Operációs rendszerek (14. rész)

Sorozatunk jelenlegi témája a lemezterület-kezelés, a fájlrendszerek megbízhatósága, egységessége és hatékonysága lesz.

Cikksorozatunk előző részében (Linuxvilág július, 62. oldal) megismerkedtünk a fájlkezelés alapfogalmaival. Most már nyugodtan belevághatunk a megvalósításba is. Mivel a fájlokat mágneslemezen tároljuk, a rendszer fejlesztőinek tulajdonképpen csak a lemez területének kezelésével kell megbirkózniuk, ami – mint hamarosan kiderül – elég nagy kihívás.

A lemezek kezelésével sorozatunk egyik korábbi részében már foglalkoztunk, amikor a blokkos beviteli–kiviteli eszközöket tárgyaltuk. Érdemes az ott leírtakat átismételni, mivel léptennyomon hivatkozni fogunk rájuk.

Minket most nem az eszközök érdekelnek, nem is fontos, hogy milyen lemezről van szó. A fájlkezelés nem más, mint a lemezterület-kezelés tudománya. Az a fontos, hogy milyen elv alapján tároljuk állományainkat a lemezen, úgy, hogy a tartalmukat bármikor visszaolvashassuk, felülírhassuk vagy letörölhessük. Ezenkívül az is nagyon fontos, hogy az egész folyamat gyors legyen.

A lemezterület kezelése sok tekintetben hasonlít a memóriáéhoz, csak még annál is bonyolultabb. Közös bennük, hogy itt is egy meghatározott méretű szabad területtel kell gazdálkodnunk olyan módon, hogy a lehető leggazdaságosabban ki tudjuk használni. Ugyanakkor ebben az esetben felmerül egy olyan tényező is, ami a memóriánál nem volt számottevő. Ez pedig nem más, mint az idő. Ha a lemezről be szeretnénk olvasni egy bájtot, akkor először is az olvasófejet úgy kell mozgatnunk, hogy az pontosan a felett a szektor felett legyen, amelyik a kívánt bájtot is tartalmazza. Egy szektor beolvasásának az ideje nem számottevő, de a fej mozgatása annál inkább. Míg a memória összes részét ugyanannyi idő alatt (gyakorlatilag azonnal) el tudtuk érni, addig a lemezek esetében a végrehajtási idő attól függ, hogy a fej éppen hol tartózkodik. Egy jó fájlrendszernél nem elég tehát, hogy gazdaságos tárolást nyújt, hanem az is fontos, hogy úgy helyezze el az adatokat, hogy például egy fájl eléréséhez a lehető legkevesebbet kelljen mozgatni a fejet.

Lássuk, milyen lehetőségek közül választhatunk! A virtuális memória esetében a szakaszolás és a lapozás volt a két meghatározó módszer, a lemezeknél is használhatunk valami ezekhez hasonlót. Például az állományokat tárolhatjuk folytonosan, azaz egybefüggő lemezterületen. Ez elsőre nagyon csábító lehet, mivel a fájlok olvasása rendkívül egyszerűen és gyorsan megoldható, és a fejet is csak egyszer kell megmozdítanunk.

Rögvest megváltozik a helyzet, amikor egy állomány növekedésnek indul. A memória esetében ilyenkor nem volt semmi gond, a szakaszt csak át kellett egy másik memóriaterületre költöztetni, ahol nagyobb szabad hely állt rendelkezésre. Csakhogy a memóriában történő mozgatás szinte semmi időbe se kerül, míg a lemezekről ez nem mondható el.

Célszerűbb, ha minden állományt egyenlő méretű blokkokra osztunk fel, amelyeknek természetesen nem kell szigorúan egymás mellett lenniük, akár össze-vissza is elhelyezkedhetnek a lemezen. Itt nem okozhat gondot a fájl méretének növekedése, viszont egy fájl teljes tartalmának a beolvasásakor valószínűleg több fejmozgatásra is szükség lesz. Mindenesetre a legtöbb fájlrendszer ezt a sémát használja.

A blokkméret megválasztása

Az odáig rendben van, hogy a fájlokat egyenlő részekre osztjuk, de vajon mekkorákra? Mivel a merevlemezek cilinderekre, sávokra, illetve szektorokra osztottak, megpróbálhatjuk ezeket választani az állományok helyfoglalási egységének.

