korlátozó típusú helyi hálózati modellt használ, amely segíthet a webalapú rendszereknél előforduló biztonsággal kapcsolatos aggodalmak eloszlatásában. Elsőként az ábrázolórendszert fejlesztettük ki, és annak bizonyítására használtuk fel, hogy a szélcsatorna ügyfelei adataikat a Weben keresztül is el tudják érni. Amikor elkezdtük az Amtek Engineering által készített Tecplot kereskedelmi adatmegjelenítő programot használni, elhatároztam, hogy az ábrázolórendszert e program köré írom meg. A felhasználók az ábrázolómintákat – a beállítások kiválasztásával és szövegmezők kitöltésével – egyszerűen HTML-űrlapon keresztül állíthatják be. Ezeket a mintákat azután Tecplot-parancsállományok előállítására használjuk, ezáltal képernyőn vagy papíron lehetőség nyílik az eredmény megtekintésére. Egy démon (szintén Perlben íródott) ugyanezeket a mintákat használja a nyomtatásra, amely minden szélcsatorna-kísérlet végén önműködően zajlik. A beállítóállomány szerkesztését egy másik webalapú programmal valósítottam meg, amelyben a kísérleti adatokat gyűjtő és egyszerűsítő programok vezérlőállományait gyors és egyszerű módszerrel lehet módosítani. A felhasználók olyan űrlapot látnak, amelynek minden sora szövegmezőket és választógombokat, valamint a hozzá tartozó értékneveket tartalmazza. Az a Perl program, amely ezt a HTML-űrlapot hozza létre, dinamikusan előállít egy JavaScript-kódot is, amelynek az a feladata, hogy az űrlap benyújtása előtt ellenőrizze a kitöltött adatok érvényességét. Ha érvénytelen bejegyzést talál, a beviteli mező mellett villogó nyíl jelenik meg, és egy előugró párbeszédablakban a hiba jellege lesz olvasható.
Az adatállomány-megjelenítő egy egyszerű CGI program, amely egy adott szélcsatornapróba adataiért kutatja át a lemezterületet. Minden bejegyzéshez, amely a keresési mintára illeszkedik, egy HTML-gombot készít. Ezek a gombok táblázatban helyezkednek el, ahol minden sor a hasonló csatornapróbákat, illetve minden oszlop a megadott adattípust tartalmazza (például nyers, egyszerűsített). Bármely gomb megnyomására új böngészőablak nyílik meg, ahol a kiválasztott állomány formázott és elemzett alakban tekinthető meg. Ezután a felhasználók számára adott a lehetőség, hogy a saját gépükre CSV, Matlab vagy bármely más formátumban letöltsék. Minden új alkalmazás és számos régebbi kód állapotüzeneteket állít elő – eseményeket, amelyeket eseménynaplózó rendszerrel kezelünk. Ez a rendszer két fő részből tevődik össze: az első egyszerű Perl-démon, amely egy TCP/IP-kaput figyel, hogy érkezik-e rajta üzenet, s amennyiben igen, a naplóállományokban tárolja őket. A rendszer másik része egy webalapú megjelenítő, amellyel a felhasználók különböző szempontok

*4. kép* Forgó F18-as modell a vízcsatornában

alapján kereshetnek a naplóállományok között, mint például az esemény ideje, a számítógép neve, az esemény súlyossági szintje. Ez a program jelentéktelennek tűnik, pedig nélkülözhetetlen, mert az adatgyűjtő, -kezelő és -megjelenítő rendszer számos, különböző operációs rendszerrel működő számítógépből áll. Az ilyenfajta osztott rendszerekben a hibakeresés nagyon nehéz (különösen az összetettség miatti összehangolás okoz gondokat), általános eseménynaplózó rendszer nélkül csaknem lehetetlen. A felhasználók szempontjából az egyetlen nem interaktív eszköz a dinamikus adatmegjelenítő rendszer: Perl-kiszolgálón alapul, amely az adatgyűjtőrendszertől adatcsomagokat fogad. A felhasználók egy CGI-programon keresztül a kiszolgálóhoz kapcsolódva tudják megjeleníteni ezeket az adatokat. A CGI-program NPH-t (nonparsed header) vagy „server push”-t használ. A program egy adattáblázatot jelenít meg, amelyben az újabb adatokat dinamikusan a táblázat tetejére írja, a régi adatot pedig lefelé tolja. A program készítése folyamán aggódtam a memóriahézagok miatt, amelyek nemcsak Perlben vagy Apache-ban, de az ügyfelek böngészőjében is előfordulhatnak. Fölöslegesen, hiszen akadtak különleges NPH-ügyfelek, amelyek a kiszolgálóhoz folyamatosan kapcsolódva olyan anyagokért kutatták át a rendszert, amelyek több mint öt hete készültek. Eközben minden gond nélkül 500 MB-nál több adatot jelenített meg.
Ennek az öt programnak bármelyike külön-külön jól használható lenne, de aligha nevezhetők forradalminak. Mihelyt azonban egyesítjük őket, egyszerű, szilárd és nagyméretű környezetet kapunk. Nincs szükség nehezen érthető parancsokra, hosszú adatelérési útvonalakra, parancsbillentyű-kombinációkra vagy bármi másra, amely különféle típusú kezelőfelületekre jellemző. Nem kell mást tenni, mint kattintani, kijelölni, és kitölteni a mezőket – mindenki tudja, mit és hogyan tegyen.

