Az Apache beállítása, trükkjei és hibakeresés

RPM, mod_perl, apachectl, telnet, mod_status, DSO, apxs, Apache::Status. Eléggé ijesztő? Lerner úr most mindent elmesél ezekről, és remélhetőleg megkönnyíti a rendszergazdák dolgát.

A legtöbb régi motoros tisztában van azzal, hogy a Linux, az Apache és más nyílt forráskódú programcsomagok megbízhatóbbak és beállításuk os egyszerűbb, mint üzleti terjesztésű társaiké. Ez azonban egyáltalán nem azt jelenti, hogy az ingyenes programok hibamentesek és mindig minden úgy működik, ahogy beállítottuk. Az igazat megvallva, a nyílt forrású programok néha bosszantóan összetettek: nem mindig világos, hogy mely tulajdonságokat kell megváltoztatnunk, vagy egyáltalán hol kell kezdenünk a munkát.

E hónapban a rendszergazdák által az Apache beállításához szükséges eszközök egy részét tekintjük át. A cikk természetesen nem lehet teljes, hiszen megjósolhatatlan, hogy mi fog elromlani. Ha azonban a hiba okát sikerül feltárnunk, máris jó úton haladunk a megoldás, vagy legalábbis annak kidolgozása felé.

Az RPM-ek eltávolítása

Öt éve használom a Red Hat Linuxot. Akkoriban még a Red Hat egy kis connecticuti cég volt, és nem egy nagyvállalat, melynek nevét még az anyukám is ismeri. Azóta vagyok az RPM (Red Hat Package Manager) rajongója. Még emlékszem azokra az időkre, amikor az Internetről letöltött programok fordításához és telepítéséhez a makefile-okat kellett módosítani. Ám még mindig lenyűgöz az, hogy ma csak letöltöm a program bináris változatát, egyetlen paranccsal telepítem, és ugyanilyen egyszerűen eltávolíthatom a rendszerből. (Úgy hallottam, hogy a Debian csomagtelepítő rendszere ennél is barátságosabb, de ezt még nem volt alkalmam kipróbálni.)

Az RPM-et használó rendszergazdák általában egyszerűen lusták forráskódból telepíteni. Ez nemcsak időigényes művelet, hanem a program hónapokkal, évekkel későbbi eltávolítását is nehézkessé teszi. A hátrányok ellenére érdemes néhány programot forráskódból telepíteni; ezek közé tartozik a Red Hat csomag részét képező Apache is. Maga az Apache tulajdonképpen kis méretű program. A szolgáltatások többségét a fordításkor beépülő modulok valósítják meg. Ha például azt szeretnénk, hogy az Apache automatikusan kijavítsa a félregépelt címeket, a **mod_speling** (nem elírás!) modult kell a fordításkor felhasználnunk. Vagy ha biztos, hogy a kiszolgálón soha nem fut majd CGI program, akkor eltávolíthatjuk a **mod_cgi** modult. És így tovább – az Apache segítségével az igényeinknek tökéletesen megfelelő kiszolgálót alakíthatunk ki. Éppen ezért javaslom, hogy az Apache-t *mindenképpen* forráskódból telepítsük. A folyamat gyors, egy korszerű számítógépen nem tart tovább néhány percnél. A következő lépés a már létező RPM-ek eltávolítása azért, hogy elkerülhessük az RPM-adatbázis összezavarását. Ráadásul így mi is tudni fogjuk, hogy melyik fájl honnan származik.

Ha az Apache-t szeretnénk eltávolítani a Red Hat csomagból: az `rpm -e apache` parancsot kell használnunk. Ha pontosan szeretnénk tudni, hogy mi is zajlik a háttérben, a programot rábírhatjuk, hogy több üzenetet jelenítsen meg. Ehhez az `rpm -evv apache` parancsot adjuk ki. A legtöbb rendszer esetében azonban ez a parancs önmagában nem elég. Az RMP nemcsak a telepített fájlokról, hanem a csomagok egymás közötti kapcsolatáról is nyilvántartást vezet. Mivel a Red Hat-telepítés általában a **mod_perl**-hez és a **mod_php**-hoz is tartalmaz RPM-eket, valószínűleg ezeket is törölnünk kell:

```
rpm -evv apache mod_perl mod_php
```

Ha egy csomag eltávolítása valamilyen kapcsolatot megszakít, az RPM azonnal hibaüzenetet ad és leáll, valamint azt is közli, melyik az a tevékenységünk, mely más csomagok használatát akadályozza meg. Ilyenkor el kell döntenünk, hogy a csomag (például mod_perl) csak egy másik csomaggal (például Apache) együtt használható-e, illetve hogy az eltávolítás nem okoz-e túl sok galibát.