A cilindert azonban semmiképp sem érdemes választani, mivel túl nagy. A nagy blokkmérettel az a gond, hogy sok kis állomány esetén rengeteg szabad kapacitás megy veszendőbe. Főleg, ha egy olyan rendszerről van szó, amelyik rengeteg apró fájllal dolgozik (például jellemzően ilyenek a Unixok).

Nagy a cilinder? Semmi gond, nézzük a szektort, ami sokkal kisebb. Valójában a szektor nemcsak kicsi, hanem túl kicsi ahhoz, hogy eszményi blokkméretnek választhassuk. Ez ugyan a lehető legkevésbé pazarló megoldás, viszont a nagy állományok rengeteg kis blokkból állnának, s ezek valószínűleg a lemezen egymástól jó távolra helyezkednének el. Egy ilyen állomány beolvasása a sok pozícionálás miatt próbára tenné a felhasználók türelmét.

Egyből két tanulságot is levonhatunk. Az első, hogy helyfoglalási egységet semmiképp sem a lemez „fizikai” egységei alapján érdemes választani. A másik pedig az, hogy minél kisebb a blokkméret, annál jobb a tárolási hatékonyság, viszont a fájlrendszer időhatékonysága annál rosszabb. Ez az állítás fordítva is igaz.

Mekkora lehet vajon a tökéletes blokkméret? Hát valahol a nagyon nagy és a nagyon kicsi között. Ez a nyilván megegyezésen alapuló választás nagymértékben függ a rendszertől, illetve attól, hogy kik használják. A legújabb kutatások mindenesetre azt mutatják, hogy a felhasználók átlagosan egyre nagyobb állományokat birtokolnak, tehát minden jel arra utal, hogy a nagyobb blokkméreteké a jövő.

Szabad blokkok

Amikor új állományt hozunk létre, vagy egy régihez újabb dolgokat írunk hozzá, akkor a rendszernek szabad blokkot kell foglalnia. Ehhez azonban fontos tudnia, hogy az összes blokk közül melyik szabad, és melyik nem.

Erre két elterjedt módszer létezik. Az első esetben a szabad blokkok címeit is blokkokban tároljuk, és ezeket egy láncolt listába fűzzük. Ez az úgynevezett láncolt lista, amit úgy képzelhetünk el, hogy van egy blokk, amelyben csak szabad blokkok címeit találjuk. Az utolsó helyen lévő cím azonban nem egy szabad blokk címe, hanem a következő olyan blokké, amelyik szabad címeket tartalmaz.

A másik az úgynevezett bittérképes módszer. Itt minden blokkhoz egy bitet rendelünk, ami azt jelzi, hogy szabad-e vagy sem.

Mindkét módszernek akadnak hátrányai. Vegyünk egy

1 GB-os merevlemezt 1 KB-os blokkmérettel – ez azt jelenti, hogy körülbelül egymillió blokkot tartalmaz. Láncolt listás megoldáskor, ha a teljes lemez üres, körülbelül négyezer blokkot (4 MB-ot) kell szánnunk az üres blokkok tárolására (32 bites blokkcímek esetében). A bittérképesnél az egész csak egymillió bitet, azaz pontosan 128 KB-ot foglal el.

A bittérképes megoldás jobbnak tűnhet, mint a láncolt listás. Ez azonban csak akkor igaz, ha lehetőség nyílik az egész bittérkép memóriában tárolására. Ma már memória ugyan sok van, de például egy 80 GB-os lemez esetében ez már 10 MB-unkba fog kerülni. Ez már elég nagy ahhoz, hogy ne tartsuk az egész bittérképet a memóriában.

Tegyük fel, hogy a memóriában mindig csak egy blokknyi bittérképet tartunk. Ha szabad blokkra van szükségünk, de a bittérképben nincs ilyen, akkor a rendszernek a bittérkép egy másik szeletét kell beolvasnia. Azt azonban semmi sem garantálja, hogy abban már találunk szabad blokkokat. Ha a merevlemezünk már csaknem tele van, azaz kevés a szabad blokk, megeshet, hogy szinte az egész bittérképet végig kell olvasnunk. A láncolt listás megoldásnál azonban csak egyetlen blokkal kell ezt megtennünk, és máris nyertünk újabb 255 szabad blokkcímet.

