Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Tržaška 25, Ljubljana

Seminarska naloga

Surov diskovni gonilnik

Character Drivers IV: The Raw Disk Driver [Poglavje 8]

Operacijska programska oprema

Ljubljana, 31. januar 2007

mentor:
prof. dr. Borut Robič

avtorja:
Tine Lesjak,
Borut Lesjak

Kazalo

Opredelitev

Surov disk

Izraz **surov disk** (ang. raw disk) se nanaša na dostop do podatkov na trdem disku ali kakšni podobni napravi za shranjevanje podatkov (npr. mediju) neposredno, to je na **nivoju zloga** (ang. byte), namesto prek **datotečnega sistema** (ang. file system), kot se to počne običajno.

Datotečni sistem ima sicer veliko prednosti pred surovim dostopom (npr. organiziranje in iskanje datotek ali metapodatki), vendar je surov dostop nujen vsaj v naslednjih primerih:

- **povrnitev** izbrisanih datotek (vemo, da se pri brisanju datoteka fizično ne izbriše z diska, marveč se izbriše samo referenca nanjo) in
- ustvarjanje **slike** diska oz. particije (kopira se bit po bit).

Surov dostop do diska je v sistemu Linux preprost. Le "izpišemo" vsebino naprave:

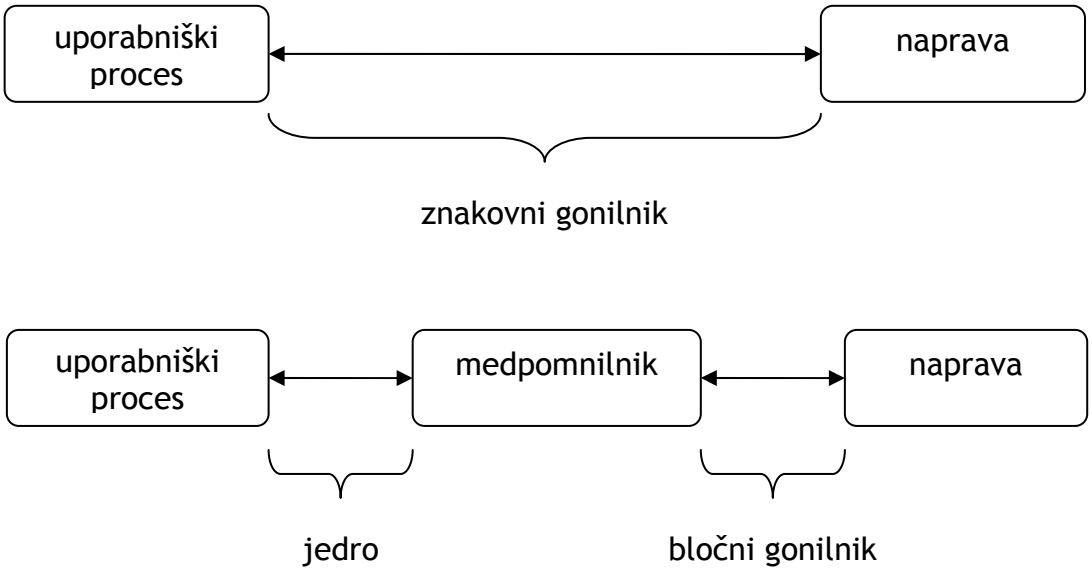
```
cat /dev/hda
```

Surov V/I gonilnik

Poznamo bločne ali medpomnilniške ali sistemske (ang. block/buffer/system) in znakovne ali neposredne ali surove (ang. character/direct/raw) vhodno/izhodne (V/I) gonilnike. Za pomnilniške naprave (npr. diskovne pogone), na katerih se nahajajo datotečni sistemi vrste UNIX, se navadno uporablja bločne gonilnike.

