu za obdelavo DOS funkcij.

Ob CLOSE ukazu operacijski sistem sprosti XFCB in izvede CLOSE ukaz na ustreznem FCB-ju.

Obdelava odvzetih XFCB-jev, ki jih uporabnik hoče ponovno uporabiti je enaka kot za odvzete FCB-je.

3.3.3 Vmesnik za delo z diskom



Slika 3.6 Struktura vmesnika za delo z diskom

Polji Drive in Logical_Block opredelita diskovno enoto in zaporedno številko sektorja na tej enoti. Polji Next_Pointer in Prev_Pointer služita za implementacijo LRU algoritma. Polje Access_Count služi za štetje števila dostopov do tega vmesnika.

Polje Written_Flag služi za indikacijo vpisovanja v ta vmesnik in vsebuje 1 bit za vsak logični blok (128 bytov) znotraj fizičnega sektorja (1 K byte).

V polju FLAGS operacijski sistem hrani stanje tega vmesnika :

Bit 7	Vmesnik je zaklenjen v spomin
Bit 6	Vsebina vmesnika je spremenjena
Bit 5	Modul za povezavo z diski piše vmesnik na disk
Bit 4	Vmesnik bo zamenjan
Bit 3	Modul za povezavo z diski bere vmesnik z diska

Bita 5 in 3 služita za sinhronizacijo med modulom za delo z diskovnimi enotami in ostalimi moduli operacijskega sistema, saj bi se ob uporabi prekinitev in zakasnjenega pisanja na disk zlahka zgodilo, da bi modul za obdelavo DOS funkcij uporabljal vmesnik, ki še ni bil prebran z diska ali ki se ravnokar vpisuje na disk.

Bit 4 služi za sinhronizacijo med modul za dodeljevanje vmesnikov in modulom za obdelavo DOS funkcij, saj preprečuje uporabo vmesnika, ki ga bo modul za dodeljevanje vmesnikov zamenjal.

Opisi vmesnikov so zbrani v tabeli, ki vsebuje 256 elementov, sami vmesniki se nahajajo v ostanku spomina ali v drugih spominskih bankah v mikroračunalnikih, ki imajo več kot 64K bytov spomina.

3.3.3.1 Dodeljevanje vmesnikov :

Modul za obdelavo DOS funkcij zahteva določen logični blok z diska. Če je ta blok že v enem od vmesnikov, ga modul za dodeljevanje vmesnikov vrne in ga premakne v LRU vrsti, drugače poišče modul za dodeljevanje vmesnikov prost vmesnik in zahteva branje tega vmesnika z diska.

Če modul za dodeljevanje vmesnikov ne more najti prostega vmesnika, poskuša najti popolnoma zapisan vmesnik (Written_Flag = 255) in ga zapiše na disk, zahteva branje vmesnika in ga vrne.

Če nobeden od vmesnikov ni popolnoma poln, modul zapiše na disk prvi dostopen vmesnik in zahteva branje novega vmesnika.

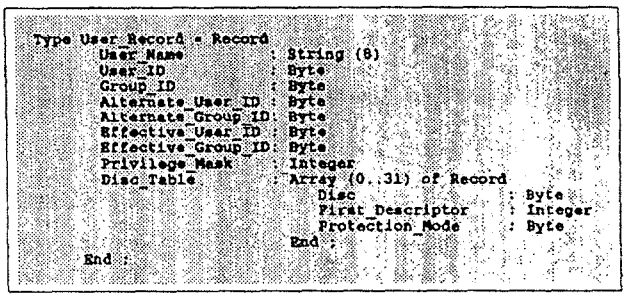
Če so vsi vmesniki zasedeni (LOCKED), modul sprosti vse vmesnike, ki so bili dodeljeni odprtim datotekam za opis datoteke in označi v vseh FCB-jih, da niso več pravilno nastavljeni. Po tej operaciji modul ponovno poskuša dobiti prazen vmesnik s pomočjo zgornjih treh točk.
Če je med iskanjem vmesnika modul za dodeljevanje vmesnikov zašel pregloboko v listo vmesnikov, torej je so vmesniki že precej zasedeni, bo modul zahteval zapis vseh popolnoma vpisanih vmesnikov (Written_Flag = 255) na disk. Če na ta način ne pridobi dovolj vmesnikov, bo modul zahteval še zapis vseh delno napolnjenih vmesnikov na disk.

