Ismerjük meg a 2.6-os rendszermagot!

Az új ütemező és hangalrendszer mindössze kettő a felfedezendő képességek közül, ami csak ránk vár, amint a 2.5-ös fejlesztői változat végre 2.6-ra vált. Hallgassuk meg, mit gondol erről egy magprogramozó!

A rendszermag hosszú fejlődésen ment keresztül, amióta Linus 2001. november 22-én a 2.4.15 változatból kiemelve létrehozta a 2.5.0-t. Ezután beindult a fejlesztés, aminek eredményeképpen egy teljesen más, jóval többre képes rendszermag jött létre. Ebben a cikkben az érdekesebb, fontosabb képességeket emeljük ki, illetve megnézzük, hogy ezek milyen hatással vannak a Linux teljesítményére és megbízhatóságára.

A 2.5-ös változat története napjainkig

A linuxos hagyományoknak megfelelően a rendszermag kisebbik változatszámából mindig kiolvashatjuk, hogy az adott rendszermag a fejlesztői vagy az üzembiztos sorozatba tartozik-e. Az üzembiztos rendszermagokat páros kis változatszámok jelölik, a fejlesztői változatok pedig páratlan változatszámot kapnak. Amikor már a fejlesztői változat teljesen kiforrott, és megbízhatónak minősítik, kis változatszámát a következő páros számra növelik. Például, a 2.4-es megbízható magsorozat a 2.3-as fejlesztői rendszermagból született.
A jelenlegi fejlesztői mag változatszáma 2.5. A fejlesztői sorozat kezdeti munkái általában igen élénken indulnak, és számos új képesség és fejlesztés kerül bele ilyenkor a rendszermagba. Ha Linus és a magfejlesztők elégedettek az új képességkészlettel, bejelentik a képességbefagyasztást (feature-freeze), ennek célja a fejlesztés ütemének lassítása. Az utolsó képességbefagyasztás 2002. október 31-én volt. Eszményi esetben a képességbefagyasztást követően Linus már nem fogad el újabb képességeket – kizárólag a már meglévő munkarészek bővítését. Amikor a kiválasztott képességek elkészültek és csaknem teljesen stabilak, kihirdetik a kódbefagyasztást (code-freeze). A kódbefagyasztás ideje alatt kizárólag a hibajavításokat fogadják el, hogy a rendszermag végre megérhesse a megbízható kiadást.
Amikor a fejlesztői kiadás elkészül, Linus bejelenti az üzembiztos változatot. A jelenlegi üzembiztos rendszermag nagy valószínűséggel a 2.6.0 változatnevet viseli majd. Bár a hivatalos kiadás időpontját akkor tudjuk meg, „amikor elkészült" a rendszermag, 2003 harmadik vagy negyedik negyede viszonylag jó becslésnek tűnik.
2001 márciusában, majd 2002 júniusában a rendszermag vezető fejlesztői a Kernel Summits keretében találkoztak és vitatták meg a feladatokat. A 2.5-ös elsődleges célja az elöregedő blokkrendszernek (a rendszermag e része felelős a blokkeszközökért, például a merevlemezért) a XXI. századi követelményeknek megfelelő korszerűsítése volt. A további célok közt találjuk a méretezhetőséget, a válaszidő és a virtuális memória (VM) fejlesztését. A rendszermagírók el is érték valamennyi felsorolt – és sok egyéb – célt. Az alábbiakban felsoroltuk a fontosabb új képességeket:

- O(1) ütemező,
- időosztásos rendszermag,
- lappangási fejlesztések,
- újratervezett blokkréteg,
- jobb VM alrendszer,
- jobb száltámogatás,
- új hangréteg.

Ebben a cikkben számos új módszerről és tervezetről fogok majd beszélni, amelyek bekerültek a 2.5-ös rendszermagba, és a 2.6-ban jelennek majd meg. A fejlesztés sok ember kemény munkájának az eredménye. Tartózkodni fogok a nevektől, hiszen ha elkezdenék köszöntet nyilvánítani, elkerülhetetlenül megfeledkeznék néhány emberről, így inkább semmilyen listát sem írok, mintsem hiányos, netán hibás felsorolást adjak közre. Ha kíváncsiak vagyunk rá, hogy melyik rész kinek a műve, a Linux Kernel Mailing List archívuma nagyon jó kiindulási alapot kínál hozzá.

