SISTEMSKI UPORABNIŠKI VMESNIK ZA 8-BITNI UNIX ZDRUŽLJIV OPERACIJSKI SISTEM

INFORMATICA 1/88

Marjan Bradeško
Ljubo Pipan
Ivan Pepelnjak
Nikolaj Zimic
Fakulteta za elektrotehniko, Ljubljana

UDK 681.3.06

Izvleček

Članek opisuje izboljšavo 8-bitnega operacijskega sistema do te mere, da ga uporabnik z vidika sintakse vidi kot operacijski sistem Unix. Navedene so nekatere splošne značilnosti novega sistema, natančneje pa je opišan visokonivojski sistemski vmesnik, ki je v veliko pomoč uporabniku pri pisanju sistemskih uslužnostnih programov. S tem ima uporabnik sam možnost širiti nabor ukazov, ki predstavljajo vez med njim in operacijskim sistemom. Z že obstoječimi programi in zunanjim videzom predstavlja opisani operacijski sistem korak naprej pri prehajanju na bistveno zmogljivejše in bolj fleksibilne sisteme kot je CP/M, na osnovi katerega sloni nadgradnja, opisana v pričujočem članku.

Abstract

The paper presents the improvements of an 8-bit operating system to Unix - compatible environment as seen through user-level command line interface. We are describing some general features of the new operating system along with high - level system interface, which help the system programmers in writing new system utilities. With this aid, the end user can write his own utility programs, which represent the link between end-user and operating system environment. Existing programs can run under simulated CP/M environment, so this operating system represents a significant link between existing 8 bit operating systems and new UNIX - based systems.

1. UVOD

Doba 8-bitnih mikroračunalniških sistemov počasi mineva, hkrati z njimi pa se umikajo tudi operacijski sistemi, napisani zanje. Razvoj gre v smeri 16- in 32-bitnih sistemov. Temu ustrezno se prilagajajo tudi operacijski sistemi.

Uporabnik, ki bi rad na hitro prešel iz 8-bitnih na novejše sisteme, se znajde pred težavo. Aplikacijska programska oprema je vsa napisana na kožo njegovemu 8-bitnemu operacijskemu sistemu in običajno ni enostavno prenosljiva na nove sisteme. S tem problemom in z eno od uspešnih razrešitev le-tega se bomo ukvarjali v pričujočem članku.

Pri našem delu smo izhajali iz 8-bitnega mikroračunalniškega sistema z operacijskim sistemom CP/M. Sisteme smo povezali v lokalno mrežo z enim glavnim računalnikom s trdim diskom kapacitete 20 M. Glavni računalnik skrbi izključno za nadzor mreže, v katero je lahko povezanih do 16 postaj z dvema gibkima diskoma, ki imata vsak kapaciteto po 800 K. Prenos po mreži poteka po koaksialnem kablu (hitrost 1 Mbit), tako da je dostop iz katerekoli postaje do trdega diska glavnega računalnika hitrejši kot dostop na lokalni gibki disk postaje. S tem smo bistveno izboljšali performanse samega mikroračunalnika. S spremembo operacijskega sistema pa smo dobili sistem, ki navzven daje povsem drugačen videz od 8-bitnih sistemov. Uporabnik ga namreč z vidika sintakse vidi kot operacijski sistem UNIX, ki pa je že domena 16- in 32-bitnih sistemov. Pri vsem tem je v ozadju še vedno CP/M, tako da uporabnikova aplikacijska oprema ne potrebuje nobenih sprememb, hkrati pa je sistem zaradi povezave v mrežo in Unix-ove sintakse bistveno bolj fleksibilen.

Operacijski sistem na glavnem računalniku je povsem nov, na postajah pa je le modificiran CP/M. Pred BDOS je namreč dodan modul, ki ugotovi, če je neka sistemska funkcija rešljiva v okviru postaje, ali pa je potrebna intervencija glavnega računalnika. V tem primeru postaja pošlje zahtevo po servisiranju NDOS (Network DOS) funkcije v mrežo.

