21. ÓRA

Munka kiszolgálói környezetben

A korábbi fejezetekben áttekintettük, hogyan társaloghatunk távoli számítógépekkel és hogyan vehetünk át adatokat a felhasználótól. Ebben a fejezetben ismét kitekintünk, olyan eljárásokat tanulunk meg, amelyek külső programok saját gépünkön való futtatására használhatók. A fejezet példái Linux operációs rendszerre készültek, de a legtöbb alapelv felhasználható Microsoft Windows rendszerben is.

A fejezetben a következőket tekintjük át:

- Hogyan közvetítsünk adatokat a programok között?
- A héjparancsok végrehajtására és az eredmények megjelenítésére milyen lehetőségeink vannak?
- Hogyan írhatunk biztonságosabb PHP programokat?

Folyamatok összekötése a popen() függvénnyel

Ahogy a fájlokat nyitottuk meg írásra vagy olvasásra az `fopen()` segítségével, ugyanúgy nyithatunk adatcsatornát két folyamat között a `popen()` paranccsal. A `popen()` paramétereiben meg kell adni egy parancsot elérési úttal, és azt, hogy írási vagy olvasási módban akarjuk használni a függvényt. A `popen()` visszatérési értéke az `fopen()` visszatérési értékéhez hasonló fájlazonosító. A `popen()`-nek a `'w'` jelzővel adhatjuk meg, hogy írási módban, az `'r'` jelzővel pedig azt, hogy olvasási módban akarjuk használni a függvényt. Ugyanazzal az adatcsatornával nem lehet egyszerre írni és olvasni is egy folyamatot. Amikor befejeztük a munkát a `popen()` által megnyitott folyamattal, a `pclose()` paranccsal le kell zárnunk az adatcsatornát. A `pclose()` függvény paraméterében egy érvényes fájlazonosítót kell megadni.

A `popen()` használata akkor javasolt, amikor egy folyamat kimenetét sorról sorra szeretnénk elemezni. A 21.1-es példában a GNU `who` parancs kimenetét elemezzük, és a kapott felhasználóneveket `mailto` hivatkozásokban használjuk fel.

21.1. program A who UNIX parancs kimenetének olvasása a popen() segítségével

```
 1: <html>
 2: <head>
 3: <title>21.1. program A who UNIX parancs kimenetének
 4: olvasása a popen() segítségével</title>
 5: </head>
 6: <body>
 7: <h2>A rendszerbe bejelentkezett felhasználók</h1>
 8: <?php
 9: $ph = popen( "who", "r" )
10:          or die( "A 'who' paranccsal nem vehető fel
                     kapcsolat" );
11: $kiszolgalo="www.kiskapu.hu";
12: while ( ! feof( $ph )  )
13:      {
14:      $sor = fgets( $ph, 1024 );
15:      if ( strlen( $sor ) <= 1 )
16:          continue;
17:      $sor = ereg_replace( "^([a-zA-Z0-9_\-]+).*",
18:      "<a
         href=\"mailto:\\1@$kiszolgalo\">\\1</a><br>\n",
19:        $sor );
20:      print "$sor";
21:      }
```

21.1. program (folytatás)

```
22: pclose( $ph );
23: ?>
24: </body>
25: </html>
```

A `who` parancs eredményének kiolvasásához szükségünk van egy fájlmutatóra a `popen()` függvénytől, így a `while` utasítás segítségével sorról sorra elemezhetjük a kimenetet. Ha a sorban olvasható kimenet csupán egyetlen karakter, a `while` ciklus következő lépésére ugrunk, kihagyva az elemzést, különben az `ereg_replace()` függvénnyel újabb HTML hivatkozással és soremeléssel bővítjük a végeredményt. Végül a `pclose()` függvénnyel lezárjuk az adatcsatornát. A program egy lehetséges kimenete a 21.1. ábrán látható.

21.1. ábra

A who UNIX parancs kimenetének olvasása

A `popen()` függvény használható parancsok bemenetének írására is. Ez akkor hasznos, ha egy parancs a szabványos bemenetről vár parancssori kapcsolókat. A 21.2. példában azt láthatjuk, hogyan használhatjuk a `popen()` függvényt a `column` parancs bemenetének írására.