Znakovni gonilniki prenašajo podatke neposredno med uporabniškim procesom in napravo, medtem ko bločni gonilniki vedno prenašajo podatke med medpomnilnikom in napravo. Za prenose med uporabniškim procesom in medpomnilnikom pa je odgovorno jedro (ang. kernel) operacijskega sistema.

Jedro, ki sodeluje samo pri bločnih gonilnikih, prikrije fizične lastnosti naprave, podpira datotečne sisteme in uporabnikom omogoča transparentno delo s podatki.



Slika 1: Razlika med znakovnim in bločnim gonilnikom.

Neposredni dostop

Čeprav je model medpomnilnika in bločnega gonilnika še posebno učinkovita metoda za upravljanje datotečnih sistemov in izvajanje običajnih aplikacij, ki uporabljajo V/I, se lahko izkaže kot manj optimalna, kadar prebiramo ali zapisujemo več (manjših) blokov podatkov zaporedoma. Namesto optimalnejše izrabe V/I sistema medpomnilnik samo vsiljuje dodatno, nepotrebno kopiranje podatkov.

Četudi znakovnega pomnilnika ne moremo uporabljati za dostop do diska prek datotečnega sistema, ga lahko izrabljamo za neposredni dostop do naprave. Pri obsežnih prenosih podatkov (prenosih večjega števila blokov) bi bil to precej hitrejši način kot prenašanje prek medpomnilnika.

Dejansko se to pogosto počne pri UNIX sistemih. Znakovni gonilnik je za diskovne pogone kvečjemu dodan k običajnim bločnim. Ker se znakovni gonilnik izogne medpomnilniku in omogoča uporabniškemu procesu neposreden nadzor nad diskom, se disku, kadar do njega dostopimo prek znakovnega gonilnika, pogosto reče surov (ang. raw).

Kot se bomo v nadaljevanju prepričali, je razvoj znakovnega gonilnika za napravo, za katero že obstaja bločni gonilnik, povsem preprost. Jedro (govorimo o UNIX sistemu) ponuja programske rutine za opravljanje vsega bolj zahtevnega dela, ostalo pa lahko izvedemo prek obstoječe *strategy* rutine bločnega gonilnika.

Navidezni pomnilnik

Večina današnjih (UNIX) sistemov uporablja obliko pomnilniškega upravljanja, imenovano **navidezni pomnilnik**. To pomeni, da jedro rokuje z enoto za upravljanje s pomnilnikom (ang. memory management unit, MMU) na tak način, da uporabniški proces vidi, da ima na voljo več pomnilnika, kot pa ga je dejansko dodeljenega. Vsebina pomnilnika, ki trenutno ni dejansko dodeljena, je shranjena na disku.

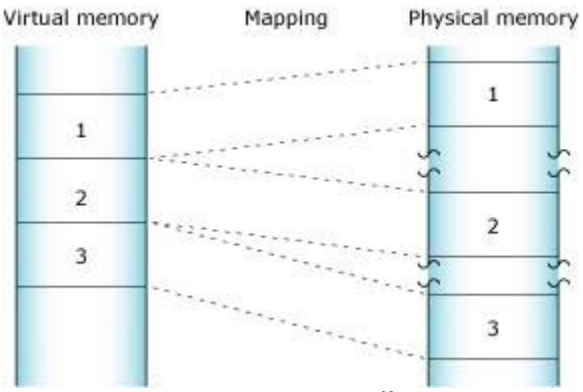
Kadar proces dostopa do pomnilnika, ki dejansko ni dodeljen, se proces začasno ustavi (ang. suspend), dodeli se pomnilniško področje in vsebina se prebere z diska. Proces se nato nadaljuje, kot da bi bil dostopan pomnilnik že ves čas na voljo.

Podobno se zgodi, kadar jedru zmanjka pomnilnika. Takrat jedro pregleda, kateri deli pomnilnika so bili dodeljeni posameznim procesom in ugotovi, kateri deli se lahko prepišejo nazaj na disk in se sprostijo. Tej tehniki se reče **ostranjevanje**.

