Memóriabütykölés

Hogyan léphetjük át az i386-alapú Linuxok folyamatszám-korlátját?

A folyamatok kezelése az operációs rendszer egyik legfontosabb feladata. Tervezése és megvalósítása nagymértékben befolyásolhatja a teljesítményt. Többfolyamatos operációs rendszerben számos folyamat fut egymással párhuzamosan, így növekedik a processzor kihasználtsága és a teljesítmény. A folyamatok párhuzamos futtatásával számos szolgáltatást biztosíthatunk, és egyszerre több ügyfelet is kiszolgálhatunk – ez a korszerű operációs rendszerek fő feladata.

Az Intel i386-alapú Linux felépítése támogatja a többfolyamatos működést. Megfelelő módszert választva a rendszer válaszideje a folyamatok ütemezése közben is alacsony marad, míg viszonylag nagy teljesítményt nyújt. Sajnos, a 2.2.x változatú rendszermagban egy olyan határ található, mely 4090-re korlátozza az egy időben futtatható folyamatok számát. Ez a szám bőven elegendő egy asztali gép esetében, de kevésnek bizonyulhat egy nagyvállalat kiszolgálóján.

Vegyük példaként egy általános webkiszolgáló működését, mely többfolyamatú/többszálú módszereken alapul. Amikor az ügyfél kérése megérkezik, a webkiszolgáló létrehoz egy új gyermekfolyamatot vagy szálat. Ily módon egy nagyobb terhelésű kiszolgálón könnyen előfordulhat, hogy egyszerre több ezer folyamat fut. Igaz, a nagyvállalati kiszolgálók túlnyomó része nem is Linuxot, hanem Solarist, AIX-ot, HP-UX-ot stb. futtat.

Számos Linux-fejlesztő észlelte ezt a gondot, és megpróbáltak úrrá lenni rajta. A 2.4-es rendszermagban már szerencsére megtalálható a megoldás. Mivel a 2.4-es mag több helyen is változott, sok esetben mégis a korábbi változat mellett maradunk, míg „hozzá nem szokunk” az újdonságokhoz. De hogyan lépjük át ezt a bűvös határt? Lehetséges olyan megoldást találni, mely képes a 2.2.x változat korlátainak áttörésére? Ahhoz, hogy minderre választ adjunk, először meg kell ismernünk a 2.2.x rendszermag folyamatkezelésének működését.

Az Intel i386 szerkezet és a Linux 2.2.x változatának memóriakezelése

A folyamatok kezelése szoros kapcsolatban áll a memóriakezeléssel. Mivel a memóriakezelés megvalósítása a gép felépítése alapján történik, először vessünk egy pillantást az i386 szerkezetére. A korszerű operációs rendszerekben széles körben alkalmazzák a virtuális memóriakezelés módszerét. Ennek köszönhetően a programok több memóriát használhatnak, mint amennyi ténylegesen elérhető. A programok által használt memóriacímek tehát virtuálisak, az elérés során a processzor által biztosított eljárások révén alakulnak át valódi címmé.

Két alapvető memóriakezelési módszer van: a szakaszolás (segmentation) és a lapozás (paging). A szakaszolás során a memóriát nagyszámú részekre osztják, majd részmutatók és eltolások (segment, offset) használatával érik el. Ezt az eljárást olyan korai rendszerekben is használták, mint például a PDP-11.

A lapozás azt jelenti, hogy a memóriát számos, azonos méretű lapra osztják fel, majd a lapokat a memóriakezelés alapegységeiként használják. A memóriaelérés során a laptábla adatai szerint végeznek átalakítást a program által használt cím és a tényleges cím között.

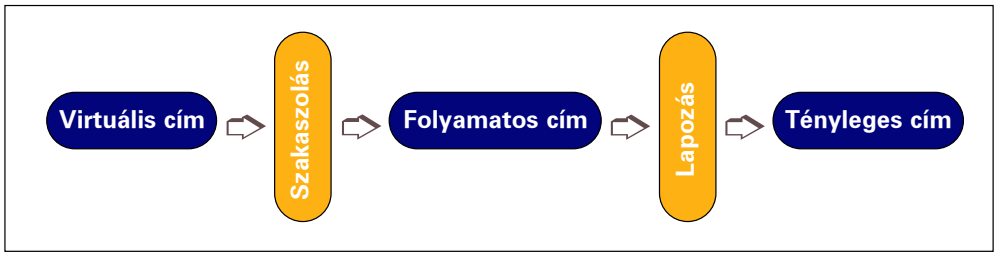
Az i386 szerkezetben használt memóriakezelés szakaszolásos lapozás. A virtuális memóriacím-tartományt két táblázat, az Általános Leírótábla (GDT) és a Helyi Leírótábla (Local Description Table – LDT) segítségével először szakaszokra osztják. Ezt követően a virtuális címet folyamatos címmé (linear address) alakítják. A folyamatos címeket kétszintű táblázathasználattal, a Lapkatalógus (Page Directory Table) és a Laptábla (Page Table) segítségével alakítják tényleges címmé. A virtuális és tényleges cím közötti átalakítás folyamata az *1. ábrán* látható.