O(1) ütemező

A folyamatütemező (vagy egyszerűen csak ütemező) a rendszermagnak az az alrendszere, amelyik a processzoridő foglalásáért felelős. Ez a rész dönti el, hogy mikor melyik folyamat fog futni. Ez korántsem mindig olyan egyszerű: az ütemezőnek akár egy hosszú listából is ki kell tudnia választani a futtatandó legértékesebb folyamatot. Amikor nagy számú futtatható folyamatunk van, a legjobb folyamat kiválasztása időbe telhet; a többprocesszoros gépek pedig további kihívást jelentenek.
A szükséges módosítások listáján az ütemező igen előkelő helyet foglalt el. A fejlesztőknek három fő céljuk volt, részletesebben:

- Az ütemezőnek teljes körű O(1) ütemezést kell nyújtania. Az ütemező minden algoritmusa állandó idő alatt végezzen, a futó folyamatok számától függetlenül.
- Az ütemezőnek kifogástalan SMP-méretezhetőséggel kell rendelkeznie. Eszményi esetben minden egyes processzornak külön zárolási rendszere és futtatási sora lenne. A futtatási sor a folyamatoknak az a listája, amiből az ütemező választhat.
- Az ütemezőnek jobb SMP-képességekre van szüksége. Képesnek kell lennie csoportosítani folyamatokat az egyik központi egységre, és ott lefuttatni őket. A futtatási sorok hosszának kiegyensúlyozatlanságait feloldandó lehetőséget kell adni, hogy a folyamatok az egyik processzorról a másikra kerülhessenek.

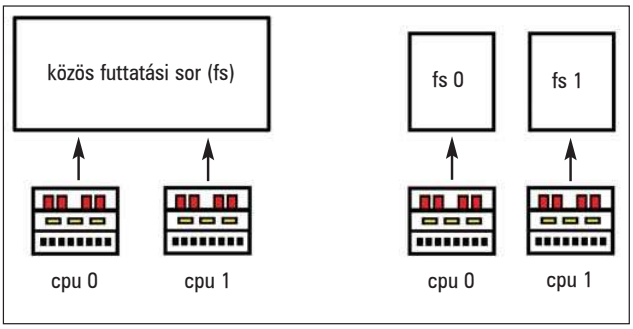
Az új ütemező valamennyi fenti célt megvalósította. Az első cél a teljes O(1) ütemezés volt. Az O(1) olyan algoritmust jelöl, ami állandó (konstans) idő alatt hajtódik végre. A rendszeren futó folyamatok száma – vagy bármilyen más változó – nincs hatással az ütemező egyik részének végrehajtási idejére sem. Képzeljünk el egy algoritmust, amelyik eldönti, hogy melyik folyamat fog következő lépésben futni. Meg kell találunk a legnagyobb prioritású futtatható folyamatot, ami még maradék időszelettel rendelkezik. A korábbi ütemezőben az algoritmus szerkezete a következő volt:

```
for (minden futtathat  folyamatra) {
        keresd a folyamat ørtØkessØgØt
        if (ha ez a legØrtØkesebb folyamat
eddig) {
                emlØkezz rÆ
        }
}
futtasd a legØrtØkesebb folyamatot
```

A fenti algoritmus alapján minden egyes folyamat értékét le kell ellenőriznünk. Ennek megfelelően az algoritmus n elem esetén n-szer ismétel. Így O(n) algoritmusnak nevezzük, hiszen végrehajtási ideje a folyamatok számának növekedésével lineárisan növekszik.
Ezzel szemben az új ütemező a folyamatok számát nézve állandó végrehajtási idejű; azaz teljesen mindegy számára, hogy öt vagy 5000 futtatható folyamat van a rendszeren. Mindig pontosan ugyanannyi időre van szüksége a folyamat kiválasztására és végrehajtására:

- Kérd le a legmagasabb prioritást, amihez folyamatok tartoznak.
- Kérd le az adott prioritási szint listájának az első tagját.
- Futtasd ezt a folyamatot.