A már létező Apache RPM-ek eltávolítása után elkezdhetjük a fordítást és a telepítést. (Természetesen a fordítás után is törölhetjük az RPM-eket; a lényeg, hogy a lefordított program telepítésekor a fájlok már ne legyenek a rendszerben.)

Töltsük le a legújabb változatot (a cikk írásakor ez az 1.3.12-es) egy helyi tükröoldalról, vagy a ➲ http://www.apache.org/ címről. A csomagot a

```
tar -zxvvf apache_1.3.12.tar.gz
```

paranccsal csomagolhatjuk ki. A `z` kapcsoló a csomagot először a `gunzip` használatával kibontja, a `vv` kapcsoló hatására pedig minden üzenet megjelenik a képernyőn. Mindkét kapcsoló csak a GNU tar programmal működik, mely a legtöbb Linux-változatban is megtalálható.

A forráskódok kicsomagolása után az Apache-t a következő parancsokkal fordíthatjuk le.

```
cd apache_1.3.12
# Belépünk az Apache könyvtárába.
./configure
# Az alapértelmezett beállítások betöltése.
make
# A forráskód lefordítása.
make install
# Az Apache-t a /usr/local/apache könyvtárba
# telepíti.
```

A fenti négy lépés az Apache-hoz kapcsolódó programokat a /usr/local/apache/bin, a naplófájlokat a /usr/local/apache/logs, a HTML fájlokat a /usr/local/apache/htdocs, a CGI programokat pedig a /usr/local/apache/cgi-bin könyvtárba helyezi. A „make install” parancsot rendszergazdaként kell kiadnunk.

Az Apache-t az **apachectl** paranccsal futtathatjuk, ennek alapértelmezés szerinti helye a /usr/local/apache/bin könyvtár. Az **apachectl** egy parancsfájl, mely az Apache indítását és leállítását könnyíti meg, illetve lehetővé teszi, hogy egyszerre csak egy Apache folyamat fusson a gépen. Az Apache indításához a

```
/usr/local/bin/apachectl start
```

parancsot kell kiadnunk. E sort az egyik rendszerindítási parancsállományba (például a /etc/rc.d/rc.local fájlba) illesztve az

Apache a Linux betöltésével egy időben indul el. Ez különösen fontos akkor, ha a már említett módszerrel eltávolítottuk az RPM-eket, hiszen az RPM-változat magától elhelyezi a megfelelő hívást az /etc/rc.d/init.d fájlba. Az **apachectl** segítségével le is állíthatjuk a kiszolgálót:

```
/usr/local/bin/apachectl stop
```

Az Apache indulásakor először a beállításfájljt olvassa be, ez a /usr/local/apache/conf/httpd.conf. Ez a fájl számos parancsot tartalmaz, melyeket egy vagy több érték követ. A kiszolgáló nevét például az alábbi sorral állíthatjuk be:

```
ServerName www.lerner.co.il
```

Minden modul saját parancsokat hoz létre, melyekkel az adott modul tulajdonságait szabályozhatjuk. Például a **mod_userdir** modul a UserDir parancsot teszi elérhetővé, mellyel a felhasználói könyvtárakon belül a honlap fájljai számára fenntartott könyvtárat határozhatjuk meg. De mi van akkor, ha a mod_userdir modult nem telepítettük? Ilyenkor az Apache nem tud mit kezdeni a UserDir paranccsal, s ezért hibaüzenettel leáll. A megoldás az `apache configtest`, mely ellenőrzi, hogy a httpd.conf fájlban megadott parancsok és értékek értelmezhetők-e. Ha minden utasítás ismert és az értékekkel sincs gond, akkor az „OK” üzenetet kapjuk.
Egy másik megoldás, hogy a bizonytalan kimenetelű parancsokat két <IfModule> tag közé zárjuk. Ez a rész csak akkor kerül végrehajtásra, ha a megadott modul be lett töltve. A UserDir parancs tehát mindig szerepelhet a fájlban, de csak ha szépen „becsomagoljuk”:

```
<IfModule mod_userdir.c>
       UserDir public_html
</IfModule>
```

Az <IfModule> rugalmassága különösen a DSO-ként lefordított modulok használatakor jön jól.