Linuxnál a rendszermag a nullás szinten fut. Az Általános Leírótábla beállításával a rendszermag külön címterületre helyezi el programkódját és adatait. Minden egyéb program a hármas szinten fut, adataik és programkódjaik azonos címtartományban találhatók. Külön laptáblák létrehozásával ezek a felhasználói programok védhetők. A 2.2.x változatú Linuxban használt Általános Leírótábla a *2. ábrán* látható. A programok a Helyi Leírótábla (LDT) átírásával oldhatják meg, hogy más adat/kódterületet használjanak.

A 2.2.x rendszermag folyamatkezelése

Egy folyamat egy futó program és a hozzárendelt erőforrások összessége. Ez egy meglehetősen képlékeny megfogalmazás. Sokan a feladatokat (tasks) is egy-egy folyamatnak tekintik. Az egyszerűség kedvéért a továbbiakban mi továbbra is a folyamat elnevezést használjuk. A folyamatkezelés fogalma olyan műveletekkel függ össze, mint a rendszer indítása, folyamatok létrehozása és megszüntetése, ütemezés, folyamatok közötti kapcsolattartás stb. Linux esetében a folyamat valójában adatszerkezetek egy csoportja, mely magába foglalja a folyamat környezetét, az ütemezési adatokat, jelzőket, a folyamatok várakozási sorát, a folyamat azonosítóját, más folyamatokkal fenntartott kapcsolatát stb. Ezt az adategyüttest nevezzük Folyamatvezérlő Tömbnek (Process Control Block – PCB). A futtatás során a PCB a folyamatverem alján található.

A Linux folyamatkezelése nagymértékben támaszkodik a géptípus lehetőségeire. Az előbbiekben az i386 rendszerek szakaszolásos lapozásának alapjait tárgyaltuk csak, de a memóriaszakasz (segment) valójában nem csak egyszerűen a memória egy darabját jelenti. Például a Feladatállapot Szakasz (Task Status Segment – TSS) az i386-alapú rendszer egyik legfontosabb szakasztípusa. Rengeteg olyan adatot tartalmaz, mely a rendszer működéséhez szükséges. Minden folyamatnak van egy, a TR regiszter által mutatott TSS-e. Az i386 előírásai szerint a TR-ben tárolt mutatónak a GDT-ben kell kijelölnie egy leírót. Emel-



1. ábra Virtuális címek átalakítása

lett az LDTR-ben található mutatónak – mely egy folyamat LDT-jét adja meg – a GDT-ben is rendelkeznie kell egy megfelelő bejegyzéssel. Ahhoz, hogy a fenti követelményeknek megfeleljen, a 2.2.x Linux minden lehetséges folyamathoz egy GDT-t rendel. A párhuzamos folyamatok legnagyobb száma a rendszermag indításakor kerül meghatározásra. A rendszermag két GDT-bejegyzést tart fenn minden folyamathoz.

Rendszerindítás

A Linux 2.2.x változata alatt a rendszer indításakor bizonyos folyamatkezeléssel kapcsolatos adatok is betöltésre kerülnek. Ezek közül a legfontosabb a GDT és a folyamatlista.
Amikor a rendszermag elindul, meg kell határoznia a GDT méretét. Mivel minden folyamathoz két bejegyzés tartozik a GDT-ben, a GDT méretét a párhuzamosan futtatható folyamatok legnagyobb száma határozza meg. Linux alatt ez az érték fordítási időben, NR_TASKS néven érhető el. A GDT mérete 10 + 2 (APM-mel) + NR_TASKS * 2.
A folyamatlista valójában PCB-mutatók egy tömbje, melyet a következők szerint adunk meg:

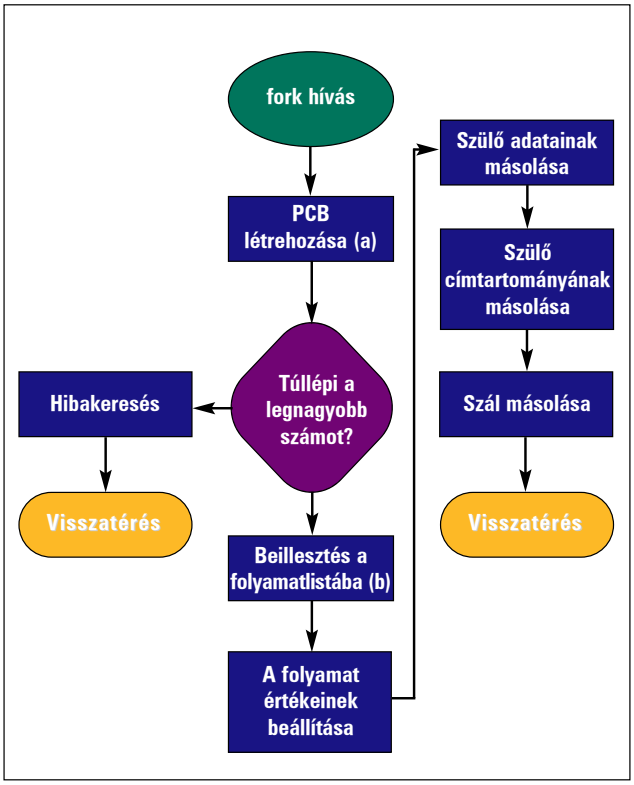
```
Struct task_struct *task[NR_TASKS] =
       {&init_task,};
```

A fenti sorban az init_task a főfolyamat (root process) PCB-je. Miután a folyamatot beillesztettük a folyamatlistába, a folyamatkezelő rendszer is megkezdheti munkáját. Megjegyezzük, hogy a folyamatlista mérete szintén függ az NR_TASKS értékétől.

Folyamatok létrehozása

A Linux 2.2.x változatánál a folyamatokat egy rendszerhívás, a **fork** hozza létre. Az új folyamat az eredeti folyamat gyermeke lesz. Másolat készítésével, klónozással létre tud hozni új szálat, mely tulajdonképpen egy pehelysúlyú folyamat. Mint látjuk, valójában a Linux 2.2.x alatt nincs tényleges „szál”. A *3. ábrán* a fork rendszerhívás működése látható.
A fork legfontosabb lépései a következők:
1. Az új folyamat PCB-jének létrehozása: a rendszermag két lapot lefoglal az új folyamat vermének, majd annak aljára elhelyezi a PCB-t.
2. Az új folyamat beillesztése a folyamatlistába: a rendszermagnak keresnie kell egy üres bejegyzést a folyamatlistában. Ha elértük a párhuzamosan futtatható folyamatok legnagyobb számát, a rendszermag nem talál ilyet, és sikertelenül végződik a rendszerhívás.
3. A szülőfolyamat címtartományának másolása: a gyermekfolyamatnak saját címtartománya van, de először a szülőfolyamattal megosztott címtartományt kap (copy-on-write módszerrel). Az új folyamat LDT-jéhez tartozó GDT-leíró is ilyenkor jön létre.
4. A TSS beállítása az új folyamathoz: az új folyamat TSS-e létrejön a PCB-ben, illetve létrejön a hozzá tartozó GDT leíró is.

2. ábra Az Általános Leírótábla szerkezete



3. ábra A fork rendszerhívás

Ütemezés

Az ütemező eljárás vázlata az alábbiakban látható. Itt azonban csak érintőlegesen foglalkozunk a folyamatváltással. A Linux 2.2.x változatánál a folyamatok közti váltást a switch_to eljárás végzi. A következők szerint működik:
1. Új TSS betöltése a TR beállításával.
2. A régi FS és GS regiszterek mentése a régi PCB-be.
3. Az LDT betöltése, ha az új folyamatnak szüksége van rá.
4. Új laptáblák betöltése az új folyamathoz.
5. Az új folyamat FS és GS értékeinek betöltése.

Megjegyezzük, hogy a TR és LDTR értékét a PCB szolgáltatja.

A legnagyobb folyamatszám túllépése

Mi is a folyamatok legnagyobb számára vonatkozó megszorítás? Az eddigiek alapján már tudjuk is, hogy egyáltalán miért van ilyen megszorítás. A Linux 2.2.x változatában megadott NR_TASKS állandó értékkel, már fordítási időben megszabja az egymással párhuzamosan futtatható folyamatok legnagyobb számát.
Ugyancsak az NR_TASKS, és ugyancsak fordítási időben szabja meg a GDT méretét. Amint az i386 szerkezet is meghatározza, a GDT legnagyobb mérete 8192x8 bájt, azaz összesen 8192 leírót tartalmazhat. 2.2.x változatú Linux esetében a rendszermag indításakor a GDT használata az alábbiak szerint történik:

1. NULL leíró (0 bejegyzés), fenntartott leíró (1., 6., 7. bejegyzés).
2. A rendszermag programkódjának és adatainak leírói (2. és 3. bejegyzés) és a felhasználói programkód és adatok leírói (4. és 5. bejegyzés).
3. APM BIOS leírók (8–11 bejegyzések).

Tehát összesen 12 bejegyzést használunk. Mivel minden folyamatnak két GDT bejegyzésre van szüksége, elméletileg (8192-12)/2=4090 folyamatot futtathatunk egy időben.

1. lista Rendszerindítás

```
# A GDT méretét a lehetõ legnagyobbra (8192) állítja a head.S fájlban.
# A desc.h-ban található GDT-meghatározásba illesszük be az egyes processzorokhoz
# tartozó bejegyzéseket, a többi bejegyzést pedig (az eredeti rendszerhez hasonlóan)
# a folyamatok számára tartja fenn.
CPU0: SHARED_TSS_ENTRY = 12
SHARED_LDT_ENTRY = 13
CPU1: SHARED_TSS_ENTRY + 1 = 14
SHARED_LDT_ENTRY + 1 = 15
...
CPUn: SHARED_TSS_ENTRY + n = SHARED_TSS_ENTRY + n
SHARED_LDT_ENTRY + n = SHARED_LDT_ENTRY + n
( n < NR_CPUS )
# Az NR_TASKS-ot makróról változóra módosítja, és ezt a start_kernel() rendszerindítási
# függvényben végzi:
Task = kmalloc(sizeof(void *) * NR_TASKS, GFP_ATOMIC);
# Ez dinamikusan foglalja le a folyamatlista tömbjét.
# A parse_options függvény módosításával egy nrtasks nevû kiegészítõ értéket is
# használhatunk, mely az egy idõben futó folyamatok legnagyobb számát jelzi.
# Így a felhasználó beállíthatja a folyamatok legnagyobb számát.
```

A nehézségek megszüntetése

Annak ellenére, hogy a GDT méretét a gép korlátozza, találhatunk megoldást a gondra. Egyetlen processzoron adott pillanatban csak egy folyamat futhat. Ennek megfelelően tehát nem kell GDT leírókat fenntartani az összes többi folyamatnak. Amikor egy folyamatot futtatni készülünk, leíróit dinamikusan állítjuk be.
A PCB felépítésének elemzése után megtalálhatjuk benne a TSS-t és az LDT-t – ha vannak egyáltalán. Folyamatváltáskor tehát a PCB mutató alapján találhatjuk meg ezt a két szakaszt, a következők szerint:

```
TSS: proc->tss
LDT: proc->mm->segments
```

2. lista: Megosztott GDT-bejegyzések használata

```
...
#define set_shared_tss_desc(addr,cpu)\
_set_tssldt_desc(gdt_table+
SHARED_TSS_ENTRY+2*cpu,(int)addr,235,0x89);

#define set_shared_ldt_desc(addr,size,cpu)\
_set_tssldt_desc\
       (gdt_table+SHARED_LDT_ENTRY+2*cpu,
       (int)addr,((size<<3)-1),0x82);
...
void __switch_to(task_struct *prev,
       task_struct *next){
...
if(next->tss.tr <= 0x0000ffff)
{
              //az eredeti kód helye
} else {
              set_shared_tss_desc(&next->tss),
                     smp_processor_id());
       set_shared_ldt_desc(&next->mm->segments,
              LDT_ENTRIES,smp_processor_id()); }
// az LDTR és a TR beállítása
...
}
```

Folyamatváltáskor a PCB mutatót valójában a folyamatlistából kereshetjük elő. Mivel mind a TSS, mind az LDT előkereshető, ezeket teljesen felesleges folyamatosan a GDT-ben tartani.
Megoldásunk az, hogy minden processzorhoz csak két GDT-leírót tartunk fenn, közös bejegyzéseket használva minden folyamathoz. Egy kétprocesszoros gépnél például négy bejegyzést kell fenntartani. Amikor az A folyamat fog futni az egyes processzoron, a harmadik és negyedik GDT bejegyzést állítjuk be az A folyamat TSS és LDT leíróihoz. Ezeknek a bejegyzéseknek a régi értékeit elvesztjük.
A fennmaradó GDT-bejegyzéseket ugyanúgy használjuk, mint az eredeti rendszer esetében.