Ezzel az algoritmussal megoldhatók a „kérd le a legmagasabb prioritást” és a „kérd le a lista első tagját” műveletek, mivel ezeket az értékeket az ütemező folyamatosan nyilvántartja – keresés helyett egyszerűen csak le kell kérdeznie őket. Eredményképpen az új ütemező a folyamatok végignézése nélkül is képes a legmagasabb prioritású folyamatot kiválasztani.
A második cél a tökéletes SMP-méretezhetőség elérése volt. Ide tartozik, hogy az ütemező teljesítménye egy adott processzoron akkor is azonos marad, ha újabb processzorokat adunk a rendszerhez. A korábbi ütemezőben ez nem így volt: az ütemező teljesítménye a zárolási verseny következtében a folyamatok számának növekedésével egyre csökkent. Az ütemezőnek és adatszerkezeteinek a karbantartása meglehetősen költséges, s ennek legnagyobb részét a futtatási sor kezelése tette ki. Egy zárolásnak nevezett műveletet használunk annak érdekében, hogy egyszerre csak egy processzor tudjon változtatni a futtatási soron. Ez lényegében azt jelenti, hogy az ütemezőt egyszerre csak egy processzor futtathatja.
Ezt a problémát orvosolandó az új ütemező egyetlen közös futtatási sor helyett minden processzorhoz külön sort használ. Ezt a megvalósítást gyakran többsoros ütemezőnek is nevezik. Minden egyes processzor futtatási sorában külön folyamatlista várakozik a végrehajtásra. Amikor egy adott processzor végrehajtja az ütemezőt, csak a saját futtatási sorából választ. Ennek következtében a futtatási sorban sokkal kisebb lesz a versengés, és a teljesítmény a rendszer processzorszámának növekedésével nem csökken. Az *1. ábrán* láthatjuk a közös futtatási sorral és egyedi futtatási sorokkal rendelkező kétprocesszoros gépet.
A harmadik és egyben utolsó cél a fejlettebb SMP-affinitás elérése volt. A korábbi Linux-ütemező rendelkezett egy nemkívánatos jellemzővel, nevezetesen: a folyamatokat pattogtatta a processzorok között. A fejlesztők ezt a viselkedést pingpong-hatásnak nevezték el. Az *1. táblázat* a legrosszabb ilyen esetet mutatja be.
Az új ütemező a processzoronkénti külön futtatási soroknak köszönhetően képes feloldani ezt a nehézséget. Mivel minden processzor saját listával rendelkezik a futtatható folyamatokról, a folyamatok azonos processzoron maradnak. A *2. táblázat* ezt a fejlettebb működési elvet mutatja be. Természetesen néha



1. ábra Bal oldalt a 2.4-es futtatási sora; jobb oldalon a 2.5-ös, illetve a 2.6-os futtatási sora

előfordul, hogy a folyamatokat át kell tenni egyik processzorról a másikra, például ha a processzorokon futtatott folyamatok száma nincs egyensúlyban. Ilyen esetekben a különleges, terhelés-egyensúlyozó módszer kiszedheti a folyamatokat a sorokból. Ez a művelet viszonylag ritkán fordul elő, így az SMP-affinitás megmarad.
Az új ütemező sokkal többre képes, mint gondolnánk. A *3. táblázatban* az új ütemező teljesítménypróbáit adjuk közre.