Ob hkratnem delu modula za delo z diskovno enoto in ostalega operacijskega sistema ter dovoljšnjem številu vmesnikov (več kot 128) lahko s tem algoritmom dovolj zgodaj ugotovimo možno zasičenje sistema in zahtevamo vpis polnih vmesnikov na disk preden pride do popolnega zasičenja sistema. Takšno vpisovanje vmesnikov na disk poteka hkrati z ostalim delom sistema in dela sistema praktično ne prekinja. Zapisovanje vmesnikov na disk ob zasičenju sistema blokira sistem do konca zapisovanja in tako zmanjša hitrost delovanja sistema.

3.3.4 Opis uporabnika

Vsako vozlišče, ki se lahko pojavi v mreži, ima v glavnem računalniku ustrezno podatkovno strukturo, ki definira uporabnika, ki dela na tem vozlišču.
V tej podatkovni strukturi so zbrani vsi podatki o uporabniku, ki jih operacijski sistem potrebuje za svoje delo :

- Ime uporabnika



Slika 3.7 Opis uporabnika

- USER_ID in GROUP_ID sta polji, ki ju sistem uporablja ob določanju zaščite datoteke proti določenemu uporabniku
- ALTERNATE_USER_ID in ALTERNATE_GROUP_ID sta polji, ki ju sistem uporablja za shranjevanje USER_ID in GROUP_ID v primeru, ko uporabnik s pomočjo privilegiranega ukaza menja svoj USER_ID.
- EFFECTIVE_USER_ID in EFFECTIVE_GROUP_ID sta polji, ki ju operacijski sistem uporablja za shranjevanje USER_ID in GROUP_ID ob zagonu programa, ki začasno menja USER_ID in GROUP_ID na svoj USER_ID in GROUP_ID. Takšen program lahko izvaja privilegirane funkcije tudi za navadnega uporabnika.
- PRIVILEGE_MASK vsebuje definicijo privilegijev, ki jih ima uporabnik, torej seznam vseh zaščitenih operacij, ki jih uporabnik lahko izvaja.
- DISC_TABLE vsebuje preslikavo logičnih diskov v poddirektorije. Vsak element tabele vsebuje opis enega logičnega diska (A: do P: so stalni, ostali ostanejo samo za čas delovanja programa) in sicer :
  - diskovna enota, kjer se nahaja logični disk
  - kazalec na opis direktorija, ki ustreza logičnemu disku
  - zaščita direktorija, kot jo določi operacijski sistem ob OPEN klicu

S pomočjo tabele logičnih diskov lahko operacijski sistem zelo enostavno določi s katerim poddirektorijem hoče uporabnik delati, saj mu ni treba ponovno izvajati OPEN ukaza.

3.3.5 Opis diskovne enote

```
Type Disk_Description = Record
       Drive             : Byte
       Flags             : Byte
       Volume_Name       : String (8)
       Root_Descriptor   : Integer
       Bitmap_Descriptor : Integer
       Bitmap_Size       : Byte
       Dismount_Flag     : Byte
       Test_Offline      : Integer
       DOS_Init          : Integer
       DOS_Entry         : Integer
       Bitmap_First      : Byte
       Bitmap_Count      : Byte
       Bitmap_Buffer     : Integer
       Bitmap_Pointers   : Array (0..7) of Integer
     End ;
```

Slika 3.8 Opis diskovne enote

Opis diskovne enote vsebuje vse informacije, ki jih operacijski sistem potrebuje za delo z diskom :

- ime diska (samo za informacijo uporabnika)
- kazalec na opis glavnega kazala
- kazalec na opis tabele zasedenosti diska in njena velikost
- naslov procedure za
  - testiranje prisotnosti diska
  - inicializacijo diska
  - izvedbo DOS funkcije na tem disku
- Polja za vnaprejšnje dodeljevanje prostih sektorjev

Naslova procedure za inicializacijo diska in izvedbo DOS funkcije na disku omogočata modularno izvedbo diskovnega operacijskega sistema in celo uporabo večih diskovnih struktur na različnih diskih (npr. CP/M na disku A:, FENIX na disku B:).