A Telnet használata

Az Apache-kiszolgáló lefordítása, telepítése, beállítása és elindítása után még mindig adódhatnak gondok. Az `apachectl configtest` paranccsal ellenőrizhetjük a beállításfájl parancsainak és értékeinek érvényességét, de ez még nem jelenti azt, hogy a parancsok pontosan azt teszik, amit mi szeretnénk.
A kiszolgáló működésének ellenőrzésére a telnet a legalkalmasabb. Ezt a programot minden Linux-felhasználó ismeri: segítségével egy másik gépre jelentkezhetünk be. A telnet azonban bármelyik TCP-kapun csatlakozhat a hívott számítógéphez, nem csak a 23-ason. Használhatjuk tehát a 25-ös (SMTP), a 110-es (POP) és akár a 80-as (HTTP) kapukat is. Ez a módszer tökéletesen alkalmazható nemcsak annak ellenőrzésére, hogy a webkiszolgáló működik-e, hanem alapvető ellenőrzésekre is.
A módszer használatához azt kell megértenünk, hogy minden TCP/IP szolgáltatáshoz egy-egy kapu tartozik, a helyi és a távoli gépen egyaránt. Ezért a két gép közötti telnet kapcsolathoz két IP-címre, a helyi gép egy tetszőleges kapujára és a távoli gép 23-as kapujára lesz szükségünk. Hasonlóképpen, egy levél továbbításához a helyi gép egy tetszőleges kapujáról a kiszolgáló 25-ös kapujára kell csatlakoznunk. A leggyakrabban használt szolgáltatások kapuszámát (beleértve azokét is, melyeket nem célszerű megváltoztatnunk: FTP, SMTP TELNET) a /etc/services fájlban találjuk meg. Ha a **telnet**-tel kívánunk egy gép valamelyik kapujára csatlakozni, akkor egyszerűen adjuk meg a kapuszámot is (vagy a nevét, amennyiben az szerepel a /etc/services fájlban). Ha például a www.lerner.co.il gép 80-as kapun elérhető HTTP-kiszolgálójára kívánunk csatlakozni, akkor a

```
telnet www.lerner.co.il 80
```

parancsot kell begépelnünk. Ha a kiszolgáló más kaput használ (ezeket az Apache Listen és Port parancsaival állíthatjuk be), akkor természetesen azt a számot kell megadnunk. Ha például a kiszolgáló a 8080-as kapun érhető el, akkor a

```
telnet www.lerner.co.il 8080
```

paranccsal kapcsolódhatunk.
Az Apache képes egyszerre több kapun is kérelmeket fogadni, ha azokat helyesen beállítjuk a httpd.conf fájlban.
Ha a megadott kaput semmilyen szolgáltatás nem használja, akkor a „connection refused” üzenetet kapjuk. Ilyenkor kukkantsunk bele az Apache naplófájljába (alapértelmezés szerint ez a /usr/local/apache/logs/error_log) a hiba felderítéséhez.
Ha a megadott kaput valóban egy HTTP-kiszolgáló használja, előbb az a karakter jelenik meg, mellyel visszatérhetünk a telnet parancssorához (általában CONTROL), majd létrejön a kapcsolat. Amit a bejelentkezés után látunk, az a hívott kiszolgáló típusától függ. Míg az SMTP, az FTP és a POP a kiszolgáló nevének kiírásával üdvözli a felhasználót, a HTTP általában nem jelenít meg semmit. Feltételezi, hogy tudjuk a kiszolgáló nevét (ha már egyszer sikerült kapcsolódnunk hozzá...), és azt is, hogy a kapcsolat sikeres volt.
Most elkezdhetjük beírni a HTTP-parancsokat. A legegyszerűbb lekérdezés a `GET /`, mely után természetesen ENTER-t kell nyomnunk. Ezt a formát a kompatibilitás a legtöbb korszerű kiszolgáló támogatja, de már nem használatos. Nem sokkal az ENTER leütése után a honlap kezdőlapjának tartalmát láthatjuk. A formázatlan HTML-fájl olvasása először kissé nehézkes, de ne felejtsük el, hogy most csupán egy egyszerű hibakereső eljárásról beszélünk.
Összetettebb lekérdezéseket is intézhetünk a kiszolgálóhoz a HTTP/1.0 használatával. Ez a HTTP első olyan változata, melynél a lekérdezésekben és válaszokban fejléc is található. A kérelmet és a választ egy vagy több fejlécsor vezeti be. A fejléc a nevét, egy kettőspontot és egy karakterláncot tartalmaz. A fejléc és a törzs között egy üres sor található.
A fenti egyszerű lekérdezést a `GET / HTTP/1.0` alakban is megadhattuk volna, ezzel jelezve, hogy az ügyfél a HTTP 1.0-s változatát is megérti. Ezután kétszer kell ENTER-t nyomnunk – az első a sor végét jelöli, a másik pedig azt, hogy a parancssor és a HTTP-kérelem között nem akarunk fejlécet elküldeni.
A telnet segítségével név–érték párokat is küldhetünk a kiszolgálónak, a két elem közé egyenlőségjelet, a párok elé pedig & jelet kell tennünk. Általában ezeket az értékeket egy GET kérelemmel adjuk át egy CGI vagy más, dinamikus tartalmat létrehozni képes programnak. Például:

```
GET /cgi-bin/foo.pl?nev1=ertek1&nev2=ertek2 HTTP/1.0
```