Időosztásos rendszermag

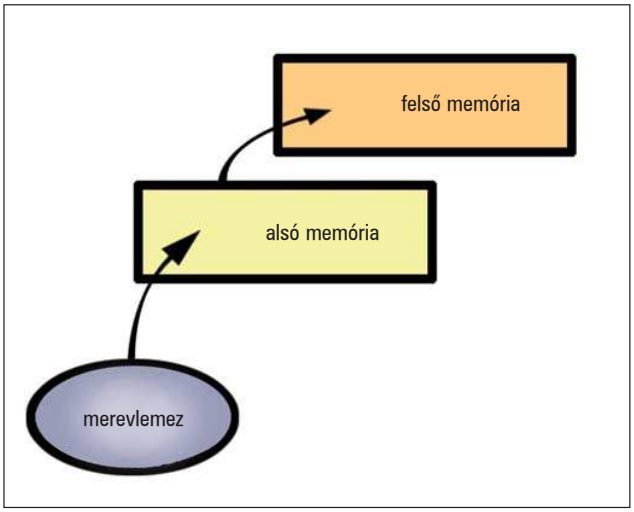
A rendszermag-időosztás feladata a lappangási idő csökkentése. Az eredmény: gyorsabb rendszerválaszidő és kellemesebb interaktív élmény lesz. A Linux-rendszermag a 2.5.4 változattól kezdve vált időosztásossá (preemptív), korábban együttműködő módon hajtódott végre. Ez annyit jelent, hogy a folyamatok (legyenek akár valós idejűek) a rendszermag rendszerhívásaiban nem tudtak más folyamatokat előreengedni.
Ennek eredményeképpen az alacsony prioritású folyamatok a magasabb rendűek elé kerülhetnek, megakadályozva, hogy azok igényeik szerint érjék el a processzort. Mégha az alacsonyabb rendű folyamat időszelete le is járt, tovább folytathatja a futását, amíg a rendszermagon belül be nem fejezte az elkezdett feladatot, vagy önként vissza nem adja az irányítást. Ha a várakozó magasabb rendű folyamat például egy szövegszerkesztő, amibe a felhasználó éppen gépelni szeretne, vagy MP3-lejátszó, ami a hangvermét újra szeretné tölteni, az eredmény gyenge interaktív teljesítmény lesz. Ha viszont a magasabb rendű folyamat valamilyen különleges valós idejű folyamat, az eredmény egyenesen végzetes is lehet.
De miért nem volt már a kezdetektől fogva időosztásos a rendszermag? Azért, mert időosztásos rendszermagot készíteni sokkal több munkát jelent. Ha a rendszermagban futó felada-

1. táblázat **A pingponghatás legrosszabb esete**

CPU	idő 1	idő 2	idő 3	idő 4	idő 5
A folyamat	CPU 0	CPU 1	CPU 0	CPU 1	CPU 0
B folyamat	CPU 1	CPU 0	CPU 1	CPU 0	CPU 1

2. táblázat **Az új ütemező megőrzi a processzorirányultságot**

CPU	idő 1	idő 2	idő 3	idő 4	idő 5
A folyamat	CPU 0	CPU 0	CPU 0	CPU 0	CPU 0
B folyamat	CPU 1	CPU 1	CPU 1	CPU 1	CPU 1



2. ábra 2.4-es rendszermag pattintóveremmel

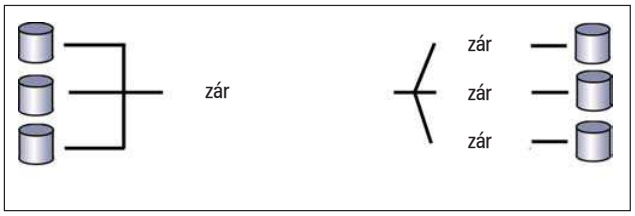
tok bármikor újraidőzíthetőek, védelmet kell beépíteni, hogy megakadályozzuk az osztott adatok egyidejű elérését. Szerencsére az időosztásos rendszermag gondjai éppen azonosak az SMP (symmetrical multiprocessing) esetében korábban felmerült gondokkal. Így aztán azt a módszert, ami eredetileg SMP alatt nyújtott védelmet, a rendszermag időosztási gondjainak megoldásához is könnyedén át lehetett ültetni. A rendszermag egyszerűen az SMP forgózárait (spinlocks) használja az időosztás jelzésére. Amikor a kódban zárolásra van szükség, az időosztás szintén kikapcsolódik. Minden más esetben a jelenlegi folyamat megelőzése teljesen biztonságos.

A lappangási idővel kapcsolatos fejlesztések

Valószínűleg már látjuk is a következő szűk keresztmetszetet. Az időosztásos rendszermag az időzítés lappangási idejét a rendszermag végrehajtási idejéről a forgózár végrehajtási idejére csökkenti. Ez egyértelműen gyorsabb, de még mindig lehetséges hibaforrás. Szerencsére a zárolási idő, vagyis az az időszak, amíg a rendszermag időosztásos módja ki van kapcsolva, csökkenthető.
A rendszermagfejlesztők az alacsony lappangási időt szem előtt tartva hatékonnyá tették a rendszermag algoritmusait. Elsősorban a VM és a VFS (Virtual FileSystem) feladataira összpontosítottak, ennek eredményképpen a zárolás ideje jelentősen csökkent. Végeredményül kitűnő válaszidőt kaptak. A felhasználók a 2.5-ös alatt még közepes gépeken sem tapasztaltak 500 nanosecundumnál rosszabb időzítési lappangási időt.