Težave

Sedaj pa razmislimo, kakšne potrebe ima lahko naprava, ki izvaja neposredni dostop do pomnilnika (ang. direct memory access, DMA). Ko prenaša podatke z diska v računalnik, prejme fizični pomnilniški naslov in tja prepiše podatke z diska neposredno v pomnilnik brez vpletanja MMU enote.

Pojavita se lahko dve težavi. Prva je, da se pri sistemih z navideznim pomnilnikom posameznim procesom ponavadi **ne dodeli sosednjih delov pomnilnika**. Kar proces vidi kot zaporedne logične naslove, je lahko povsem razmetano po vsem fizičnem pomnilniku.



Slika 2: Preslikava med navideznim in fizičnim pomnilnikom (vir: QNX).

Kaj se zgodi, če se pomnilnik, ki ga uporabniški proces dodeli za prebrane podatke, razteza prek meja strani? Naprava bo prenesla podatke v pomnilnik tja, kamor kaže fizični naslov (saj nima pomoči MMU enote) in bo zapisala del podatkov v pomnilnik, ki pripada nekemu drugemu procesu!

Druga težava nastane, kadar se jedro odloči, da pomnilniški blok, kamor naj bi se zapisali prebrani podatki, procesu ni več potreben in ga zapiše na disk ter sprosti. Ta del pomnilnika se **priredi nekemu drugemu procesu**. Ko naprava uporabi fizični naslov, ki ga ima od prejšnjega procesa, se del pomnilnika novega procesa pomotoma prepiše.

Jasno je, da nikoli ne sme priti do teh primerov.

Ti težavi se ne pojavljata v drugih gonilnikih iz dveh razlogov. Prvič, prenosi se vedno odvijajo le v času, ko je uporabniški proces v teku, in drugič, rutini za kopiranje podatkov skrijeta vse težave navideznega pomnilnika pred gonilnikom.

Pri znakovnih gonilnikih, ki uporabljajo DMA dostop, bomo morali uporabiti jedro. Jedru moramo naročiti **posebne priprave**, ki zagotavljajo, da DMA prenosi:

- ne prekoračijo meja strani in
- ne uporabljajo pomnilniških strani, ki niso naložene.

Znakovni gonilnik bomo zgradili na osnovi bločnega.

Splošna zgradba gonilnika

Splošna zgradba vseh surovih diskovnih gonilnikov je enaka. Rutini za branje (ang. read) in pisanje (ang. write) kličeta rutini jedra *physck* in *physio*. Neposredni dostop izvede rutina *strategy*, ki je del bločnega gonilnika.

Rutina *physck* zagotovi, da bo zahteva lahko postrežena s strani same naprave.
Rutina *physio* pa:

- zagotovi, da je naslovljeni pomnilnik naložen,
- prevede zahtevo v bločno zahtevo s pomočjo v ta namen ustvarjene glave medpomnilnika in
- pokliče *strategy* podprogram, kateremu poda glavo medpomnilnika, ki vsebuje nadrobnosti zahteve.

```
rawread(device)
dev_t device;
{
   if (physck(disksize, B_READ))
      physio(strategyaddr, NULL, device, B_READ);
}

rawwrite(device)
dev_t device;
{
   if (physck(disksize, B_WRITE))
      physio(strategyaddr, NULL, device, B_WRITE);
}
```

Spremenljivka *disksize* vsebuje število 512 bajtnih blokov na disku in se uporablja v rutini *physck* za potrditev veljavnosti zahteve. Rutina *physio* prebere parametre zahteve iz posebnih, prej določenih spremenljivk in pripravi glavo medpomnilnika tako, da do medpomnjenja ne bo prišlo.

physio nato pokliče *strategy* podprogram, podan z naslovom *strategyaddr*, ki izvede dejanski V/I dostop. Z vidika *strategy* podprograma izgleda zahteva enaka katerikoli drugi zahtevi jedra za praznjenje ali polnjenje medpomnilnika.

physio izvede tudi celo vrsto pomembnih funkcij poleg tega, da pridobi in nastavi glavo medpomnilnika. V pomnilniku zaklene strani, do katerih se dostopa, da jih jedro ne more vrniti na disk in sprostiti med izvajanjem V/I zahteve. Poleg tega razbije zahtevo na več delov v primeru, če le-ta seže prek meja strani.