Operacijski sistem na glavnem računalniku je večuporabniški in večposlovni, servisiranje zahtev je rešeno s čakalnimi vrstami. Tudi podatkovne strukture na trdem disku so drugačne od CP/M struktur. Spremenjen je FCB (File Control Block) in zaradi drevesne strukture direktorijev tudi directory entry. Direktoriji so namreč datoteke, katerih elementi so imena

datotek v direktoriju in kazalci na opise (deskriptorje) teh datotek. Operacijski sistem CP/M dela z datotekami na fizičnih diskih A: oziroma B: (v primeru dveh pogonov), naš sistem pa za delo z datotekami uvaja logične diske. Če torej hočemo nek direktorij brati oziroma kaj pisati po njemu, mu moramo najprej prirediti enega od logičnih diskov. Šele potem so nam dostopne datoteke, ki se na njem nahajajo. Logični diski so vsi s številkami 3-32 (diski C:, D:, itd). Poleg standardnih Unix-ovih poddirektorijev /bin, /usr, /etc in /tmp imamo še dva poddirektorija

/lca - 'local A:' - CP/M diskovni pogon A:
/lcb - 'local B:' - CP/M diskovni pogon B:

V same podrobnosti jedra operacijskega sistema se tukaj ne bomo spuščali, ponovno poudarimo le to, da lahko poganjamo vso aplikacijsko programsko opremo, ki teče pod CP/M. Povejmo še, da za uspešno prehajanje med podatkovnimi strukturami CP/M postaje in glavnim računalnikom skrbita posebni proceduri Input Formatter in Output Handler operacijskega sistema na glavnem računalniku.

Pri našem opisu bomo ostali na nivoju vmesnika med uporabnikom in sistemom, kajti prav tu je najbolj fleksibilno mesto. Uporabnik ima namreč možnost širiti sistem s sistemskimi uslužnostnimi programi, ki jih je zaradi ustreznih orodij, ki jih ponuja nov sistem, zelo enostavno pisati. Podali bomo opis vmesnika, nekaterih sistemskih parametrov in na kratko našteli in opisali doslej napisane sistemske uslužnostne programe.

Uporabniški vmesnik je napisan v visokonivojskem programskem jeziku Turbo Pascal kot knjižnica uporabnih procedur, ki jo vključimo s stikalom ($I ime_knjižnice) v naš program. Izraz program bo odslej krajši izraz za sistemski uslužnostni program.

2. OBDELAVA UKAZNE VRSTICE

Interpreter ukazne vrstice (Command Line Interpreter) ni več sestavni del jedra operacijskega sistema, pač pa omogoča povezavo uporabnika s samim sistemom. V operacijskem sistemu CP/M se omenjeni interpreter imenuje CCP (Console Command Processor), pri Unixu pa je to Shell. Shell prebere ukazno vrstico, ugotovi, za kateri ukaz gre in nato kliče ustrezni program, ki izvede ukaz v skladu s parametri v ostanku ukazne vrstice. Omeniti velja, da so tako kot pri CCP tudi v Shell nekateri ukazi že vgrajeni in jih ni potrebno zaganjati kot ukazne datoteke (podaljšek .COM).

Vsaka ukazna vrstica vsebuje ime ukaza, zastavice za morebitne opcije, argumente k zastavici in ostale ukazne argumente.

2.1 Inicializacija

Ob vstopu v nek program nimamo na voljo nobenih podatkov o ukazni vrstici, natančneje o njenem ostanku (command tail). Začetek vsake ukazne vrstice je vedno ime ukaza - programa. Ta program mora potem, ko se zažene, dobiti vse informacije, potrebne za uspešno izvrštev ukaza. Vse informacije pripravi procedura INIT, ki jo pokličemo z:

INIT (flags_with_arguments)

kjer je flags_with_arguments niz dolžine 20 znakov, v katerem navedemo zastavice (drugo poleg druge) za opcije, ki v tem ukazu zahtevajo tudi argument. Omenjena procedura zgradi množico zastavic (vključno z morebitnimi argumenti), ki jih najde v ukazni vrstici, zgradi pa tudi seznam ostalih argumentov in izvede primerjanje vzorcev (pattern matching), če je to potrebno - v primeru, da ukaz deluje nad več datotekami.