Az ENTER kétszeri leütése után a válasz fejlécét, majd a **foo.pl** nevű CGI program kimenetét láthatjuk.
A kérelemmel együtt fejléceket is továbbíthatunk. Az első GET sor után az ENTER-t egyszer lenyomva írjunk be egy vagy több fejlécet, és ne felejtsünk el mindegyik után egy sort kihagyni. Például:

```
GET /cgi-bin/foo.pl?nev1=ertek1&nev2=ertek2 HTTP/1.0
Accepts: text/html
Accept-language: text/html
```

Az utolsó fejlécsor után nyomjunk kétszer ENTER-t – egyszer a fejléc, másodszor pedig a kérelem befejezéséhez.

Hány kiszolgálót futtassunk?

Még a kevésbé látogatott weboldalak üzemeltetői is több Apache-folyamatot futtatnak egyszerre. A régebbi kiszolgálók általában addig várnak, amíg egy új kapcsolat szükségessé teszi az új folyamat indítását. Az Apache készítői nem javasolják ezt a módszert: az Apache már indulásakor több alfolyamatot is elindít.

Egy alfolyamat egyszerre csak egy HTTP-kapcsolatot kezel, ez azt jelenti, hogy a futó Apache-folyamatok számának a látogatók aktuális számával kell megegyeznie. A határértéket a MaxClients paranccsal állíthatjuk be. Ennek alapértéke 150. Ha a **MaxClientst** túl alacsonyra állítjuk, akkor az újonnan kapcsolódó látogatóknak várniuk kell, míg valamelyik alfolyamat felszabadul.

Az Apache a httpd.conf fájlban megadott elvek alapján állandóan változtatja a kiszolgálók számát, a bejövő kérelmeknek megfelelően. A **MinSpareServers** és a **MaxSpareServers** parancsokkal szabályozhatjuk, hogy hány tartalék kiszolgáló fusson állandóan a háttérben, melyek feladata a friss kérelmek azonnali kiszolgálása. Ha a szabad tartalékok száma a MinSpareServersben megadott szintre csökken, az Apache több új kiszolgálót indít. Másrészről, ha a szabad folyamatok száma eléri vagy meghaladja a MaxSpareServers paranccsal meghatározott szintet, az Apache azonnal leállítja a feleslegesen futó folyamatokat.

Ha a kiszolgáló elindul és válaszolni is képes, de a kapcsolódás túl sok időt vesz igénybe, akkor valószínűleg a fenti értékekkel lesz a gond. Ilyenkor növeljük a MaxSpareServers vagy a MaxClients értékét azért, hogy minél kevesebb felhasználónak kelljen várnia a kapcsolódásra.

Természetesen az új folyamatok indítása nagyon leterheli a számítógépet – a processzor kevesebb időt tud szánni egy-egy folyamatra, és a memória is vészesen fogy. A **mod_perl** különösen sok memóriát fogyaszt, tehát lehetőleg ne indítsunk túl sok olyan Apache-folyamatot, mely a mod_perl modult is használja. A Linux **free** parancsával bármikor ellenőrizhetjük az elérhető fizikai és virtuális memóriát, a **top** paranccsal pedig az egyes folyamatok által elfogyasztott processzoridőt és egységadatokat jeleníthetünk meg.

Mivel a webkiszolgálóknak a lehető leggyorsabban válaszolniuk kell a beérkező kérelmekre, és mivel a virtuális memória jóval lassabb a fizikai RAM-nál, ezért fordítsunk különös gondot a virtuális memória fogyasztására, korlátozzuk a használatát.

Ha a honlap és az adatbázis (például MySQL, vagy PostgreSQL) ugyanazon a gépen található, komolyabb gondok is előfordulhatnak. A dinamikusan létrehozott oldalak látogatottságának növekedésével természetesen egyre több Apache-folyamat fut a gépen. De a látogatók kiszolgálása céljából az adatbázist kezelő kapcsolatok számának is növekednie kell. Egy ponton a honlap saját népszerűségének áldozatává válik, hiszen az Apache és az adatbázis egyre véresebb küzdelmet folytat a rendszererőforrásokért. A nagy forgalmú, adatbázist is használó honlapok esetében érdemes tehát a két feladatot különválasztani: a webkiszolgáló külön gépéhez egy vagy több, az adatbázisok kezeléséért felelős gép csatlakozzon.

A mod_status

Az Apache állapotáról a **mod_status** modullal készíthetünk pillanatfelvételt. A mod_status a HTTP-kiszolgálók pillanatnyi állapotát írja le (új kapcsolatra vár, kérelmet olvas be, kezeli a kérelmet, a választ állítja össze stb.)