Ne felejtsük el azt sem, hogy a láncolt listás megoldásnál a lemez betelítésével csökken azon blokkok száma, amelyet a szabad blokkok nyilvántartására kell használnunk. Az 1 GB-os merevlemez esetében a bittérképnek mindig fent kell tartanunk szabad 128 kilobájtot, míg a láncolt listásnál egyet sem, feltéve, ha csurig töltjük merevlemezünket.

A fájlrendszer hatékonysága

A merevlemezek átlagosan százezerszer lassabbak, mint a memória. Ez az érték attól is függ, hogy a fej éppen hol tartózkodik, illetve mennyi adatot szeretne a felhasználó kiolvasni belőle. A memóriát például bájtonként, egyes rendszerekben szavanként címezhetjük, lemezek esetében azonban mindig az egész blokkot be kell olvasnunk, illetve ki kell írnunk. Ez nem szerencsés akkor, amikor csak egy-két bájtra lenne szükségünk.

Ezért a lemezterület kezelésnél előtérbe kerül a hatékonyság fogalma, azaz ki kell találnunk olyan módszereket, amivel a lemezműveleteket némileg felgyorsíthatnánk. Erre kétféle, ám együttesen használható megközelítés is létezik: egyrészt a lehető legkevesebbszer forduljunk a lemezhez, másrészt, ha már meg kell tennünk, akkor csökkentsük a lemezműveletek idejét. Az első célkitűzést egy gyorsítótár (cache) bevezetésével megvalósíthatjuk. Egy szektor beolvasását ugyan nem tudjuk felgyorsítani, ha viszont úgy rendezzük a blokkokat a lemezen, hogy a fejet ne kelljen minden kérés alkalmával pakolgatni, akkor a második feladatot is teljesíteni tudjuk.

A gyorsítótárnál alkalmazott elv sok tekintetben hasonlít a lapozott memóriáéhoz. Ha egy keresett blokk bent csücsül a gyorsítótárban, akkor a kérést anélkül ki tudjuk szolgálni, hogy a lemezhez fordulnánk. Ha nincs, akkor a gyorsítótárból egy blokkot kiválasztva ki kell írnunk a lemezre, majd betenni a helyére azt, ami éppen kell nekünk. Ehhez előbb ki kell választanunk, hogy melyik legyen az a blokk, amelyiknek távoznia kell. Ennek eldöntésére használhatjuk a lapozási algoritmusok valamelyikét, például az LRU-t vagy a FIFO-t.

A helyzet azonban nem ilyen egyszerű. Ha „színtiszta” lapozási algoritmusokat (például az LRU-t) használunk, akkor könnyen kerülhetünk kellemetlen helyzetbe. Képzeljük el azt a helyzetet, amikor módosítunk egy, a fájlrendszer szempontjából alapvető fontosságú blokkot, például olyat, ami fájlleírót (inode) tartalmaz. A módosítás a lemezen csak akkor könyvelődik el, ha az algoritmus a gyorsítótárból ezt a blokkot takarítja ki. Ha eközben esetleg egy rendszerösszeomlás következik be (vagy áramszünet, vagy hasonló katasztrófa), a fájlrendszerünk sérülni fog.

Ezt orvosolhatjuk akképpen, hogy bizonyos blokkokat (amelyek kiemelt szerepet játszanak a fájlrendszer életében) egyből kiírunk a lemezre, vagy az LRU-lánc elejére teszünk, tehát a következő cserénél az a blokk lesz az, amelyik kiíródik.

Az adatblokkokat csak akkor érdemes cserélnünk, ha várhatóan rövid időn belül nem lesz rájuk szükségünk. Az úgynevezett csonka adatblokkok mindig az LRU-lánc végére kerülnek, mivel ezek hamarosan kellenek, hiszen valószínűleg valamelyik alkalmazás írni fog bele.

Előfordulhat olyan eset, hogy egy adatblokk sosem kerül az LRU-lánc elejére, azaz nem kerül kiírásra. Jellemzően ilyen eset, ha valaki szövegszerkesztővel dolgozik. Például e cikk szerzője e sorok írásakor már harmadik órája szenved a cikkével, és mivel más lemezműveleteket végrehajtó programot nem futtat, a dokumentumot tartalmazó blokk minden bizonnyal nem kerül ki a gyorsítótárból. Ha a szerző egy után megtervezett operációs rendszert használna, és esetleg áramszünet következne be, akkor minden bizonnyal háromórányi munkája veszne el. Ezen még az sem segítene, ha az önműködő mentési szolgáltatás be lenne kapcsolva.