Kot primer si poglejmo začetek programa tail, ki izpiše zadnjih n vrstic (znakov, blokov) datoteke ali pa preskoči prvih n vrstic (znakov, blokov) datoteke in jo nato izpiše do konca. Ukaz tail ima lahko štiri opcije (p, l, b, c), od katerih tri zahtevajo argumente (l-število vrstic, b-število blokov in c-število znakov).

```
begin
  INIT ('lbc') ;
  ...
end.
```

Slika 2.1 Inicializacija branja opcij

2.2 Zastavice za opcije

V različnih programih nastopajo najrazličnejše opcije. Procedura INIT zgradi ustrezne strukture, ki jih lahko pregledujemo z nekaterimi funkcijami in procedurami. Za testiranje prisotnosti neke opcije (zastavice za opcijo) je na voljo funkcija TEST_FLAG, ki vrne vrednost true, če je opcija v ukazni vrstici prisotna, sicer pa false. Za testiranje vrednosti argumenta, ki ga neka opcija zahteva, pa je na voljo funkcija FLAG_VALUE, ki vrne niz dolžine 40 znakov, če je argument prisoten in prazen niz, če argumenta ni (v primeru, da želimo argument pri opciji, ki ga sploh ne zahteva). Pokličemo ju z :

TEST_FLAG (flag) in FLAG_VALUE (flag)

kjer je flag ena od zastavic tipa char. Kot primer si spet vzemimo del programa tail.

```
begin
  ...
  if TEST_FLAG ('b') then Nb := FLAG_VALUE ('b') ;
  Num_Blocks := StrToInt (Nb) ;
  ...
end.
```

Slika 2.2 Testiranje zastavic

Gornji primer preveri, če je prisotna opcija b (število blokov) in v niz Nb vrne število blokov.

2.3 Argumenti

V ukazni vrstici se poleg opcij in njihovih argumentov pojavljajo še razni drugi argumenti kot so npr. poti, imena datotek, itd. Programerju so dostopni preko spremenljivke ARGC, ki pove, koliko je teh argumentov, in funkcije ARGV, ki jo pokličemo z :

ARGV (Arg_No)

Spremenljivka Arg_No je celoštevilska in pomeni zaporedno število argumenta v ukazni vrstici, ki ga vrne funkcija ARGV. Sem seveda štejejo tudi argumenti, ki jih generira proces primerjanja vzorcev, v primeru, da ukaz deluje nad več datotekami, ki jih podamo kot vzorec. Kadar so argumenti datoteke, dostikrat rabimo tudi funkcijo GET_PROT, ki nam vrne ustrezno zaščito datoteke (ki smo jo dobili s funkcijo ARGV) kot celo število. Če gre za lokalni CP/M direktorij, je zaščita -1, sicer pa velja naslednja shema :

Slika 2.3 Zaščita datoteke

Štirje biti (1 nibble) imajo pri tipu datoteke (Type) naslednji pomen) :

bit0 - datoteka je/ni direktorij
bit1,2,3 - nepomembni

Pomen bitov pri zaščiti datoteke (World, Group, User) pa je naslednji :

bit0 Read navadno datoteko lahko beremo, ne moremo pa je izvajati kot program (onemogočen zagon podatkovne datoteke)

bit1 Write navadno datoteko lahko spreminjamo; direktorij lahko spreminjamo (REN, DEL)

bit2 Execute datoteko lahko izvajamo kot program; v direktoriju lahko odpiramo datoteke

bit3 Delete datoteko lahko brišemo

Uporabo zgoraj opisanih funkcij pokaže naslednji del programa, ki izpiše vse argumente in pri tistih, ki so direktoriji, to posebej označi.
Omeniti velja še, da funkcija Argv vrača celoten argument tudi v primeru, da je argument neka pot z vzorcem. Primer :

```
begin
  ...
  if ARGC <> 0 then
    for Arg_No := 1 to ARGC do begin
      Write (ARGV (Arg_No)) ;
      if GET_PROT and $1000 <> 0 then Write ( <DIR>" ) ;
      Writeln ;
    end ;
  ...
end.
```

Slika 2.4 Branje zaščite datotek

/usr/include/*.pas

Argv po primerjanju vzorcev vrača argumente v obliki :

/usr/include/fcl.pas
/usr/include/tail.pas
/usr/include/head.pas
...

V primeru, da smo na enem od CP/M diskov, pa Argv vrača argumente v standardni CP/M obliki disk:ime_datoteke. Običajno potrebujemo le CP/M format imena datoteke, zato obstaja funkcija ARGV_RAW, ki iz celotnega opisa poti izloči le ime datoteke, ki ji spredaj doda logični disk, kateremu je prirejen direktorij, na katerem se datoteka nahaja. Zgornji primer po uporabi funkcije ARGV_RAW nad vsemi argumenti izgleda takole :

F:fcl.pas
F:tail.pas
F:head.pas
...

kjer je F: logični disk, kateremu je prirejen poddirektorij /usr/include.