Dodatne skrbi pri podprogramu *strategy*

Kadar se *strategy* rutine kličejo le za praznjenje in polnjenje medpomnilnika, se nam ni treba ukvarjati z velikostjo prenosa, saj je vedno nastavljena na en blok (navadno 1 kB).
Ko pa dodamo surov vmesnik bločnemu gonilniku in začnemo klicati *strategy* preko rutine *physio*, se stvari spremenijo. *physio* nastavi velikost prenosa na vrednost, ki je lahko kar velikost strani (ponavadi 4 kB).
Vedeti pa moramo, da čeprav diskovni gonilniki zmorejo obravnavati zahteve poljubne velikosti, nekateri bločni gonilniki ne znajo delati s prenosi, večjimi od 1 kB.

Dodatna skrb je, da se pomnilniški naslov medpomnilnika zdaj nahaja v uporabniškem naslovnem prostoru in ne več v naslovnem prostoru jedra. Posledično mora gonilnik uporabiti druge funkcije, ki znajo dobiti fizični naslov medpomnilnika.

Uporaba

Surov diskovni gonilnik se uporablja pri:

- **vračanju** izbrisanih datotek (mimo datotečnega sistema),
- ustvarjanju **slike** diska oz. particije (kopira se bit po bit),
- **obsežnejšemu** prenašanju podatkov (brez dodatnega kopiranja v medpomnilnik),
- podatkovno kritičnih aplikacijah, kjer morajo biti podatki **takoj zapisani** na disk (v primeru sistemske napake se podatki ne izgubijo) in
- specializiranih aplikacijah, ki imajo **svoje medpomnilniške algoritme** (npr. podatkovne baze).

Koncept surovega gonilnika ni nov. V preteklosti so večkrat poskušali uvesti neposredni dostop na UNIX sistemih. V resnici ga danes podpira le nekaj inačic (npr. IBM AIX). Težava je v tem, da je potrebno imeti dobesedno dvakrat več logičnih naprav v sistemu (bločne in znakovne).

Avtorji sistema Linux so v jedru različice 2.4 vpeljali način, da je lahko več naprav povezanih s poljubno bločno napravo. Predstavili so objekt *kiobuf*. *kiobuf* skrije vse komplikacije v zvezi z navideznim pomnilnikom. Neposredni prenos lahko dosežemo tako, da ustvarimo *kiobuf* in mu podamo fizične strani, ki jih proces uporablja za V/I.

Viri

[1] Pajari G. Writing UNIX Device Drivers. Addison-Wesley Professional. 1991. Poglavje 8, str. 147-153.
[2] Raw Disk Definition. The Linux Information Project. 2006. http://www.bellevuelinux.org/raw_disk.html
[3] Kalev D. Linux Tips and Tricks: Raw Disk I/O. ITworld.com, Accela Communications, Inc. 12. oktober 2001. http://www.itworld.com/nl/lnx_tip/10122001/
[4] UnixWare 7 Documentation: physio(D3oddi). The SCO Group, Inc. 29. september 2004. http://uw714doc.sco.com/en/man/html.D3oddi/physio.D3oddi.html
[5] UnixWare 7 Documentation: buf(D4). The SCO Group, Inc. 29. september 2004. http://uw714doc.sco.com/en/man/html.D4/buf.D4.html
[6] QNX Developer Support: Finding Memory Errors. QNX Software Systems. 31. januar 2007. http://www.qnx.com/developers/docs/6.3.0/ide_en/user_guide/memory.html