21.2. program A column parancs bemenetének írása a popen() függvény segítségével

```
 1: <html>
 2: <head>
 3: <title>21.2. program A column parancs bemenetének
          írása
 4: a popen() függvény segítségével</title>
 5: </head>
 6: <body>
 7: <?php
 8: $termekek = array(
 9:          array( "HAL 2000", 2, "piros" ),
10:          array( "Modem",  3, "kék" ),
11:          array( "Karóra", 1, "rózsaszín"  ),
12:          array( "Ultrahangos csavarhúzó", 1,
                    "narancssárga"   )
13:          );
14: $ph = popen( "column -tc 3 -s / >
fizetve/3as_felhasznalo.txt", "w" )
15:      or die( "A 'column' paranccsal nem vehető fel
                  kapcsolat" );
16: foreach ( $termekek as $termek )
17:      fputs( $ph, join('/', $termek)."\n");
18: pclose( $ph );
19: ?>
20: </body>
21: </html>
```

A 21.2. példában megfigyelhető, hogyan érhetők el és írhatók ASCII táblázatként fájlba a többdimenziós tömbök elemei. A `column` parancs bemenetéhez adatcsatornát kapcsolunk, amin keresztül parancssori kapcsolókat küldünk. A `-t` kapcsolóval megadjuk, hogy táblázattá formázza a kimenetet, a `-c 3` megadja a szükséges oszlopok számát, a `-s /` a `/`-t állítja be mezőelválasztóként. Megadjuk, hogy a végeredményt a `3as_felhasznalo.txt` fájlba írja. A `fizetve` könyvtárnak léteznie kell és a megfelelő jogosultság szükséges, hogy a program írhasson bele.

Most egyetlen utasítással egyszerre több dolgot tettünk. Meghívtuk a `column` programot és kimenetét fájlba írtuk. Valójában parancsokat adtunk ki a héjnak: ez azt jelenti, hogy az adatcsatorna használatával, egy folyamat futtatásával más feladatokat is elindíthatunk. A `column` kimenetét például a `mail` parancs segítségével postázhatjuk valakinek:

```
popen( "column -tc 3 -s / | mail kiskapu@kiskapu.hu", "w" )
```

Így elérhetjük, hogy a felhasználó által beírt adatokkal rendszerparancsokat hajtsunk végre PHP függvényekkel. Ennek néhány hátrányos tulajdonságát még áttekintjük az óra további részében.

Miután rendelkezünk egy fájlazonosítóval, a `$termekek` tömb minden elemén végiglépkedünk. Minden elem maga is egy tömb, amit a `join()` függvény segítségével karakterlánccá alakítunk. Az üres karakter helyett mezőelválasztóként a `'/'` karaktert választjuk. Ez azért szükséges, mert ha üres karakterek jelennének meg a termékek tömbjében, az összezavarná a `column` parancsot. Az átalakítás után a karakterláncot és egy újsor karaktert írunk ki az `fputs()` függvénnyel.

Végül lezárjuk az adatcsatornát. A `3as_felhasznalo.txt` fájlban a következők szerepelnek:

```
HAL 2000                  2  piros
Modem                     3  kék
Karóra                    1  rózsaszín
Ultrahangos csavarhúzó    1  narancssárga
```

A kódot hordozhatóbbá tehetjük, ha a szöveg formázására a `sprintf()` függvényt használjuk.

Parancsok végrehajtása az exec() függvénnyel

Az `exec()` egyike azoknak a függvényeknek, amelyekkel parancsokat adhatunk ki a héjnak. A függvény paraméterében meg kell adni a futtatandó program elérési útját. Ezen kívül megadható még egy tömb, mely a parancs kimenetének sorait fogja tartalmazni és egy változó, amelyből a parancs visszatérési értéke tudható meg.