Szerencsére a szerző rendszerében (és minden Unixban) van egy `SYNC` nevű rendszerhívás, ami az összes gyorsítótárban lévő blokkot kiírja a lemezre. Ez a rendszerhívás mindig végrehajtódik, amikor egy lemezegység leválasztására készülünk, de ez még kevés. A `SYNC`-nek akkor van igazán haszna, ha valaki folyamatosan végrehajtja. Ezért az összes Unix induláskor egy folyamatot indít el (amelynek általában `update`, vagy rendszertől függően valami hasonló neve van), ami nem tesz mást, mint hogy félpercenként meghívja a `SYNC`-et. Rendszerösszeomláskor itt is bekövetkezhet némi adatvesztés, de csak annyié, amennyit az utolsó 30 másodpercben alkottunk.

Létezik azonban olyan rendszer is, amelyik minden módosított

blokkot egyből kiír a lemezre (ez az úgynevezett írásáteresztő gyorsítótár). Ilyen például az MS-DOS. Ezért van az, hogy DOS alatt mindenféle komolyabb következmény nélkül cserélgethetjük a hajlékonylemezeinket, míg például Linux alatt először le kell azt választanunk.

Ennek a módszernek az a hátránya, hogyha karakterenként írjuk az állományt, akkor annyi lemezműveletet kell végrehajtani, ahány bájtot kiírunk. Ez sokkal lassabb, mintha először összegyűjtenénk a változásokat a gyorsítótárba, majd egy SYNC rendszerhívás következtében egyetlen lemezművelettel az egészet kiírnánk. (Ezért, akik DOS alá fejlesztettek, gyakran programon belüli gyorsítótárat használtak. Ennek az volt a módja, hogy a kiírandó karaktereket először egy tömbbe tették, és ha az megtelt, akkor az egész tömböt kiíratták, majd indulhatott az egész elölről.)

Meg kell jegyeznünk azonban, hogy az MS-DOS-nak ez nem tervezési hibája, valójában kifejezetten ilyenre alkották. Ugyanis ezt a programot a személyi számítógépekre tervezték, még azokban az időkben, amikor a merevlemez nem volt alapvető kellék, és mindenki cserélhető lemezegységeket használt. A Unix azonban olyan gépeken fejlődött, amik nem éppen hajlékonylemezekről indultak, így ott volt értelme az írási műveletek gyorstárazásának is. Ma már nyilván nem a DOS példája a követendő.

A gyorsítótáron kívül fájlrendszerünk hatékonyságát úgy is növelhetjük, hogy valahogy csökkentjük a fej mozgását. Ezt úgy érhetjük el, hogy azokat a blokkokat, amelyekre nagy valószínűséggel egymás után fognak hivatkozni, a lehető legközelebb helyezzük el egymáshoz (még jobb, ha egy cilinderen is vannak).

Ezt úgy érhetjük el, hogy amikor szabad blokkokat kell foglalnunk, akkor amennyiben lehetséges, mindig egymás utáni blokkokat foglaljunk. Bittérképes nyilvántartás esetén, ha az egész bittérkép a memóriában helyezkedik el, ez egyszerű feladat. Láncolt listásnál sokkalta nehezebb.

A fájlleírókat használó fájlrendszerek esetében azonban egy másik gond is felmerül: még a legkisebb állomány eléréséhez is legalább két blokkot be kell olvasni. Amennyiben a fájlleírók a lemez elején lennének, a fejet általában a lemez közepére kellene pozícionálni. Javíthat a helyzeten, ha a fájlleírókat a lemez közepén helyezzük el, így átlagosan nézve a fej csak feleannyi távot fog befutni.