Újratervezett blokkréteg

A blokkréteg a rendszermagnak az a része, ami a blokkos eszközök kezeléséért felelős. A hagyományos Unix-eszközök kétfajta alkatrészt támogatnak: a karakteres és a blokkos eszközöket. A karakteres eszközök, például a soros kapuk és a billentyűzet, egyesével karakterek vagy bájtok folyamaként értelmezik az adatokat. Ezzel szemben a blokkos eszközök előre megadott méretű adagonként kezelik az adatokat (ezeket nevezzük blokknak). A blokkos eszközök nem csupán fogadják és küldik az adatfolyamokat; bármikor bármelyik blokkjuk elérhető. Az egyik blokkról a másikra történő ugrást keresésnek hívják (seeking). Blokkos eszközre példa a CD-ROM-meghajtó, a merevlemez és a szalagos egység.
A blokkos eszközök kezelése egyáltalán nem magától értetődő. A merevlemezek meglehetősen összetett alkatrészek, amelyek bármelyik érvényes blokkjához az operációs rendszernek írási és olvasási hozzáférést kell biztosítania. Mivel a keresés költséges, az elérési idő mérséklése érdekében az operációs rendszernek okosan kell kezelnie valamint besorolnia a blokkos eszközkéréseket.
A Linux blokkrétegére már nagyon ráfért az újratervezés. Szerencsére a 2.5.1-gyel megkezdődött a réteg kipofozása.
A munkába bevont legérdekesebb részek a blokkos I/O-kérelmeknek megfeleltetett új, rugalmas és általános szerkezet bevezetése, a pattintóvermek (bounce buffers) eltüntetése, és az I/O-folyamatoknak közvetlenül a magas memóriában történő támogatása, valamint a várakozási soronként létrehozott `io_request_lock` és az új I/O-ütemező.
A 2.5-ös változat előtt a blokkos eszköz az I/O-kérelmek leírására a `buffer_head` szerkezetet használta. Ez a módszer több okból sem volt hatékony, de a legkomolyabb érv ezek közül az, hogy a blokkos rétegnek az adatszerkezetet gyakran kisebb darabokra kellett törnie, majd az I/O-ütemezőben később újra össze kellett állítania. A 2.5-ös rendszermag az I/O-műveletek tárolására új adatszerkezetet használ, a bioszerkezetet (bio struct). Ez a szerkezet egyszerűbb, a nyers és átmeneti „tárazott" I/O-műveletekhez egyaránt felhasználható, a magas memóriában is működik, ráadásul könnyedén szétszedhető, illetve összerakható. A blokkos réteg egységesen az új bioszerkezetet használja, így a kód tisztább és hatékonyabb lesz.
A következő feladat a magas memóriában végzett I/O-műveletekhez használt pattintóverem kiiktatása volt. A 2.4-es rendszermagban minden, a blokkos eszközökről a magas memóriába irányuló I/O-átvitel kényszermegállót volt kénytelen beiktatni. A magas memória olyan, nem állandóan belapolt memóriarész, amihez a rendszermagnak különleges támogatást kell nyújtania. Az Intel x86 gépein az összes 1 GB feletti rész ilyen típusú. A magas memóriába irányuló minden I/O-kérelem (például a merevlemezről beolvasni egy fájlt valamely 1 GB feletti címre) kénytelen az alacsony memóriában található különleges pattintóvermet használni. A gyakorlat azt mutatja, hogy néhány eszköz képtelen megérteni a magas memóriacímeket. Ennek megfelelően az eszközöknek mindig alacsony címen kell az I/O-átvitelt elvégezniük. Amennyiben a célterület valójában a magas memóriában található, a blokkos eszköz adatait csak az alacsony memórián keresztül, „pattintva" lehet a magas memóriába juttatni (lásd a *2. ábrát*). Ez a további másolás sok felesleges munkát jelent. A 2.5-ös rendszermag már önműködően támogatja a közvetlen magas memóriaátvitelt, és az erre képes eszközök esetében megszünteti a pattintóverem-logikát.
A következő szűk keresztmetszet, amivel a fejlesztők szembetalálkoztak, a teljes körű I/O-kérelemzár volt. Minden blokkos eszközhöz egy kérelemsor tartozik, ami a blokkok I/O-kérelmeit, azaz az egyes blokkok írását és olvasását jelző egyedi bioszerkezeteket tárolja. A rendszermag – ahogy a meghajtók felveszik vagy törlik a kérelmeket – folyamatosan frissíti a sorokat. Az egyidejű módosítástól a sorokat az `io_request_lock` védelmezi – a kód csak akkor módosíthatja a sort, ha birtokolja a zárat. A 2.5 előtti rendszermagokban egyetlen teljes körű zár oltalmazta a rendszer összes kérelmét. A teljes körű zár az összes sor egyidejű elérését megakadályozta, holott a zárnak pusztán az egyetlen soron belüli egyidejű eléréseket kellene megakadályoznia. A 2.5-ös alatt a teljes körű zárat külön sorokhoz rendelt finom felbontású zárrendszerrel helyettesítették (*3. ábra*). Ennek megfelelően a rendszermag egy időben több sort is képes kezelni.