Za izločanje imena datoteke iz podane poti obstaja še druga funkcija, katere opis bo podan kasneje.

3. DELO Z LOGIČNIMI DISKI

Kot smo že večkrat omenili, moramo direktorije prirediti logičnim diskom, če hočemo delati z njimi oziroma z datotekami na njih. Zato je v knjižnici na voljo nekaj procedur oziroma funkcij, ki omogočajo omenjene akcije. Imena vseh procedur se pričnо z ASG (assign). ASG_FILE je inačica ukaza ASSIGN v programskem jeziku Pascal. ASG_FILE priredi ime neki datoteki. Razlika med ASSIGN v Pascalu in ASG_File je v tem, da prvi zna delati le z imeni datotek v CP/M formatu, slednji pa z opisi celotne poti v drevesni strukturi. ASG_FILE kliče funkcijo ASG_PATH, ki je uporabna zato, ker kot rezultat vrne celo število, ki je številka logičnega diska, kateremu je omenjena funkcija priredila pot, ki smo jo podali kot argument. Ime datoteke iz opisa poti pa izloči funkcija ASG_PATH_EXTRACT, ki vrne celo število, ki povečano za ena pomeni položaj prvega znaka imena datoteke v opisu poti. Omenjene procedure oziroma funkcije kličemo z :

ASG_FILE (f, path)

ASG_PATH (path)

ASG_PATH_EXTRACT (path)

kjer je path niz dolžine 255, f pa neka datoteka, ki ji prirejamo ime. Omenjene funkcije pojasni spodnji primer. Program odpre za branje datoteko primer.pas na poddirektoriju /usr/include in izpiše njeno ime. Obenem izpiše, kateremu logičnemu disku je priredil pot /usr/work.

```
begin
  ...
  path := '/usr/include/primer.pas' ;
  ASG_FILE (f, path) ;
  reset (f) ;
  k := ASG_PATH_EXTRACT (path) ;
  Writeln ('File = ', Copy (path, Succ (k), 255)) ;
  ...
  k := ASG_PATH ('/usr/work') ;
  Writeln ('Logical disk = ', Chr (k + 64)) ;
  ...
end.
```

Slika 3.1 Prirejanje logičnega diska

Program da kot rezultat naslednji izpis :

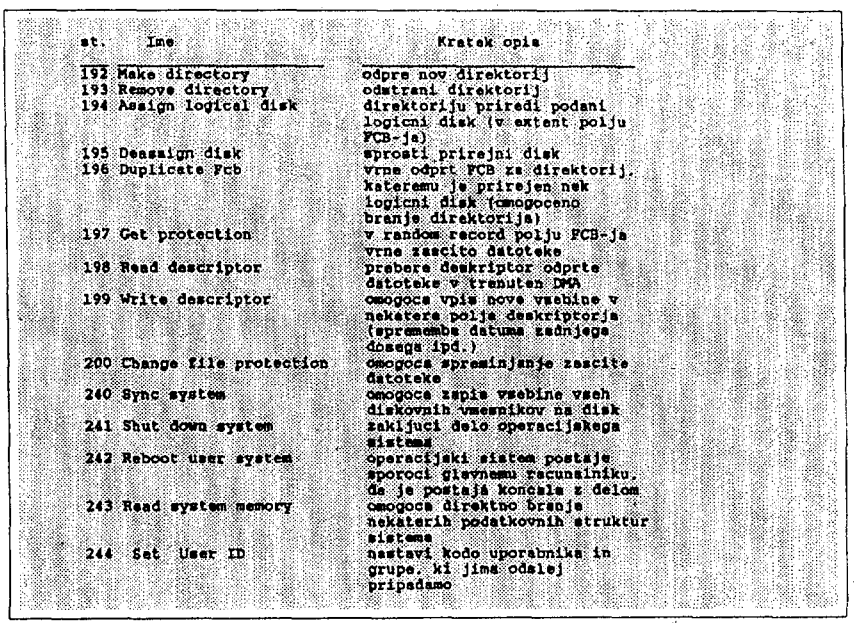
File = primer.pas

Logical disk = F

Uporabniku je na voljo še ena pomembna funkcija, ki mu omogoča klicanje BDOS funkcij (CP/M BDOS in vseh dodatnih), ki delujejo nad datotekami. Funkcijo pokličemo z :