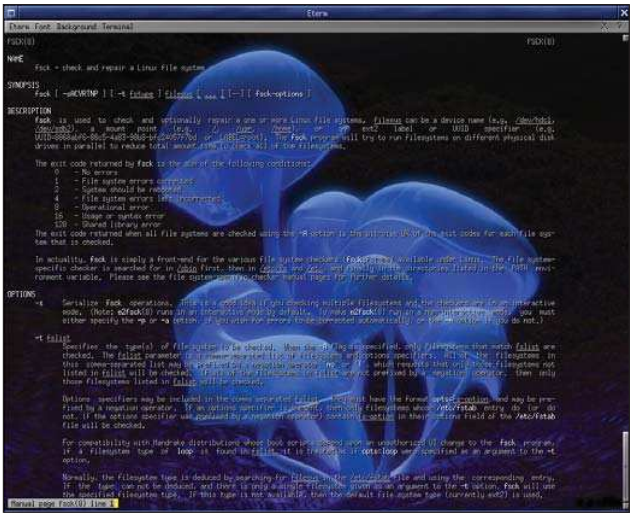
Fájlrendszer-ellenőrzés

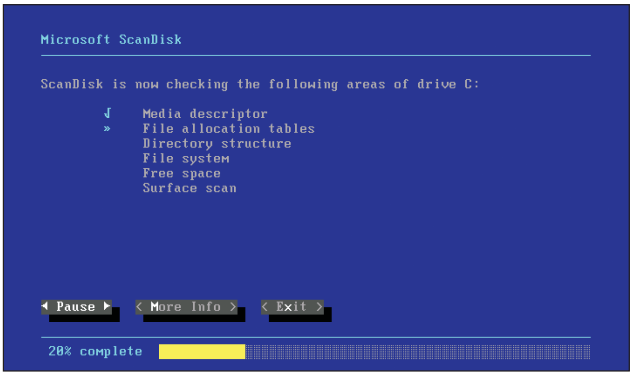
Minden igyekezet ellenére is előfordulhatnak olyan szerencsétlen esetek, amikor úgy hal meg a rendszerünk, hogy még maradt kiíratlan módosított blokk. Ha ez a blokk netán egy fájlleírót vagy egyéb fontos adatot tartalmazott a fájlrendszerről, akkor a fájlrendszer sérül, más néven inkonzisztens állapotba kerül.

Az inkonzisztens fájlrendszer nem biztos, hogy használhatatlan, lehetséges, hogy csak egy-egy blokk válik elérhetetlenné stb. Mindenesetre nem jó dolog, ha a fájlrendszer nem egészséges, ezért érdemes valamiféle inkonzisztencia-ellenőrző és -javító programot futtatni. A továbbiakban most egy ilyen alkalmazás működésének a rejtelmeibe pillantunk bele.

Az inkonzisztencia ellenőrzését végezhetjük blokkonként és fájlonként is. Először nézzük a blokkonkénti ellenőrzést!

Először is minden blokkhoz két számlálót rendelünk: az egyikben azt számoljuk, hogy az adott blokk hány fájlban fordul elő, a másikban pedig, hogy hányszor fordul elő a szabadlistában. Ezután végigmegyünk az összes fájlleírón, ennek alapján az összes blokkot, amelyhez fájl tartozik, elérhetjük. E blokkok számlálóját mindig megnöveljük eggyel. Ha ez kész, akkor jöhet a szabadlista (illetve a bittérkép) végigpásztázása. Ha egy blokk előfordul, akkor annak a másik számlálóját kell növelni. Ezzel a vizsgálat be is fejeződött: fájlrendszerünk akkor és csak akkor lehet összefüggő, ha az összes blokk számlálóinak értéke közül pontosan csak az egyik 1.





Ez logikus, mivel ha mindkét számláló nulla lenne, akkor az olyan, mintha az a blokk nem is létezne. Ebben az esetben hiányzó blokkról beszélünk. Ez nem jelent súlyos veszélyt a fájlrendszerre nézve, de kapacitásveszteséggel jár. A hiba szerencsére könnyedén javítható, csupán fel kell venni a szabadlistába. Az sem túlzottan egészséges, ha egy blokk kétszer szerepel a szabadlistában. Ebben az esetben a megoldás kulcsa az, hogy elölről építjük az egész szabadlistát, ügyelve arra, hogy az adott blokk már csak egyszer kerüljön bele.

Érdekesebb a helyzet akkor, ha azt kapjuk eredményül, hogy egy blokk több fájlhoz is tartozik. Ez nagyon veszélyes lehet a fájlrendszerre nézve, mivel ha letörlünk egy olyan állományt, amelyikben ez a blokk szerepel, akkor a blokk egyszerre lesz szabad is, meg foglalt is. Ha a másik ilyen állományt is letöröljük, akkor a blokk „kétszeresen" lesz szabad. Ezt a hibát ezért mindenképpen javítanunk kell.