3.3.5.1 Dodeljevanje prostih sektorjev :

Ko modul za obdelavo DOS funkcij zahteva prosti sektor na disku, skuša modul za dodeljevanje vmesnikov to zahtevo najprej zadovoljiti s pomočjo vnaprej zavzetih sektorjev v spominu. Če to ni možno, modul dodeli sektor s pomočjo tabele prostih sektorjev in dodatno zasede še 254 sektorjev, ki jih bo uporabil pri naslednjih zahtevah za dodelitev praznega sektorja.

Ta način dodeljevanja spomina precej izboljša dodeljevanje novih sektorjev na disku, saj v trenutku, ko je podatkovni sektor tabele prostih sektorjev v spominu skuša le-tega čimbolje izkoristiti. Tako zahteva dodeljevanje novih sektorjev dostop do diska le na vsakih 256 sektorjev, vmesnik, ki bi drugače hranil tabelo prostih sektorjev pa je lahko v vmesnem času sproščen.

Ob brisanju datoteke skuša modul za dodeljevanje vmesnikov vključiti novo sproščeni vmesnik v seznam vnaprej dodeljenih vmesnikov. Ta operacija uspe le v primeru, ko je seznam dodeljenih vmesnikov dovolj prazen in ko je sproščeni sektor dovolj blizu začetka diska. Ta zadnji pogoj preprečuje vnašanje sektorjev globoko znotraj diska v tabelo vnaprej dodeljenih sektorjev in tako zmanjša gibanje glave po disku.

4. ZAKLJUČEK

Operacijski sistem FENIX smo uspešno implementirali na mikroračunalniku DIALOG z 10M bytnim trdim diskom. Za delo z operacijskim sistemom je bil razvit poseben interpreter ukazov in podporni programi, ki so ekvivalentni UNIX podpornim programom, tako, da celota FENIX operacijski sistem in SHELL interpreter ukazov uspešno predstavljata UNIX - združljivo okolje v 8 bitni tehnologiji.

Operacijski sistem deluje dokaj hitro v primerjavi z drugimi mrežnimi operacijskimi sistemi realiziranimi v 8 bitni tehnologiji, precej zaslug za to nosi izredno hitra lokalna mreža DANTE/NET (hitrost prenosa 1M bit/s). Ob testiranju se je operacijski sistem obremenjen s štirimi delovnimi postajami izkazal za hitrejšega kot delo na sami delovni postaji z disketo, torej je njegova uporaba vsekakor upravičena.

Ob razvijanju operacijskega sistema se je rodilo nekaj idej, ki so vredne razmisleka in morda implementacije v naslednji verziji operacijskega sistema :

- podpora zaklepanja zapisov v datotekah, ki bi na sistemskem nivoju rešila problem hkratnega dostopa v bazo podatkov.
- integracija večih diskovnih pogonov v eno diskovno strukturo kot jo pozna UNIX operacijski sistem (MOUNT in UMOUNT ukaz)
- podpora datotek, ki bi se raztezale preko večih diskov. Tako bi lahko dosegli do 128M bytov velike datoteke brez spreminjanja diskovne strukture, ob manjših spremembah pa bi lahko dosegli še bistveno večje datoteke (cca. 4G byte).
- implementacija večih diskovnih struktur v okviru enega operacijskega sistema (npr. en CP/M disk in en FENIX disk)
- implementacija zahtevnejše diskovne strukture (npr. FILES-11 level 2)

Za implementacijo zadnjih ciljev bi bilo bolje uporabiti modernejšo tehnologijo kot je Z80 procesor, saj nam je 8 bitna tehnologija že ob razvijanju FENIX operacijskega sistema postavljala precej omejitev, predvsem zaradi omejitve naslovnega prostora na 64K bytov.

Literatura :

(1) Možnosti in nemožnosti mikroračunalnika DIALOG,Elektrotehniški vestnik, April 1987
(2) Peterson, Silbershatz, Operating system Concepts, Addison - Wesley 1983
(3) Comer, Operating System design, the XINU Approach, Prentice - Hall 1984
(4) XENIX operating system, Microsoft corp., 1986
(5) VAX/VMS V4.0 operating system, Digital equipment corporation, 1985
(6) IBM DOS 3.2 Technical Reference Manual, IBM Corp., 1985
(7) IBM DOS 2.0 Manual, IBM Corp. 1983
(8) Hyman, Memory resident utilities, interrupts and disk management, MIS 1986