3. ábra A 2.5-ös rendszermagban megjelenik a kérelemsoronkénti külön zár

4. ábra A fordított térképezés egy fizikai lapot egy vagy több virtuális laphoz rendelhet

Végül az új I/O-ütemező kiküszöbölte a blokkos réteg maradék, kevésbé hatékony megoldását is. Az I/O-ütemező felelős a blokkok összerakásáért és fizikai eszközre történő küldéséért. Mivel a keresés költséges, az I/O-ütemező jobban szeret folyamatos kérelmeket teljesíteni, ezért a beérkezett kérelmeket szektorok szerint rendezi. Ez a képesség a lemezek teljesítménye és élettartama szempontjából egyaránt hasznos. A gond csak az, hogy a folyamatos szektorokra irányuló ismételt I/O-kérelmek megakadályozhatják egy nem ide tartozó szektor kérelmének a kiszolgálását. Az új I/O-ütemező úgy oldja meg ezt a nehézséget, hogy határidőt vezet be az I/O-kérelmekhez. Az I/O-ütemező csak a határidő lejártáig várakoztathatja a kérelmet, azután már mindenképpen ki kell szolgálnia, és nem folytathatja a jelenlegi szektorhoz tartozó kérelmek összeállítását. Az új ütemező egyben az írás–várakozás–olvasás kérdést is megoldja azáltal, hogy az olvasási műveleteknek előnyt ad az írással szemben. Ez a változtatás nagymértékben javítja az olvasási lappangási időt. Végül, de nem utolsó sorban, a kérelemsor mostantól egyszerű lista helyett vörös, illetve fekete fa alakú, ami igen könnyen kereshető adatszerkezet.

Fejlettebb VM alrendszer

A 2.5-ös változat alatt a VM végre magára talált. A VM alrendszer a rendszermagnak az a része, amelyik az összes folyamat virtuális címterületéért felelős. Ide értendő a memóriakezelési módszer, a lapkilakoltatási stratégia (mit cseréljünk le, ha kevés a memória) és a belapolási stratégia (mikor cseréljük vissza a dolgokat). A VM gyakran okozott nehézségeket Linux alatt. A bizonyos terhelés alatt mért jó VM-teljesítmény gyakran okoz gyenge teljesítményt más részeken. Az igazságos, egyszerű, jól felépített VM mindig is elérhetetlennek látszott – egészen mostanáig. Az új VM-ben végzett három nagyobb változtatás eredménye:

- fordított térképezéses (reverse-mapping avagy rmap) VM;
- újratervezett, okosabb, egyszerűbb algoritmusok;
- szorosabb együttműködés a VFS-réteggel.