Ha az `exec()` függvénynek az `'ls -al .'` parancsot adjuk meg, akkor az aktuális könyvtár tartalmát jeleníti meg. Ez látható a 21.3. példában.

21.3. program Könyvtárlista betöltése a böngészőbe

```
1: <html>
2: <head>
3: <title>21.3. program Könyvtárlista betöltése
          a böngészőbe </title>
4: </head>
5: <body>
```

21.3. program (folytatás)

```
 6: <?php
 7: exec( "ls -al .", $kimenet, $vissza );
 8: print "<p>Visszatérési érték: $vissza</p>";
 9: foreach ( $kimenet as $fajl )
10:     print "$fajl<br>";
11: ?>
12: </body>
13: </html>
```

Ha az `ls` parancs sikeresen végrehajtódik, a visszatérési érték `0` lesz. Ha a program a megadott könyvtárat nem találja vagy az nem olvasható, a visszatérési érték `1`.

A végeredmény szempontjából nem tettünk semmi újat, hiszen az `opendir()` és a `readdir()` függvényekkel ugyanezt elérhettük volna, de elképzelhető olyan eset, amikor rendszerparancsokkal vagy korábban megírt Perl programokkal sokkal gyorsabban megoldhatjuk a feladatot, mint PHP függvényekkel. Ha a PHP program kifejlesztésének sebessége fontos szempont, esetleg érdemesebb a korábban megírt Perl program meghívása mellett dönteni, mint átültetni azt PHP-be, legalábbis rövid távon. Meg kell azonban jegyeznünk, hogy rendszerparancsok használatával programjaink több memóriát fogyasztanak és többnyire futásuk is lassabb.

21.2. ábra

A könyvtárlista betöltése a böngészőbe az exec() függvény segítségével

Külső programok futtatása a system() függvénnyel vagy a ` műveletjel segítségével

A `system()` függvény az `exec()` függvényhez hasonlóan külső programok indítására használható. A függvénynek meg kell adni a futtatandó program elérési útját, valamint megadható egy változó, amely a program visszatérési értékét tartalmazza majd. A `system()` függvény a kimenetet közvetlenül a böngészőbe írja. A következő kódrészlet a `man` parancs leírását adja meg:

```
<?php
print "<pre>";
system( "man man | col -b", $vissza );
print "</pre>";
?>
```

A `PRE` HTML elemet azért adtuk meg, hogy a böngésző megtartsa a kimenet eredeti formátumát. A `system()` függvényt használtuk, hogy meghívjuk a `man` parancsot, ennek kimenetét hozzákapcsoltuk a `col` parancshoz, amely ASCII-ként újraformázza a megjelenő szöveget. A visszatérési értéket a `$vissza` változóba mentjük. A `system()` közvetlenül a böngészőbe írja a kimenetet.

Ugyanezt az eredményt érhetjük el a ` műveletjel használatával. A kiadandó parancsot egyszerűen ilyen fordított irányú aposztrófok közé kell tennünk. Az így megadott parancsot a rendszer végrehajtja és visszatérési értékként a parancs kimenetét adja, amit kiírhatunk vagy változóban tárolhatunk.

Íme az előző példa ilyetén megvalósítása:

```
print "<pre>";
print `man man | col -b`;
print "</pre>";
```

Vegyük észre, hogy ebben a megvalósításban rögtön kiírtuk a végeredményt.

Biztonsági rések megszüntetése az escapeshellcmd() függvény használatával

Az `escapeshellcmd()` függvény ismertetése előtt tekintsük át, mivel szemben van szükségünk védelemre. A példa erejéig tegyük fel, hogy meg szeretnénk engedni a látogatóknak, hogy különböző súgóoldalakat tekinthessenek meg. Ha bekérjük egy oldal nevét, a felhasználó bármit beírhat oda. A 21.4. példában található programot ne telepítsük, mert biztonsági rést tartalmaz!