FILE_BDOS (code, f)

kjer je code številka BDOS funkcije, f pa datoteka, nad katero izvajamo omenjeno funkcijo. Funkcija FILE_BDOS tudi sama skrbi za obravnavo napak. Ostale BDOS funkcije kličemo preko standardne procedure BDOS v Turbo Pascalu. S tem imamo pri pisanju uslužnostnih programov poleg viskonivojskega vmesnika na voljo tudi direktne klice funkcij na nižjem nivoju.



št.	Ime	Kratek opis
192	Make directory	odpre nov direktorij
193	Remove directory	odstrani direktorij
194	Assign logical disk	direktoriju priredi podani logicni disk (v extent polju FCB-ja)
195	Deassign disk	sprosti prirejni disk
196	Duplicate Fcb	vrne odprt FCB za direktorij, kateremu je prirejen nek logicni disk (omogoceno branje direktorija)
197	Get protection	v random record polju FCB-ja vrne zascito datoteke
198	Read descriptor	prebere deskriptor odprte datoteke v trenuten DMA
199	Write descriptor	omogoca vpis nove vsebine v nekatere polja deskriptorja (sprememba datuma zadnjega dostopa ipd.)
200	Change file protection	omogoca spreminjanje zascite datoteke
240	Sync system	omogoca zapis vsebine vseh diskovnih vmesnikov na disk
241	Shut down system	zakljuci delo operacijskega sistema
242	Reboot user system	operacijski sistem postaja sporoci glavnemu racunalniku, da je postaja koncala z delom
243	Read system memory	omogoca direktno branje nekaterih podatkovnih struktur sistema
244	Set User ID	nastavi kodo uporabnika in grupe, ki jima odslej pripadamo

Slika 3.2 Dodatne DOS funkcije

Dodatne, tako imenovane DOS funkcije podajamo zato, da zaokrožimo celoto orodij, ki so na voljo programerju pri pisanju sistemskih uslužnostnih programov. Za pravilno uporabo teh funkcij je potreben še dodaten opis nekaterih sistemskih struktur, ki pa jih tu ne bomo navajali.

Omeniti velja, da so nekatere od gornjih funkcij dostopne samo priviligiranemu uporabniku sistema SuperUser, ki edini tudi lahko zaključi delo operacijskega sistema.

Z vsemi zgoraj naštetimi pripomočki smo napisali množico uslužnostnih programov (okrog 45), ki so povsem podobni ukazom operacijskega sistema Unix. Unix-ove ukaze si lahko vsak bralec pogleda v enem od mnogih priročnikov, ki zanj obstajajo. Povejmo še to, da zaradi specifične konfiguracije našega sistema ni bilo možno pri vsakem ukazu implementirati vseh opcij, ki so na voljo v Unix-ovi originalni verziji.

4. Zaključek

Kaj smo z zgoraj opisano rešitvijo dosegli ? Predvsem preprosto, zmogljivo konfiguracijo lokalne mreže in ob majhnih stroških prehod na okolje zahtevnejših in bistveno zmogljivejših operacijskih sistemov z moznostjo preproste nadaljne širitve programske podpore samemu sistemu. Uporabnik, ki že ima mikroračunalnik, mora vanj vgraditi le ploščico, ki mu omogoča povezavo v mrežo in dokupiti nov operacijski sistem z vsemi orodji, ki mu omogočajo širitev. Konfiguracija je zelo primerna za manjše pisarne ali obrate, na njej pa teče vsa aplikacijska programska oprema, ki je bila doslej napisana za njegov stari operacijski sistem CP/M. Uporabnik bo ob uporabi take konfiguracije počasi prešel na nove, zmogljivejše sisteme in ko se bo nekoč znašel pred 32-bitnim strojem z novim operacijskim sistemom, ne bo imel posebnih težav s prehodom nanj.

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) IBM DOS 3.2 Technical Reference Manual, IBM Corp., 1985

(6) IBM DOS 2.0 Manual, IBM Corp. 1983