A mod_status alapértelmezés szerint az Apache része, tehát indításához mindössze a megfelelő parancsokat kell megadnunk, majd az alapértelmezett kérelemkezelő (handler) eljárást kell „server-status”-ra állítanunk. Ha ezután olyan URL érkezik, melynek kezelője „server-status”, az Apache állapotjelentést készít, a kérelem többi részét figyelmen kívül hagyva.

Ezért a mod_statust általában csak egyetlen URL-lel használjuk. Például létrehozunk egy „/server-status” nevű URL-t a kiszolgálón, ezt megtekintve a látogató a kiszolgáló állapotáról tájékozódhat. Érdemes mindig a teljes állapotjelentést megjeleníteni (ExtendedStatus On). Nézzünk egy egyszerű beállítást:

```
<Location /server-status>
       SetHandler server-status
</Location>
ExtendedStatus On
```

E négy sort illesszük a httpd.conf fájlba, majd indítsuk újra az Apache-t (vagy küldjünk neki HUP jelet). Ezután a /server-status URL-t megtekintve az alábbihoz hasonló üzenetet kapunk:

```
Server Version: Apache/1.3.12 (UNIX) mod_perl/1.24
Server Built: Mar 29 2000 12:25:42

Current Time: Friday, 21-Jul-2000 16:02:51 IDT
Restart Time: Friday, 21-Jul-2000 16:02:48 IDT
Parent Server Generation: 2
Server Uptime: 3 seconds
Total accesses: 0 - Total traffic: 0 kB
CPU Usage: u0 s0 cu0 cs0
0 requests/sec - 0 B/second -
1 requests currently being processed, 4 idle servers
```

Az állapotadat a kiszolgáló indításának időpontját, a kapcsolatok számát és forgalmát tartalmazza. Azt is megjeleníti, hogy mennyi bájtnyi adatot szolgál ki éppen ez a folyamat, illetve hány folyamat várakozik feladat nélkül. A mod_status segítségével betekintést nyerhetünk az Apache-kiszolgáló működésébe, és azt is kideríthetjük, hogy megfelelően állítottuk-e be a MaxSpareServers értékét.

A mod_status ezután a következő alakban jelenít meg adatokat (igen, első ránézésre elég titokzatos):

```
W_____.....................................................
.....................................................
.....................................................
.....................................................
```

Minden „.” karakter egy feladat nélkül várakozó Apache-folyamatot jelöl. Az új kapcsolatra váró folyamatok jele „_”; a kérelmet beolvasóké „R”, a választ küldőké „W”. A jelek természetesen állandóan változnak, hiszen az Apache-folyamatok állapota sem állandó. Ezt követően az egyes működő folyamatok állapotáról tájékozódhatunk. Láthatjuk, hogy mely kapcsolatok feldolgozása tart sokáig, melyek a legnépszerűbb kapcsolatok a hónapban, s még ezernyi tényt. Természetesen nem túl bölcs dolog az állapotadatokat az egész világ elé tárnunk. Szerencsére az „Order”, „Restrict” és „Deny” parancsokkal a hozzáférést letilthatjuk, illetve elérhetővé tehetjük egy adott tartomány számára. Például:

```
<Location /server-status>
       SetHandler server-status
       Order deny,allow
       Deny from all
       Allow from .lerner.co.il
</Location>
```

A fenti beállítások hatására a mod_status kimenete csak a lerner.co.il tartomány gépei számára lesz elérhető. A más tartományokból érkező kérelmekre a kiszolgáló az „Access forbidden” (Hozzáférés megtagadva) üzenettel válaszol.

Tervezés a DSO-val

Ez idáig feltételeztük, hogy az Apache-t mindenki statikusan fordította, vagyis a modulok fordításkor bekerültek a programba. Ez az Apache fordításának hagyományos módja, és ez az alapértelmezés is, ha a `./configure` parancsot használjuk.

A fenti beállítás egyetlen hibája, hogy a rendszer így nem túl rugalmas. Mi történik például akkor, ha hónapokkal később kiderül: egy fontos modult elfelejtettünk a fordításkor megadni? Ilyenkor az egész csomagot újra kell fordítanunk, most már figyelmesebben meghatározva a programba kerülő modulokat. Ez a módszer első ránézésre nem tűnik túl hátrányosnak – elvégre nem kell mindennap új modulokat a programba fordítanunk.

Azonban a baj ennél sokkal mélyebben gyökerezik. Először is: miért foglalnánk le a nem használt modulok számára memóriát? Másodszor: miért kellene egy-egy új modul vagy modulváltozat megjelenésekor az egész csomagot újrafordítanunk?