## A jövő

Teendőim hosszú listáján előkelő helyen szerepel azoknak a Perl-kódoknak a kétprocesszoros Intel/Linux-rendszerre történő ültetése, amelyeket az utolsó megmaradt SGI-rendszeren fejlesztettem. Bár a programok futtatása a jelenlegi formájukban jelentéktelen feladat, éltem az összes kód újrahangolásának lehetőségével. Még egy alkalmazás befejezése lenne különösen fontos: a modell viselkedését vezérlő rendszer felhasználói felületéé. Ráadásul figyelembe kell vennem adatformátumaink váltását – az arcane házilag fejlesztett formátumától az XML-ig. Ez szintén néhány új kód szükségességét eredményezi. Távolabbra pillantva a jövőbe, remélem, elég időm lesz kifejleszteni néhány VRML-alkalmazást is, amelyek a szélcsatornában elhelyezett modellek és szondák terhelés- és nyomásviszonyait képesek utánozni, és 3D-ben dinamikusan megjeleníteni. A műszeres csoport is vizsgálja a beágyazott, illetve valós idejű Linux-rendszerek felhasználásának lehetőségét.
Mire mindezen munkák elkészülnek, a nyílt forrású programok egyre növekvő fontosságú szerepe már vitathatatlan lesz az Aerodinamikai Laboratórium mindennapi munkájában.

*Steve Jenkins* az Aerodynamics Laboratory of the Institute for Aerospace Research vezető programozója és elemzője, a légi kutatások adatfeldolgozásában több mint húszéves tapasztalattal rendelkezik.

# Amikor a Palm és a Linux beszélgetni kezd egymással...

Két egyetemi hallgató megoldotta a Palm és a linuxos számítógép összehangolását.

A Palm nagyszerű hordozható készülék: jegyzetelhetünk, találkozókat tervezhetünk vagy akár naplót is írhatunk vele. Csodálatos, hogy mindenhol velünk lehet útközben. Belső hálózatunk kiszolgálója szintén bámulatos masina. Vállalati terveinket, a megbeszélések napirendi pontjait, az érdekes feljegyzéseket, a címeket és a teljes üzleti adatbázist tárolja. Ez a kiszolgáló Linuxot, Apache-kiszolgálót és egy MySQL-adatbázist futtat, amelyeket egy, a célra tervezett alkalmazásmotor fog össze.

Ugye, csodálatos lenne, ha a két gépet össze tudnánk kapcsolni? Bárcsak sikerülne a Palmról elérni a Linux-kiszolgálón található adatbázist – máris jó úton haladnánk. A leírás szűkszavú, az Internet azonban hatalmas. ***Kevin*** és ***Jeffrey*** két egyetemista, akik kis erőráfordítással új megoldást fejlesztettek ki. Ennek alapján végezhetjük el a megfelelő változtatásokat a Palm-gépeken és a vállalati kiszolgálón, s e módosításokat a másik eszköz adatbázisaival is végrehajthatjuk.

A lejjebb látható kódokat egy RedHat 6.x Linuxot futtató Intel-alapú gépen és egy Palm OS 3.5-öt futtató, soros bölcsővel rendelkező Palm Vx-en próbáltuk ki, de más összeállítás sem okozhat gondot. Az alkalmazott könyvtárfájlok a Palm első piaci megjelenése óta (akkoriban még a 3Com gyártotta a gépet) nem változtak. Feltételezem (bár nem próbáltam ki), hogy a program a Visor-gépeken is gond nélkül működik, mivel ugyanazt az operációs rendszert használják.