A végeredményül kapott VM általános esetekben különlegesen jó teljesítményt nyújt, de szélsőséges esetekben sem hullik darabjaira. Lássuk mindhárom változást!
Minden virtuálismemória-rendszer rendelkezik fizikai címekkel (a fizikai RAM-lapkák tényleges lapcímeivel) és virtuális címekkel (az alkalmazások számára nyújtott logikai címekkel). A memóriakezelési egységekből (Memory Management Unit, azaz MMU-ból) álló szerkezetekkel kényelmesen kikereshetjük a virtuális címekhez tartozó fizikai címeket. Ez kívánatos is, mivel a programok folyamatosan használják a virtuális memóriacímeket, és ezt az eszköznek kell fizikai címmé alakítania. Ugyanakkor fordított irányban haladni egyáltalán nem olyan könnyű. Amennyiben fizikai címből szeretnénk virtuális címet létrehozni, a rendszermagnak minden táblabejegyzést végig kell néznie és ki kell keresnie a kívánt címet, de ez meglehetősen időigényes. A fordított térképezéses VM virtuális címekből fizikai címekre mutató térképeket is tartalmaz.

```
for (minden tÆblabejgyzØsre)
      if (ez a fizikai c m sz ksØges)
            megtalÆltuk a megfelelı c met
```

az `rmap` VM egy mutatót követve egyszerűen kikeresheti a virtuális címet. Ez a módszer sokkal gyorsabb, különösen nagyobb VM-terhelés alatt. A *4. ábra* a fordított térképezés diagramját ábrázolja.
Ezenkívül a VM programozói a VM több algoritmusát újratervezték és továbbfejlesztették, szem előtt tartva az egyszerűsítést, az általános esetekben elérendő nagy teljesítményt és a szélsőséges körülmények között szükséges legalább elfogadható teljesítményt. Az eredményül kapott VM egyszerűbb, mégis szívósabb lett.

3. táblázat **A csevegőkiszolgáló-teljesítménypróba nagyszámú folyamat közötti üzenetváltást mér. Az eredmények üzenet/másodperc mértékegységben értendők**

Ütemező	1. futás	2. futás	3. futás
2.4 ütemezője	79723	82210	94803
2.5 ütemezője	612320	620880	609420

4. táblázat **A szálkészítés és kilépés próbaeredménye: a teszt tíz kezdeti szál teljesítményét méri, amelyek mindegyike egy, öt vagy tíz párhuzamos szálat készít és zár be**

pthread könyvtár	1	5	10
LinuxThreads	140 µs	150 µs	170 µs
NGPT	75 µs	80 µs	90 µs
NPTL	20 µs	20 µs	20 µs

Végül sokat fejlődött a VM és a VFS közötti együttműködés. Ez létfontosságú, mivel a két alrendszer igen bensőséges viszonyban áll egymással. Egyszerűsödtek a fájl- és lapvisszaírások, az előreolvasás és a gyorstárkezelés. A `bdflush` rendszermagszálat a `pdflush` szálcsoport váltotta fel. Az új szálak sokkal jobb lemeztelítettség elérésére képesek; egy fejlesztő szerint a kód egy időben hatvan lemezfolyamatot tud telített állapotban tartani.

Szálak fejlesztése

Linux alatt a szálkezelés mindig utólag beleszőtt gondolatnak tűnt. A szálasított modell nem igazán illeszkedik a hagyományos Unix-folyamatmodellbe, ennek megfelelően a Linux-

rendszermagról sem mondható el, hogy elkényeztetné a szálakat. A felhasználói térben futó ***pthread*** programkönyvtár (közismertebb nevén a LinuxThreads), ami a glibc (a GNU C könyvtár) része, nemigen kap nagy segítséget a rendszermagtól, ennek eredményképpen a szálak teljesítménye sem éppen csillagászati. Rengeteg lehetőség kínálkozott a fejlesztésre, de ezt csak úgy lehetett kivitelezni, ha a rendszermag és a glibc programozói együtt tudnak működni.
Örvendezzünk, mert képesek voltak erre, következésképpen a rendszermag száltámogatási képessége jelentős mértékben megnőtt; és Native Posix Threading Library (NPTL) néven létrejött egy új, felhasználói térben használható ***pthread*** könyvtár, ami majd a LinuxThreads helyére léphet. Az NPTL, akárcsak a LinuxThreads, 1:1 arányú szálasító modellt használ. Ez annyit jelent, hogy minden felhasználói térben létező szálhoz pontosan egy rendszermagszál tartozik. Lenyűgöző, hogy a fejlesztők anélkül voltak képesek kitűnő teljesítményt elérni, hogy átálltak volna az M:N modellre (ahol rendszerszálak száma dinamikusan kevesebb is lehet, mint a felhasználói térben futó szálak száma). A magváltozások és az NPTL együtt növekvő teljesítményt és szabályosságot hozott. Egy kis ízelítő az újdonságokból:

- a szálak helyi tárolástámogatása,
- O(1) `exit()` rendszerhívás,
- fejlettebb PID-foglaló rendszer,
- a `clone()` rendszerhívás szálasító bővítése,
- szálérzékeny kódkiíró- (dump) támogatás,
- szálasított jelzés- (signal) fejlesztések,
- új, gyors, felhasználói térben futó zárolási primitívek (ezeket futexnek nevezzük).