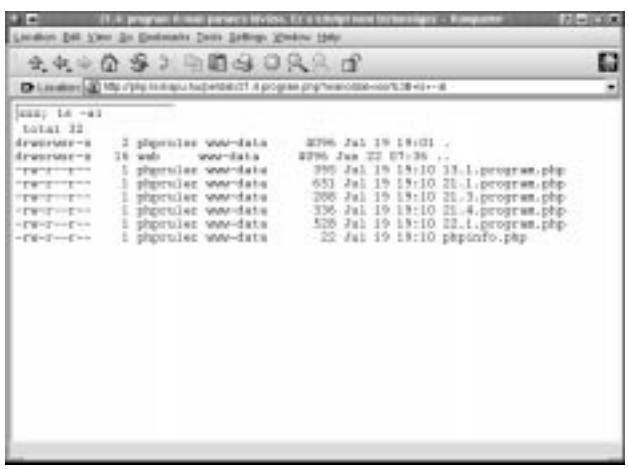
21.4. program A man program meghívása

```
 1: <html>
 2: <head>
 3: <title>21.4. program A man program meghívása.
 4:         Ez a kód nem biztonságos</title>
 5: </head>
 6: <body>
 7: <form>
 8: <input type="text" value="<?php print $manoldal;
                ?>" name="manoldal">
 9: </form>
10: <pre>
11: <?php
12: if ( isset($manoldal) )
13:      system( "man $manoldal | col -b" );
14: ?>
15: </pre>
16: </body>
17: </html>
```

Korábbi példánkat egy szöveges mezővel és a `system()` függvénnyel egészítettük ki. Megbízhatónak tűnik, UNIX rendszeren azonban a felhasználó a `manoldal` mezőhöz hozzáadhatja saját parancsait, így hozzáférhet a kiszolgálón számára tiltott részen lévő adatokhoz is. Erre láthatunk példát a 21.3. ábrán.

21.3. ábra

A man program meghívása



A felhasználó az oldalon keresztül kiadta az `xxx; ls -al` parancsot. Ezt a `$manoldal` változóban tároljuk. A program futása közben a `system()` függvénynek a következő parancsot kell végrehajtania:

```
"man xxx; ls -al | col -b"
```

Vagyis meg kell jelenítenie az `xxx` parancshoz tartozó súgóoldalt, ami természetesen nem létezik. Ebben az esetben a `col` parancs bemenete a teljes könyvtárlista lesz. Ezzel a támadó kiírathatja a rendszer bármelyik olvasható könyvtárának tartalmát. A következő utasítással például a `/etc/passwd` tartalmát kapja meg:

```
xxx; cat /etc/passwd
```

Bár a jelszavak egy titkosított fájlban, az `/etc/shadow`-ban vannak tárolva, amely csak a rendszergazda (`root`) által olvasható, ez mégis egy igen veszélyes biztonsági rés. A legegyszerűbben úgy védekezhetünk ellene, hogy soha nem engedélyezzük, hogy a felhasználók közvetlenül adjanak parancsot a rendszernek. Ennél kicsit több lehetőséget ad, ha az `escapeshellcmd()` függvénnyel fordított perjel (`\`) karaktert adunk minden metakarakterhez, amit a felhasználó kiadhat. Az `escapeshellcmd()` függvény paramétere egy karakterlánc, végeredménye pedig egy átalakított másolat. A korábbi kód biztonságosabb változata a 21.5. példában található.

21.5. program A felhasználói bemenet javítása az escapeshellcmd() függvény használatával

```
 1: <html>
 2: <head>
 3: <title>21.5. program A felhasználói bemenet javítása
 4:         az escapeshellcmd() függvény
            használatával</title>
 5: </head>
 6: <body>
 7: <form>
 8: <input type="text" value="<?php print $manoldal; ?>"
              name="manoldal">
 9: </form>
10: <pre>
11: <?php
```

21.5. program (folytatás)

```
12: if ( isset($manoldal) )
13:      {
14:      $manoldal = escapeshellcmd( $manoldal );
15:      system(   "man $manoldal | col -b" );
16:      }
17: ?>
18: </pre>
19: </body>
20: </html>
```

Ha a felhasználó most kiadja az `xxx; cat etc/passwd` parancsot, akkor a `system()` függvény meghívása előtt az `escapeshellcmd()` az `xxx\; cat /etc/passwd` paranccsá alakítja azt, azaz a `cat` utasítás súgóját kapjuk a jelszófájl helyett.