Első lépésként az egyszerűbb résszel kezdjük: a Palmot kapcsoljuk össze a kiszolgálóval. Először is a Palm bölcsőjét kell összekötnünk a kiszolgáló soros csatlakozójával. Ezután létrehozunk egy pilot nevű eszközt, ami nem más, mint a soros kapu (esetünkben a */dev/ttyS0*) másodneve:

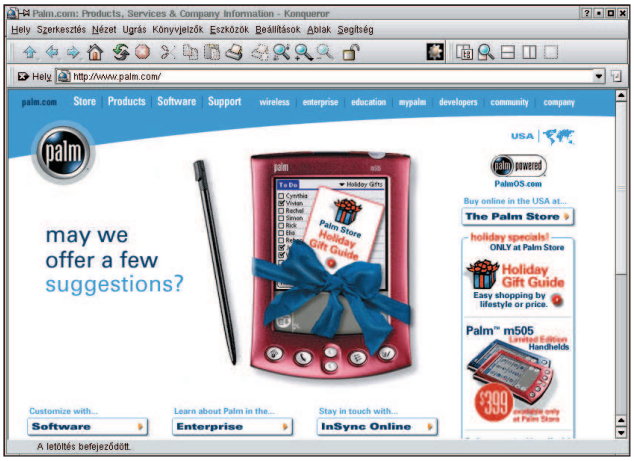
```
ln /dev/ttyS0 /dev/pilot
```

Most már egy C program és a ***HotSync*** gomb megnyomásával megnyithatjuk a kapcsolatot a Palmmal. Miután a kapcsolat létrejött, már csak a Palm adatbázisaiból kell a megfelelő adatokat kiolvasnunk.

A Palm és a számítógép közti kapcsolat és adatátvitel a ***pi*** könyvtárral egyszerűen megvalósítható. Ez a könyvtár a BSD-csatolófelületet utánozza: létrehoz egy foglalatot (socket), hozzákapcsolja az eszközhöz, figyel a bejövő kapcsolati kérelemre és elfogadja. A bejövő kapcsolatot a Palm és a bölcső kettőse váltja ki akkor, amikor a felhasználó megnyomja a HotSync gombot. Az *1. listán* láthatjuk, hogyan kell létrehozni egy Palm-kapcsolatra várakozó démont.

Miután a kapcsolat megvalósult, miként érhetjük el a Palm adatbázisait? Ezek mindegyike névvel rendelkezik. Az adatbázisokat a nevükkel nyithatjuk meg, majd meg kell adnunk az elérni kívánt rekordot, sőt az egész adatbázist is átnézhetjük. Macintosh- vagy Windows-gépeken ezt csővezetékek alkalmazásával oldják meg. A Palm is biztosít csővezetékeket e felületeken az összes, a Palm OS csomagba tartozó szabványos adatbázis számára. A Palm adatbázis-kezelő lehetővé teszi, hogy az adatbázisnak csak a módosított rekordjait tekintsük végig. Módosított – mióta is? Nos, a legutóbbi összehangolás óta, amikor a kiszolgáló utoljára érte el ezt az adatbázist. Ezt tehát programjainkban az összehangolás után kell megtennünk – a *2. listán* látható nyitott kapcsolatnál kell futtatni.

Amennyiben a Palm-adatbázisból rekordokat olvasunk ki, az nem számít összehangolásnak. Ennél többre van szükség, például arra, hogy a Palmra írunk, törlünk belőle és a saját MySQL-adatbázisunkból olvasunk. Mivel a MySQL-adatbázishoz való kapcsolódás ismertetése túlmutat e cikk keretein, most nem szólunk az összehangolás további részleteiről. ***Kevin Velghe*** nagyszerű leírást tett közzé a témáról, amelyre a ➲ http://www.duo.be/palm/mysql_palm.html címen bukkanhatunk rá.