A feladatot az Apache Dynamic Shared Objects (DSO) felhasználásával oldhatjuk meg. Így az Apache fordításához mindössze két modulra lesz szükségünk: az egyik a **mod_core** (ez az alapvető szolgáltatásokat tartalmazza), a másik pedig a **mod_so** (ezzel tölthetjük be a DSO-kat). A többi modult csak szükség esetén kell betöltenünk. Mivel így a modulok a programon kívül helyezkednek el, ezért frissítésük is egyszerűbben, a program újrafordítása nélkül elvégezhető. És éppen ez az, ami a folyamatos működést igénylő, forgalmas webkiszolgáló esetén rendkívül előnyös.

Ha az Apache-t DSO-k használatával fordítjuk, el kell döntetnünk, hogy mely modulokat kívánjuk a program részévé tenni, és melyeket DSO-ként használni. Javaslom, hogy mindent DSO-ként készítsünk el, kivéve az Apache által alapértelmezés szerint telepített modulokat. Ehhez a configure parancsfájlt más módon kell meghívunk:

```
./configure --enable-shared=max
```

Így a **mod_so** is a program része lesz, az Apache pedig az összes alapértelmezés szerinti modullal lesz lefordítva. Miután a `make` paranccsal lefordítottuk, a `make install` segítségével telepíthetjük az Apache-t, mely a szokásos módon működik ezután is. Az egyetlen észrevehető különbség, hogy a modulok csak akkor töltődnek be, ha éppen szükség van rájuk.

Az `--enable-shared` változóval fordított Apache által létrehozott eredeti httpd.conf fájl némileg különbözik attól, mint amit a modulokkal egybefordított Apache készít. A legfontosabb különbség, hogy a parancsok IfModule tagok között foglalnak helyet, ez lehetővé teszi, hogy az Apache bizonyos modulok betöltése nélkül is elinduljon. Ezenkívül minden modult a LoadModule paranccsal kell betöltenünk, majd az AddModule paranccsal engedélyeznünk. Például:

```
LoadModule perl_module lobexec/libperl.so
AddModule mod_perl.c
```

A LoadModule-nak két értéke van: az egyik a modul neve, a másik az .so fájl. A névnek meg kell egyeznie azzal a névvel, mellyel a DSO modult fordítottuk, a fájlnévnek pedig a /usr/local/apache könyvtár egyik alkönyvtárában lévő fájlra kell mutatnia. A fenti példában (és alapértelmezés szerint is) a DSO modulok a /usr/local/apache/libexec könyvtárban helyezkednek el.

A httpd.conf fájl legtöbb parancsával ellentétben a LoadModule és AddModule parancsok esetében elhelyezésük sorrendje is számít. A LoadModule parancsnak meg kell előznie az AddModule parancsot. Mielőtt egy parancsot használnánk, az azt magában foglaló modult be kell töltenünk, és engedélyeznünk kell. Hogy még kellemesebb legyen az életünk, néhány modul csak akkor működik, ha azt egy másik modul után töltöttük be. Ha van olyan beállítóprogram, mely a LoadModule/AddModule parancsokat automatikusan beilleszti, akkor használjuk azt, ugyanis segítségével rengeteg tökéletlen beállításból fakadó hibát küszöbölhetünk ki.

Az apxs

Miután az Apache-t DSO támogatással lefordítottuk, az új modulokat bármikor beilleszthetjük, de ezeket az Apache fordításánál használt beállításokkal kell lefordítanunk. Ezt a *Ralf S. Engelschall* által írt **apxs** (Apache Extension) nevű program automatikusan elvégzi. Az apxs segítségével a modulokból DSO-kat (.so fájlokat) készíthetünk, ezeket pedig beilleszthetjük az Apache-ba.

Sajnos, a programhoz nem sok leírás jár, tehát a program feladatát és működését magunktól elég nehéz megérteni.

Tegyük fel, hogy az Apache-t már lefordítottuk DSO-támogatással. Néhány héttel később észrevesszük, hogy a kiszolgáló egy csomó kapcsolatot „File not found” üzenettel tagad meg, mert a felhasználók nem képesek kiolvasni és begépelni az URL-ekben meghatározott különleges karaktereket. Az egyik megoldás, hogy az URL-eket teljesen átalakítjuk „normális” karakterekkel. Egyszerűbb azonban, ha a mod_speling modult telepítjük, s így a félregépelésből és a nagybetű–kisbetű különbségekből adódó hibákat nagymértékben kiszűrhetjük. A mod_speling DSO-ként történő fordításához a

```
/usr/local/apache/bin/apxs -c mod_speling.oc
```

parancsot kell kiadnunk. Az **apxs** a gcc meghívásával fordítja le a mod_speling modult. Eredményül nem futtatható fájlt kapunk, hanem egy, az Apache által betölthető könyvtárfájlt. Ha a fordítás sikeres volt, akkor a mod_speling-et a következő paranccsal telepíthetjük:

```
/usr/local/apache/bin/apxs -i -n -a mod_speling.so
```

Az `apachectl configtest` parancs különösen jól jön új DSO modulok telepítésekor. A parancs lehetővé teszi, hogy a hozzáadott modul tényleg a helyén legyen és működjön, illetve az Apache megérti az IfModule tagokon kívül elhelyezkedő új parancsokat.