Megjegyzés a megvalósításhoz

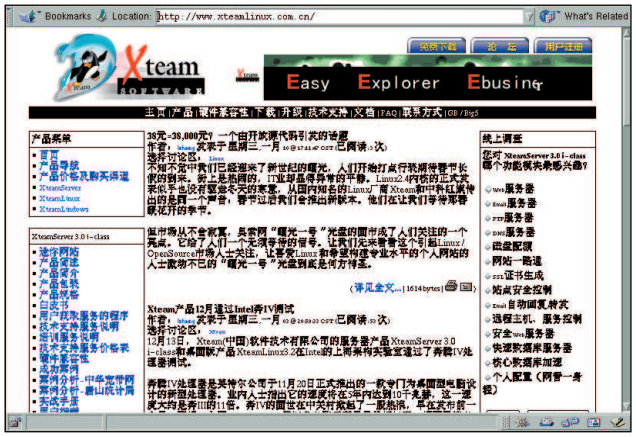
Megoldásunk alapját a folyamat TSS és LDT leírójának dinamikus beállítása képezi. (Lásd az első kódrészletet.)

Folyamatváltás

Az eredeti tervezés szerint a fork rendszerhívás végrehajtásakor a PCB tss.ldt és tss.tr elemeit használjuk az LDTR és a TR mutatóinak mentésére. Az eredeti eljárás szerint egy folyamat LDT-jében a mutató mérete átlépheti a 16 bites határt, így a mutató mentésére a tss.ldt mellett egy külön változót, a tss._ldth-t is használunk. Mivel a Linux 2.2.x változata nem használja a tss._ldth változót, változtatásunk nem fogja érzékenyen érinteni a rendszermagot. Az LDTR és a TR mentése ezután a következők szerint történik:

```
((unsigned long * ) & (p->tss.ldt)) =
       (unsigned long)_LDT(nr);
if (*((unsigned long *) & (p->tss.ldt)) <
       (unsigned long) (8192<< 3)
       set_ldt_desc(nr,ldt, LDT_ENTRIES);
       // az eredeti programkódban itt else {
       // majd nem csinálunk semmit,
       // hagyjuk, hogy a folyamatváltó programkód
       // kezelje az LDT-t és a TSS-t
}
```

A megvalósítás előnyeinek egyike az, hogy az tss.ldt értékének vizsgálatával könnyedén megtudhatjuk, vajon a folyamat sorszáma túllépte-e a 4088-at. Ez fontos lehet a teljesítmény miatt.



Ha a folyamat sorszáma 4088-nál nagyobb, nincs fenntartott bejegyzése a GDT-ben, és a megosztott GDT bejegyzéseket kell használnia. Ezeket a bejegyzéseket a következő kódrészlettel találhatjuk meg:

```
SHARED_TSS_ENTRY + smp_processor_id();
```

A második kódrészlet a megosztott GDT bejegyzések kezelését szemlélteti.
Ha elvégeztük az eddigieket, sikerült átlépnünk a folyamatok számára vonatkozó felső korlátot. A lilo beállítófájlba egy további értéket is elhelyezhetünk, mellyel meghatározhatjuk ezt az értéket. A következő programsor negyvenezerre állítja be a folyamatok legnagyobb számát:

```
Append="nrtasks=40000"
```

Tanulság

Az ismertetett megoldás szerint az egymással párhuzamosan futtatható folyamatok számának felső értékét két gigában határozhatjuk meg – elméletileg. A gyakorlatban azonban a gép és az operációs rendszer továbbra is korlátozzák ezt az értéket. Amikor új folyamat jön létre, a rendszermag a következők szerint foglal le számára memóriát:

```
Folyamatverem (2 lap) + Laptábla (1 lap) +
       Lapkatalógus (1 lap) = 4 lap
```

Ha tehát a számítógépnek 1 GB memóriája van, az operációs rendszer ebből 20 megabájtot használ, és minden folyamat öt lapot kap, a folyamatok legnagyobb száma a következő:

```
(1 GB - 20 MB) / 20 kB = 51404 ~= 50000
```

Gyakorlatiasabb példát nézve, ha minden folyamat legalább 30 kB memóriát használ, az előbbi érték a következők szerint alakul:

```
50000 × 2/3 = 33000
```

Azonban még ez a szám is lényegesen nagyobb, mint a 4090.