A rendszer biztonságát az `escapeshellcmd()` függvény segítségével tovább növelhetjük. Lehetőség szerint kerüljük azokat a helyzeteket, ahol a felhasználók közvetlenül a rendszernek adnak ki parancsokat. A programot még biztonságosabbá tehetjük, ha listát készítünk az elérhető súgóoldalakról és még mielőtt meghívnánk a `system()` függvényt, összevetjük a felhasználó által beírtakat ezzel a listával.

Külső programok futtatása a passthru() függvénnyel

A `passthru()` függvény a `system()`-hez hasonló, azzal a különbséggel, hogy a `passthru()`-ban kiadott parancs kimenete nem kerül átmeneti tárba. Így olyan parancsok kiadására is lehetőség nyílik, amelyek kimenete nem szöveges, hanem bináris formátumú. A `passthru()` függvény paramétere egy rendszerparancs és egy elhagyható változó. A változóba kerül a kiadott parancs visszatérési értéke.

Lássunk egy példát! Készítsünk programot, amely a képeket kicsinyített mintaként jeleníti meg és meghívható HTML vagy PHP oldalalakról is. A feladatot külső alkalmazások használatával oldjuk meg, így a program nagyon egyszerű lesz. A 21.6. példában látható, hogyan küldhetünk a képről mintát a böngészőnek.

21.6. program A passthru() függvény használata képek megjelenítésére

```
1: <?php
2: if ( isset($kep) && file_exists( $kep ) )
3:      {
4:      header( "Content-type: image/gif" );
5:      passthru( "giftopnm $kep | pnmscale -xscale
                  .5 -yscale .5 | ppmtogif" );
6:      }
7: else
8:      print "A(z) $kep nevű kép nem található.";
9: ?>
```

Vegyük észre, hogy nem használtuk az `escapeshellcmd()` függvényt, ehelyett a felhasználó által megadott fájl létezését ellenőriztük a `file_exists()` függvénnyel. Ha a `$kep` változóban tárolt kép nem létezik, nem is próbáljuk megjeleníteni. A program még biztonságosabbá tehető, ha csak bizonyos kiterjesztésű fájlokat engedélyezünk és korlátozzuk az elérhető könyvtárakat is.

A `passthru()` hívásakor három alkalmazást indítunk el. Ha ezt a programot használni akarjuk, rendszerünkre telepítenünk kell ezen alkalmazásokat és meg kell adni azok elérési útját. Először a `giftopnm`-nek átadjuk a `$kep` változó értékét. Az beolvassa a GIF képet és hordozható formátumúra alakítja. Ezt a kimenetet rákapcsoljuk a `pnmscale` bemenetére, amely 50 százalékkal lekicsinyíti a képet. Ezt a kimenetet a `ppmtogif` bemenetére kapcsoljuk, amely visszaalakítja GIF formátumúvá, majd a kapott képet megjelenítjük a böngészőben.

A programot a következő utasítással bármelyik weboldalról meghívhatjuk:

```
<img src="21.6.program.php?kep=<?php
 ➥ print urlencode("/utvonal/kep.gif") ?>">
```

Külső CGI program meghívása a virtual() függvénnyel

Ha egy oldalt HTML-ről PHP-re alakítunk, azt vesszük észre, hogy a kiszolgálóoldali beillesztések nem működnek. Ha Apache modulként futtatjuk a PHP-t, a `virtual()` függvénnyel külső CGI programokat hívhatunk meg, például Perl vagy C nyelven írt számlálókat. Minden felhasznált CGI programnak érvényes HTTP fejléccel kell kezdenie a kimenetét.