A bevált módszer a következő: foglalunk egy új blokkot, ami a sérült blokk tartalmával töltünk fel. Ezután a sérült blokkot kiszedjük a második fájlból, a helyére pedig beteszszük az újonnan létrehozott blokkot. Valamelyik állomány tartalma minden bizonnyal sérülni fog, de nézzük a jó

oldalát: fájlrendszerünk következetessége helyreállt.
A fájlok szerinti összefüggés ellenőrzésekor blokkok helyett az állományokon megyünk végig. Pontosabban belenézünk az összes könyvtárba, és megszámoljuk, hogy egy adott fájlleíróhoz hány állomány tartozik. (Minden fájlleíró tartalmaz egy kapcsolatszámláló nevű változót, ami megmondja, pontosan hány állomány is tartozik hozzá.) Ezután belenéz a fájlleíróba is. Ha ott több, netán kevesebb állomány szerepelne, mint amennyi a könyvtárakon végzett ellenőrzés eredménye, akkor baj van.
Ha a fájlleíró kapcsolatszámlálójának értéke nagyobb, mint amennyit mi számoltunk, az még a jobbik eset. Ha nem javítjuk, annyi történhet, hogy ha az összes olyan állományt kitöröljük, amelyik az adott fájlleíróhoz tartozik, akkor a fájlleíró maga nem szabadul fel, mivel a számláló értéke biztosan nagyobb lesz nullánál. Tehát helypazarlás áll fenn. Mindenesetre a kapcsolatszámláló csökkentésével a gond megoldható.
A legizgalmasabb azonban az az eset, amikor több állományt számoltunk össze, mint amennyiről a fájlleíró „tud". Ilyenkor ugyanis adatvesztéssel is számolnunk kell. Miért is? Tegyük fel, hogy letöröljük az egyik állományt. Ha a fájlleíró számlálója egy volt, akkor most nulla lesz, és felszabadul a fájlleírót tartalmazó blokk. A másik állományt és annak tartalmát soha többé nem fogjuk tudni elérni. Azért a helyzet korántsem ilyen súlyos, egyszerűen csak állítsuk a kapcsolatszámláló értékét annyira, amennyit mi számoltunk.
A mostani ellenőrzőprogramok e két módszert (a blokk és az állomány szerinti ellenőrzést) együtt használják, hatékonysági okból kifolyólag a két folyamat egymástól el sem választható. Ennek köszönhetően a fájlleírót csak egyszer kell átnyálazni.
Ezek a hibák, amelyek veszélyeztethetik a fájlrendszer összefüggőségét, általában mindig a felhasználótól független események miatt következnek be, például rendszerösszeomlás, áramszünet miatt. Igaz, akad egy kivétel: a rendszer helytelen leállítása. De most már tudjuk, mivel járhat egy ilyen cselekedet, így remélhetőleg senki sem fog ilyesmire vetemedni.
Nem számoltunk még azonban a felhasználó által okozott hibákról sem, ilyen például állományok véletlen letörlése, vagy a törlés parancs helytelen használata. Az ilyenek miatt nem lesz a fájlrendszer inkonzisztens, csak a felhasználó adatai tűnhetnek el, de ez elég ok arra, hogy létezzen valamilyen védelmi módszer. Nem minden fájlrendszer tartalmaz ilyet, és ha tartalmaz is, mind-mind különböző módon. Az MS-DOS például nem végezte el a tényleges törlést (azaz a blokkok felszabadítását) egészen addig, amíg volt más szabad hely a lemezen. Addig a felhasználónak lehetősége volt a megfelelő parancs segítségével visszahozni a letörölt állományokat.
A következő részben a biztonsággal foglalkozunk, pontosabban azzal, hogy az operációs rendszerek általában miként védik az adatainkat. Természetesen arról is szót ejtünk, hogy mindezt hogyan valósítják meg.

Garzó András (garzoand@interware.hu)
Körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség. Kedvenc étele a palacsinta, és van egy Richard nevű macskája. Minden észrevételt, megjegyzést, levelet szívesen fogad.