A mod_perl

A Perl nyelven írt új modulok írását és a meglévők beállítását lehetővé tevő mod_perl modult is használhatjuk DSO-ként. Ehhez, a mod_speling-hez hasonlóan, az apxs-re lesz szükségünk. A mod_perl azonban jóval összetettebb modul, mint a többi, és fordításához számos külső elemből származó adat szükséges. A mod_perl modult ezért az önálló Perl modulokhoz hasonlóan a perl Makefile.PL paranccsal kell beállítanunk, melyet egy `make` és egy `make install` parancs követ.

Ha a meglévő Apache-változatba szeretnénk a mod_perl egy újabb változatát fordítani, adjuk ki az alábbi parancsot:

```
perl Makefile.PL \
USE_APXS=1 WITH_APXS=/usr/local/apache/bin/apxs
```

Ha a mod_perlt nemcsak a PerlHandler, hanem a különböző Apache kezelők (handler) számára is elérhetővé kívánjuk tenni, akkor kapcsoljuk be az `EVERYTHING` kapcsolót:

```
perl Makefile.PL \
USE_APXS=1 WITH_APXS=/usr/local/apache/bin/apxs  \
EVERYTHING=1
```

Miután a Makefile-t a fenti módon elkészítettük, a mod_perlt a következő parancsokkal fordíthatjuk le és telepíthetjük a meglévő Apache-kiszolgálóba:

```
make
make test
make install
```

E módszerrel nemcsak a mod_perl egy új példányát telepíthetjük az Apache-ba, hanem frissíthetjük is a meglévő példányt. Annak ellenőrzéséhez, hogy a mod_perl bekerült-e az Apache-ba, jelentkezzünk be telnettel a kiszolgálóra a megfelelő (általában 80-as, vagy 8080-as) kapuszámmal, majd adjuk ki az alábbi parancsot:

```
HEAD / HTTP/1.0
```

Ez a kiszolgáló kezdőlapjának fejlécét küldi vissza, melyben többek között egy, a kiszolgáló típusát meghatározó Server fejléc sort is kell találnunk. A mod_perl ehhez az üzenethez saját jelzését is csatolja, tehát ha mindent jól csináltunk, akkor a következő szöveget kell látnunk a fejlécben:

```
Server: Apache/1.3.12 (UNIX) mod_perl/1.24
```

Mivel a mod_perl frissítései általában nem az Apache új változataival egy időben jelennek meg, ezért a fenti módszer különösen jól használható az új mod_perl telepítésére.

Az Apache::Status

A mod_status csak az Apache-folyamatok állapotáról tudósít, az egyes modulok pillanatnyi helyzetéről nem tudunk meg semmit. Mondhatnánk: miért is lenne ez fontos? Nem mindegy nekem, hogy épp mi történik a mod_mime, vagy épp a mod_speling modulban?
A mod_perl esetében azonban, ahol rengeteg összetett művelet folyik egyszerre, nem lenne rossz, ha állandó visszajelzést kaphatnánk ezekről. A Perl Apache::Status modulja, mely a mod_perl modullal működik, pontosan ezt az leírást képes nyújtani.
Az Apache::Status indításához új szakaszt kell létrehoznunk a httpd.conf fájlban. A mod_status-hoz hasonlóan most is új Location részt készítünk, mely kezelőt rendel egy virtuális URL-hez, a „/perl-status”-hoz:

```
PerlModule Apache::Status
<Location /perl-status>
       SetHandler    perl-script
       PerlHandler   Apache::Status
</Location>
```

A kiszolgáló újraindítása után (de küldhetünk neki HUP jelet is) a /perl-status URL lekérésekor egy menü tárul elénk, „Environment”, „Inheritance tree” és hasonló lehetőségekkel. A mod_perl más Perl moduljai, például a HTML::Mason, is képesek csatlakozni az Apache::Status-hoz, így azok tulajdonságaiba is betekintést nyerhetünk. A Mason esetében például egyszerű kezelőfelülettel tekinthetjük át a pillanatnyi helyzetet, illetve a lefordított és gyorstárba került összetevők listáját.