➲ http://www.xteamlinux.com.cn

Zhang Yong (leon@xteamlinux.com.cn) az Xteam Software Co., Ltd. vezető programmérnöke. Munkája a Linux számos területét érinti, ilyen a Linux rendszermag fejlesztése, a Linux I18N&I10N, hálózati alkalmazások stb.

Craig Hunt: Linux DNS Server Administration

A tartománynév-szolgáltatás (Domain Name Service, DNS) a hálózatkezelés és az Internet szerves része. A Linux-változatok általában a BIND-et, a Berkeley Internet Name Domain csomagot tartalmazzák, ez végzi a DNS kezelését. A Linuxról szóló könyvek azonban ritkán foglalkoznak a DNS használatával és beállításával. Bár a BIND beállítása nem a legegyszerűbb feladat, mégsem lehetetlen. A Craig Hunt által írt *Linux DNS Server Administration* ismerteti a DNS működését és a beállításához szükséges lépéseket. A szerző, mint ismert TCP/IP- és Linux-szakértő, körültekintően mutatja be a linuxos BIND működését.
A könyv első része a DNS szerkezetével, a használható protokollokkal és a BIND csomaggal foglalkozik. A /etc/hosts fájról szóló rész végre pontosan bemutatja a fájl használatát, előnyeit és hátrányait. A DNS szerkezetét a szerző a tartományokon, a tartománynév-keresésen és a kérelmek feldolgozásán keresztül mutatja be. A DNS-üzeneteket ábrák mutatják be, és arra is fény derül, hogy a DNS-adatbázisokat hogyan lehet összehangolni.
Ezt követi a BIND telepítése és üzemeltetése. A szakasz végén tippeket találhatunk a számunkra leginkább megfelelő DNS-rendszer felállításához.
A második részben a DNS beállítása a téma, s ennek három fejezetet szentelt az író. Az elsőben a resolv.conf, a host.conf és az nsswitch.conf szerepét és működését vesézi ki, a következőben pedig a gyorstárazás (caching) és a másodlagos (slave) kiszolgálók beállítását tárgyalja, s javaslatokat is tesz létrehozásuk módjára. A szakasz záró fejezetében az elsődleges (master) kiszolgáló létrehozásáról olvashatunk (ez felel az adott tartományért és beállítása talán a legösszetettebb). Minden fejezetben találunk példaként szolgáló beállításfájlokat, tartalmukat a szerző minden esetben részletesen elmagyarázza.
A harmadik szakaszban a BIND beállítása kerül sorra. Itt sok érdekességről olvashatunk: hogyan állíthatunk föl nem kiutalt részt-tartományokat egy zónán belül, mikor van szükség kiutalt tartományok létrehozására, illetve miként oszthatunk szét egy hálózati szolgáltatást. Ebben a részben kaptak helyet a DNS „finomhangolására” szolgáló módszerek is, melyek mind-mind a teljesítmény növelésében játszanak fontos szerepet. A dinamikus DNS (DDNS) sem maradhatott ki – ez a protokoll megszabadít bennünket a DNS beállításának nehezétől, hiszen a gépet utasítja arra, hogy a hálózaton elérhető adatok alapján hozza létre az adatbázist.
A könyv befejező részében a szerző egy működő DNS-szolgáltatás karbantartásának lépéseit ismerteti. A könyvhöz négy függelék is tartozik. Az „A” függelék a BIND 9 új lehetőségeit és a Beta 2 kiadás telepítését ismerteti. A „B” függelék a named.conf fájlban használható parancsokat foglalja össze, a „C” függelékben pedig a BIND által támogatott 41 erőforrásrekord kerül bemutatásra. Az utolsó függelékben a *Network Information Service* (NIS) kiszolgáló beállításáról olvashatunk.
Véleményem szerint e könyv viszonylag könnyen érthető, útmutatásai alapján otthoni hálózatomban is könnyedén sikerült egy DNS-kiszolgálót felállítanom. A kiadvány rengeteg ábrája, meghatározása és mintafájlja még könnyebbé teszi munkánkat.

Ralph Krause

Craig Hunt: Linux DNS Server Administration (ISBN: 0782127363)
Beszerezhető a Kiskapu Kft. mintaboltjában:
1081 Budapest Népszínház utca 29. Tel.: (06-1) 303-9119