Az eredmény önmagáért beszél. Egy adott gépen 2.5-ös rendszermag és az NPTL alkalmazásával százezer szál egyidejű létrehozása és törlése kevesebb mint két másodpercig tart. Ugyanitt a rendszermag-változtatások és az NPTL nélkül ugyanez a próba körülbelül 15 percet vesz igénybe.
A 4. ábrán láthatjuk a szálkészítő és szálkilépő próbateljesítmény eredményeit, az NPTL, NGPT (az IBM M:N arányú Next Generation POSIX Threads nevű ***pthread*** könyvtára) és a LinuxThreads rendszerek felhasználásával. Ez a próba szintén százezer szálat készít, de sokkal kisebb párhuzamos lépésekkel. Ha még ez sem nyűgözött le, akkor igazán keményfejű vagy.

Új hangréteg

A Linux sound architecture (ALSA) régóta várt beillesztése a 2.5.5-ös változatban kezdődött meg. Az ALSA rengeteg fejlesztést tartalmaz az előző hangréteg, az Open Sound Sytem (OSS) rendszerhez képest. Mind közül a legfontosabb, hogy az ALSA sokkal erősebb és gazdagabb API-t nyújt, mint az OSS. Az ALSA-meghajtók és a hozzájuk tartozó programkönyvtár (alsa-lib) segítségével kevesebb erőfeszítéssel tudunk fejlett audioalkalmazásokat készíteni.
Az ALSA rengeteg hangeszközt támogat, továbbá visszafelé együttműködő OSS-csatolófelülete is létezik. Nagyon valószínű azonban, hogy az OSS-meghajtók a 2.6-osban is megmaradnak azok kedvéért, akiknek továbbra is OSS-re van szükségük, esetleg jobban kedvelik.

Jövőkép

Egy kicsit talán felelőtlen dolog a 2.6-os jövőjét boncolgatni, hiszen még meg sem jelent. Ugyanakkor érdekes elgondolkodni azon, milyen is lesz a 2.7-es fejlesztői rendszermag (vagy legalábbis milyet szeretnénk). Ha szerencsénk van, most a régóta várt tty-réteg (terminál) újraírása következik. A tty-réteg mostanra ugyanis hatalmasra duzzadt, és eléggé zavarossá vált.
Mindenki kívánságlistáján szintén az első helyezettek között áll a SCSI-réteg újraírása. Jelenleg a SCSI-réteg túlságosan buta, a meghajtók pedig túl okosak. Sőt az IDE- és SCSI-rétegeket is egyesíteni lehetne, akár egyetlen általános lemezrétegben! De akárhogy is legyen, a SCSI-rétegnek mindenképpen szüksége van egy kis tisztogatásra.
A fentieken kívül minden más elég bizonytalan. Kockázatos lenne találgatásokba bocsátkozni; az eddig felsoroltak is inkább csak a jelenlegi igények egyszerű megfigyelése alapján születtek. Mint mindig, a 2.7-es tényleges munkái is – akárcsak a vakaródzás – attól függnek majd, hogy hol viszket a fejlesztőknek.
A jövőtől függetlenül a 2.6-os rendszermag nagyszerűnek tűnik – kiváló méretezhetőség, fürge munkafelületi válaszidő, fejlettebb igazságosság, illetve boldogan együttműködő VM és VFS-réteg jellemzi.

Linux Journal 2003. május, 109. szám

Robert Love (rml@tech9.net)
Matematika és számítógépes tudományok szakos hallgató a Floridai Egyetemen. Amikor éppen nem Linuxot elemez, autóversenyzik, thai ételeket eszik vagy punk zenét hallgat.