Írjunk egy Perl CGI programot! Ne aggódjunk, ha nem ismerjük a Perlt. Ez a program egyszerűen egy HTTP fejlécet ír ki és minden rendelkezésre álló környezeti változót felsorol:

```
#!/usr/bin/perl -w
print "Content-type: text/html\n\n";
foreach ( keys %ENV ){
    print "$_: $ENV{$_}<br>\n";
}
```

Mentsük a programot a `cgi-bin` könyvtárba `proba.pl` néven.
Ezután a `virtual()` függvénnyel a következőképpen hívhatjuk meg:

```
<?php
virtual("/cgi-bin/proba.pl");
?>
```

Összefoglalás

Ebben a fejezetben áttekintettük, hogyan működhetünk együtt a héjjal és rajta keresztül a külső alkalmazásokkal. A PHP sok mindenre használható, de lehet, hogy külső programok használatával az adott probléma elegánsabb megoldásához jutunk.

Megtanultuk, hogyan kapcsolhatunk össze alkalmazásokat a `popen()` függvény segítségével. Ez akkor hasznos, amikor a programok a szabványos bemenetről várnak adatokat, de mi egy alkalmazás kimenetéből szeretnénk továbbítani azokat.

Megnéztük, hogyan használható az `exec()` és a `system()` függvény, valamint a fordított irányú aposztróf műveletjel a felhasználói utasítások közvetítésére a rendszermag felé. Láttuk, hogyan védekezzünk a betörésre használható utasítások ellen az `escapeshellcmd()` függvény segítségével. Láttuk, hogyan fogadhatók bináris adatok rendszerparancsoktól a `passthru()` függvénnyel és hogy hogyan valósíthatjuk meg a kiszolgálóoldali beillesztéseket (SSI) a `virtual()` függvénnyel.

Kérdések és válaszok

Sokat emlegettük a biztonságot ebben a fejezetben. Honnan tudhatunk meg többet a szükséges biztonsági óvintézkedésekről?
A legbővebb számítógépes biztonsággal foglalkozó forrás Lincoln Stein (a híres Perl modul, a CGI.pm szerzője) által fenntartott FAQ. Megtalálható a `http://www.w3.org/Security/Faq/` címen. Érdemes a PHP kézikönyv biztonságról szóló részét is tanulmányozni.

Mikor használjunk külső alkalmazásokat saját PHP kódok helyett?
Három szempontot kell megvizsgálnunk: a hordozhatóságot, a fejlesztési sebességet és a hatékonyságot. Ha saját kódot használunk külső programok helyett, programunk könnyebben átvihető lesz a különböző rendszerek között. Egyszerű feladatoknál, például könyvtár tartalmának kiíratásakor, saját kóddal oldjuk meg a problémát, így csökkenthetjük a futási időt, mivel nem kell minden futás alkalmával egy külső programot meghívnunk. Másrészről nagyobb feladatoknál sokat segíthetnek a kész külső programok, mivel képességeiket nehéz lenne megvalósítani PHP-ben. Ezekben az esetekben egy külön erre a célra készített külső alkalmazás használata javasolt.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik függvényt használjuk alkalmazások összekapcsolására?
2. Hogyan olvasunk adatokat egy folyamat kimenetéről, miután elindítottuk?
3. Hogyan írunk adatokat egy folyamat bemenetére, miután elindítottuk?
4. Az `exec()` függvény közvetlenül a böngészőbe írja a végeredményét?
5. Mit csinál a `system()` függvény a végrehajtott külső parancs kimenetével?
6. Mi a fordított irányú aposztróf visszatérési értéke?
7. Hogyan adhat ki biztonságosan a felhasználó parancsokat a rendszerhéjnak?
8. Hogyan hajthatunk végre külső CGI programot a PHP programokból?

Feladatok

1. Írjunk programot, amely a UNIX `ps` parancsának segítségével megjeleníti a böngészőben a futó folyamatokat! (Megjegyezzük, hogy nem biztonságos, ha a felhasználók futtathatják a programot!).

2. Nézzük meg a `ps` parancs súgójában a lehetséges parancssori kapcsolókat! Módosítsuk az előző programot, hogy a felhasználó a kapcsolók egy részét használhassa. Ne küldjünk ellenőrizetlen utasításokat a rendszermagnak!