Egy rövid történet

A cikk írása előtti napokban éppen a fenti módszerekkel igyekeztem megoldást találni egy, a saját rendszeremben felmerült, a HTML::Mason telepítésével kapcsolatos nehézségre. Egy ismerősöm kiszolgálójának gondot okozott a növekvő forgalom kezelése: néhány óránként minden Mason-alapú kiszolgáló leállt, ezt egy-két órával később a nem Mason-alapú kiszolgálók leállása követte. A „leállás” azért talán erős kifejezés: a böngésző elküldte a kérelmet a kiszolgálónak, de a kapcsolat úgy tíz perc után időtúllépés miatt megszakadt.
Tehát pontosan mi is történt ott, és hogyan javítottam meg?
Az első gondolatom az volt, hogy a kiszolgálón elfogyott a fizikai memória. A top és a free segítségével megvizsgáltam a rendszert, de semmi különöset nem tapasztaltam. Egyrészről ez megnyugtató érzés volt, viszont ez azt is jelentette, hogy a kiszolgálón egyszerűen elfogynak a szabad folyamatok, annak ellenére, hogy a kapcsolatok legnagyobb számát 150-re állítottam. Saját honlapom elég szép forgalmat bonyolít, de azért 150 egyszerre kapcsolódó felhasználó még nálam is ritkaságszámba megy. Valami más lehet a baj, gondoltam. Az világos volt, valami a Masonnal és talán a mod_perl-lel nincs rendben. Ekkor úgy döntöttem, hogy megpróbálom a mod_perl-t frissíteni a legújabb változatra (akkor ez az 1.24 volt), a fentebb ismertetett eljárások segítségével. Frissítettem tehát a HTML::Mason-t és a kapcsolódó modulokat, az apachectl paranccsal újraindítottam az Apache-t, és reménykedtem, hogy a galibát elfelejthetjük.
Sajnos azonban a frissítés nem oldott meg semmit. A kiszolgáló néhány óra működés után továbbra is beszüntette a válaszadást a kérelmekre. Az Apache::Status-szal áttekintettem a mod_perl állapotsorát, de ott sem volt bibi.
Ekkor a mod_status modullal belenéztem az Apache állapotjelentésébe és észrevettem, hogy egyszer csak rengeteg Apache-folyamat leáll a válaszadás ("W" jel) közben. Tehát az állapotjelentés sorai lassan, de biztosan csupa "....." jelből (szabad folyamatok) "WWWWW" jelekké (válaszadás folyamatban) alakultak át. A Mason minden meghívása lefoglalt egy folyamatot, melyet soha többé nem engedett el! Ezek után nem csoda, hogy lassan leállt a kiszolgáló: ha a honlap Masonra alapuló részeit többen keresték volna fel, a leállás még hamarabb bekövetkezett volna.
Átnéztem a Mason beállításfájlját és rájöttem, hogy az Apache::Session modul (ez a mod_perl programjai számára a felhasználói tevékenység nyomon követését teszi lehetővé) nem tér vissza. Tehát a Mason-összetevő meghívásakor minden rendben zajlott, egészen addig, míg az eljárásnak vissza kellett volna térnie – ekkor ugyanis a futtató program a MySQL adatbázisra történő, örökkévalóságig tartó várakozásba kezdett. Az általam talált megoldás nem volt különösebben ügyes, de megállta a helyét: leállítottam az Apache::Session MySQL változatát (neve Apache::Session::MySQL) és helyette a hagyományos változatot indítottam el (neve Apache::Session::File). A kiszolgáló újraindítása után nagy örömünkre minden a legnagyobb rendben működött.

Összegzés

Az Apache egy csodálatos és megbízható HTTP-kiszolgáló, ennek ellenére rendkívül összetett, használatához és megfelelő behangolásához elkél némi tapasztalat. Ha a mod_perl is bekapcsolódik a folyamatokba, a helyzet még bonyolultabbá válik. Szerencsére az Apache-t úgy is telepíthetjük, hogy a modulok telepítése és frissítése egyszerűen és fájdalommentesen végrehajtható legyen.
A számos modul és segédeszköz (például a mod_status, vagy az Apache::Status) lehetővé teszi, hogy a „motorháztetőt” működés közben hajtsuk fel, ezek segítségével a bajok felderítése és kijavítása egyszerűbbé és gyorsabbá válik. Ezzel pedig rengeteg időt nyerhetünk – senki nem görnyed órákon, napokon keresztül a képernyő előtt csak azért, hogy végre működésre bírja azt a fránya kiszolgálót.

Az ATF honlapja: ➲http://www.lerner.co.il/atf/

Reuven M. Lerner (reuven@lerner.co.il) cége internetes tanácsadást vállal, székhelyük Modi'in-ben, Izraelben van. Éppen mostanában fejezi be (végre!) *Core Perl* című könyvét, mely a Prentice Hall kiadónál jelenik meg.