*1. lista* A Palm összehangolása

```
Main() {
        int sd;
        struct pi_sockaddr addr;

        sd = pi_socket(PI_AF_SLP,
PI_SOCK_STREAM, PI_PF_PADP);
        addr.pi_family = PI_AF_SLP;
        strcpy(addr.pi_device, "/dev/pilot");
        pi_bind(sd, (struct sockaddr*)&addr,
➥sizeof(addr));

        sd = pi_listen(sd, 1);

        sd = pi_accept(sd, 0, 0);
        printf("HurrÆ! LØtrej tt a
              ➥kapcsolat...");
        pi_close(sd);
}
```

*2. lista* A módosított rekordok átfésülése

```
{
        int db, len, I, attr;
        recordid_t id;
        unsigned char buffer[4096];
...lØtrej n a kapcsolat...
        sd = pi_accept(sd, 0, 0);
        dlp_OpenDB(sd, 0, 0x40+0x80,
                ➥"DateBookDB", &db);
        for (;;) {
                len = dlp_ReadNextModifiedRec
                        ➥(sd, db, buffer, &id,
                        ➥&I, 0, &attr, 0);
                if (len < 0) break;
                printf(buffer); printf("\n");
        }
        dlp_ResetSyncFlags(sd, db);
        dlp_CleanUpDatabase(sd, db);
        dlp_CloseDB(sd, db);

        pi_close(sd);
}
```


### További érdekességek

*Michael J. Hammel* a Linux Journal 1998 júniusában megjelent „Linux and the PalmPilot” című cikke nem a legfrissebb, de ismerteti a pilot-xfer használatát, ami nagyon hasznos segédprogram. Elolvashatjuk a ➲ http://www.linuxjournal.com/lj-issues/issue50/2711.html címen.
A „PalmOS Desktop HOWTO” című HOGYAN-ja a ➲ http://www.linuxdoc.org/HOWTO/PalmOS-HOWTO.html címen található meg. Az ➲ http://orbits.com-ra mutató hivatkozások legtöbbje már nem működik, hiszen ez a vállalat minden bizonnyal megszüntette az említett HOGYAN-ok fejlesztését. Ha valakinek sikerül megtalálnia az elveszett anyagokat, kérjük, küldje el őket a palm@duo.be címre, hogy a ➲ http://www.duo.be/palm címen mindenki számára elérhetővé tehessük.

*3. lista* A pack függvény

```
{
        int app_size, len;
        recordid_t pal_id, new_id;
        unsigned char buffer[512];
        struct Appointment app;

...
        strcpy(app.description,"Tennival ");
        app.begin = ...
...
        size = pack_Appointment(&app, buffer,
                ➥512);
        palm_id = 0;
        len = dlp_WriteRecord(sd, db, 0,
                ➥palm_id, 0, AppBuffer,
                ➥Appointment_size, &new_id);

}
```

A Palm-adatbázisban tárolt rekordok saját számokkal rendelkeznek. Amikor az eszközre rekordot írunk, ezt a számot kapjuk visszatérési értékként. Ezt a gépen vagy egy központi adatbázisban érdemes tárolnunk, így egy adott rekordot bármikor törölhetünk vagy frissíthetünk.
A `dlp_WriteRecord` egy Palm-rekordazonosítót fogad el. Amennyiben ez nulla, a Palm OS újat foglal le a számunkra; ha pedig létező azonosítót adunk át, akkor a megfelelő rekord kerül frissítésre. A legtöbb szabványos adatbázisrendszerben a rekordot a `pack` függvény csomagolja egy átmeneti tárolóba. Ez a folyamat a *3. listán* látható.

## A Palm azonosítása

Amennyiben egy kiszolgálóval (tulajdonképpen egy bölcsővel) több Palmot is használunk (mint ebben az esetben is), meg kell határoznunk, hogy éppen melyik Palm csatlakozik a bölcsőre. Amint a kapcsolat létrejött, hívjuk meg a `ReadUserInfo` függvényt:

```
{
        int db, len, I, attr;
        recordid_t id;
        struct PilotUser U;
...lØtrej n a kapcsolat...
        sd = pi_accept(sd, 0, 0);
        dlp_ReadUserInfo(sd,&U);
        printf("Palm neve: %s", U.username);

        pi_close(sd);
}
```

## Törölt rekordok

A Palm adatbázis-kezelő a rekordokat kiolvasás után nem törli – törlésre jelöli ki őket, de azt is megteheti, hogy a gépen vagy a kiszolgálón mentésre jelöli ki őket. Módosított rekord olvasásakor a fájltulajdonságokat ellenőrizni kell, amelyből megállapítható, hogy az adott rekordot törölni (vagy menteni) kell-e. Amint az adatbázis kitisztul, végérvényesen törlődik a Palmról: