© Kiskapu Kft. Minden jog fenntartva

Változó tartalom a Weben

A hagyományos programnyelvek, mint amilyen a C, vagy a Fortran, segítségünkre lehetnek a számításigényes webalkalmazások elkészítésekor.

Kezdetben, amikor az első webkiszolgálók megjelentek, elsődleges feladatuk az volt, hogy bizonyos állományokat távolról is el lehessen érni arról a gépről, amelyen futottak. Az ötlet magva, hogy a fájlok tartalmát egyszerűen át kell vinni HTTP segítségével a TCP-hálózaton keresztül. Hamar rájöttek azonban e megoldás ama korlátjára, hogy segítségével nem lehet változó tartalmat szolgáltatni. Erre kínált megoldást a webkiszolgálókhoz hozzáadott CGI-felület.
A Common Gateway Interface (Közös Átjárófelület) használata lehetőséget ad a webkiszolgálónak arra, hogy a kérelem beérkezésekor egy folyamatot elindítson. Ennek végeredményét a futtatott kód határozza meg, és ezt úgy küldi el az ügyfél böngészőprogramjának, mintha az egy állandó tartalmú fájl lenne. Azóta sokféle parancsfájlmotor és CGI-rendszer fejlődött ki, melyek egyszerűsítették a programozók feladatát és hatékonyabban kezelték a több külső szál futtatását (Perl, Python, PHP stb.). Ezeknek a nyelveknek azonban megvan az a hátrányuk, hogy futási időben értelmezettek. Nem hagyhatjuk figyelmen kívül azt sem, hogy rengeteg kódot írtak már C és Fortran (Emlékszünk még?) nyelveken, ezek bonyolult, számításigényes feladatokat oldottak meg. Nagyon hasznos egy lekérdezés alapján előálló változó tartalmat értelmezett nyelvekkel létrehozni, de nem lenne jó ötlet ezekkel megoldani képfeldolgozási vagy Fourier-átalakítási feladatokat, mert a megvalósításukhoz szükséges idő jóval meghaladná azt a válaszidőt, amit a felhasználó elvár. Két okunk is van tehát arra, hogy C vagy Fortran nyelvű CGI-alkalmazásokat írjunk. Ezeket azután természetesen gépi kódra fordíthatjuk. Egyrészt számos előre megírt forráskód létezik, másrészt bizonyos feladatok igénylik a C és a Fortran által lehetővé tett számítási sebességet.

Hogyan adja át a webkiszolgáló az adatokat a programunknak?

A HTTP-protokollal két módszer használható a böngészőprogramból a webkiszolgáló felé irányuló adatátvitelre. Az egyik a GET módszer, itt az adat valójában az URL része, általában a „?” utáni rész. A másik pedig a POST módszer, ez a böngészőprogram által visszaküldött űrlap név–érték párjaiból áll. A GET megértéséhez tekintsük a következő címet: ➲ http://www.mydomain.com/pages/external.cgi?additional-data.

A GET adatrész: additional-data

A POST működése egy csöppet bonyolultabb, vegyük szemügyre a kérést létrehozó oldal forrását, keressük meg az elküldendő űrlapot:

```
<FORM>

<INPUT TYPE="text" NAME="fld01" VALUE="val01">

<INPUT TYPE="hidden" NAME="fld02" VALUE="val02">

<INPUT TYPE="checkbox" NAME="fld03" CHECKED>

<SELECT NAME="fld04">

<OPTION VALUE="val03"> Val-03
```

➲ www.ichus.net/CGI/City/

```
<OPTION SELECTED VALUE="val04"> Val-04

<OPTION VALUE="val05"> Val-05

</SELECT>

</FORM>
```

A POST adatok

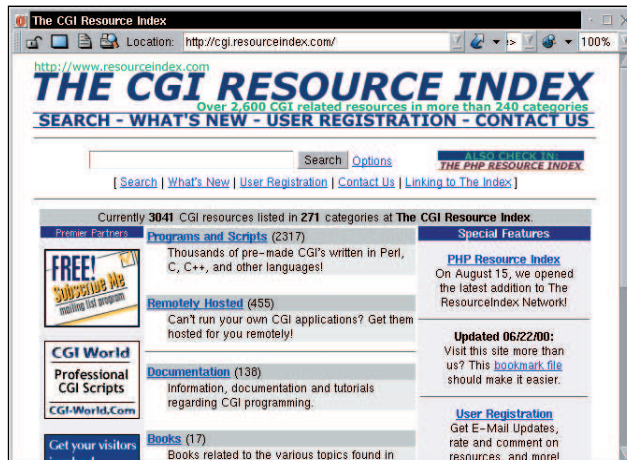
```
fld01=val01&fld02=val02&fld03=on&fld04=val04
```

Láthatjuk, hogy a POST adatok egy folytonos karakterláncban érkeznek, ahol a név és értéke között egyenlőségjel ("="), az egyes név–érték párok között „és” jel (&) található.
Valójában a POST adatok átviteli formátuma tovább bonyolódik, mert bizonyos karakterektől „meg kell védeni” a webkiszolgálót. Ilyenek például a vezérlőkarakterek és az elválasztókarakterek, ezek ugyanis megzavarnák a webkiszolgálót. Erre az a megoldás született, hogy pluszjelekkel ("+") kell lecserélnünk a szóközöket és a nem nyomtatható karaktereket, az és jelet, valamint a plusz- és az egyenlőségjelet, és a „%[0-9,A-F][0-9,A-F]” formában kódoljuk. Igen, a szóközt nemcsak a "+" jellel, hanem a "%20" kódsorozattal is leírhatjuk. Az Apache, valamint az IIS általam ismert változatai mindkettőt elfogadják.
A webböngésző másféle módon adja át a GET és a POST adatokat a külső szálnak. A GET adatok egy környezeti változóba kerülnek, ez az adott szál helyi környezetében érhető el. A környezeti változó neve "QUERY_STRING". A GET adatok megszerzése C/C++ programból rendkívül egyszerű:

```
char *pszGetData = getenv("QUERY_STRING");
```

Ez minden Unix és Microsoft fejlesztői környezetben működik.
A POST adatok a külső szál szabványos bemenetére kerülnek.

Akiknek nem ismerős a szabványos bemenet fogalma, gondoljanak arra, hogy ez olyan adatforrás, mint a billentyűzet, tehát ugyanolyan módon kell a POST adatokat feldolgozni, mint a billentyűzeten keresztül bevitt adatokat. A szabványos bemenet folyamtípusú, és a folyam természetéből fakad, hogy nem lehet tudni előre, mennyi adat vár feldolgozásra. Kézenfekvő megoldás lehet addig olvasni bájtról bájtra a folyamot, amíg véget nem ér, és ennek megfelelően menet közben növelni a tárolóhelyet. Szerencsére, a webböngészők átadnak egy további adatot, amely



➲ cgi.resourceindex.com

megmenti a fejlesztőket attól, hogy a növekvő tárolókkal bajlódjanak és lehetővé teszi, hogy lekezeljék a kivételeket, amikor egy futási időben kért memóriafoglalás nem jár sikerrel.
Amikor a webböngésző átadja a POST adatokat a külső szál szabványos bemenetére, az egész POST adathalmazt egyszerre helyezi oda, ugyanis azután, hogy elkezdtük kiolvasni az adatokat a folyamból nem fogadható további adat.
A webböngésző azt is elárulja a folyamatnak, hogy mennyi adatot helyezett a szabványos bemenetére. A szál helyi környezetében elérhető a „CONTENT_LENGTH” nevű környezeti változó, amelyben ASCII szövegként van megadva a szabványos bemeneten beolvasásra váró bájtok száma. Tehát a POST adatok megszerzése C/C++-ban egy háromlépcsős folyamat, amely minden Unix és Microsoft környezetben működik:

```
long   iContentLength = \
       atol(getenv("CONTENT_LENGTH"));

char  *szFormData = (char *) \
       malloc(iContentLength * sizeof(char));

bzero(szFormData, iContentLength * sizeof(char));

fread(szFormData, (iContentLength - 1) * \
       sizeof(char), 1, stdin);
```

Egy adott weboldal tartalmazhat egyidejűleg GET és POST adatokat is, a két módszer tehát egyszerre használható. A <FORM> tagban adható meg, hogy az űrlap tartalma GET vagy POST módszerrel kerüljön továbbításra.

Hogyan adunk vissza adatokat a webkiszolgálónak?

A weboldalról megszerzett adatokat programunk feldolgozza, és megmondja a webkiszolgálónak, hogy mit kell válaszolnia. A válasz

1. lista Számolás tízig

```
printf("HTTP/1.0 200 Okay\n"

"Content-;Type: multipart/x-;mixed-;replace;"

"boundary=SoMeRaNdOmTeXt\n"

"\n--SoMeRaNdOmTeXt\n");

for (Count = 1; Count <= 10; Count++) {

printf("Content-type:
        ➥text/html\n\n<HTML><HEAD><HEAD>"

       "<BODY><H3>%d.
        ➥frissítés</H3></BODY></HTML>\n"

       "\n--SoMeRaNdOmTeXt\n", Count);

fflush(stdout);

sleep(1);

} /* end for */

printf("Content-type:
        ➥text/html\n\n<HTML><HEAD></HEAD>"

       "<BODY><H3>Ennyi volt,
        ➥srácok!</H3></BODY></HTML>\n");

fflush(stdout);
```

© Kiskapu Kft. Minden jog fenntartva

lehet sima szöveg, HTML-dokumentum (ez a leggyakoribb), kép (rendszerint GIF vagy JPEG) vagy bármi más összetett adattípus. Ezeket az adattípusokat MIME-típusoknak nevezik, és szabványos részhalmazukat majdnem minden jelenleg használatos böngészőprogram ismeri. Az adatátvitel a programunktól a webkiszolgálóig – amennyiben az ügyfél böngészőprogramjának szánjuk – úgy zajlik, hogy a szál szabványos kimeneti folyamába írunk, pontosan akként, ahogyan a képernyőre írnánk kedvenc programnyelvünkön. A használandó adatformátum egyszerű:

```
Content-type:[szóköz][MIME-típus];
       [CR][CR][Dokumentumadatok]
```

Első lépésként a programunknak meg kell adni a MIME-típust. Ezek a következők lehetnek:

- text/plain – egyszerű szöveg, ezt írógépbetűkkel jeleníti meg pontosan úgy, ahogyan átküldjük, nem változtatja a szóközöket és a sorvégejeleket.
- text/html – szabványos HTML-dokumentum.
- image/gif – a Compuserve GIF előírásainak megfelelően kódolt kép. Itt jegyzendő meg, hogy a formátum a Lempel-Ziv (LZ77) tömörítő eljárást használja. Bővebb tájékoztatásért érdemes megnézni a ➲ http://lpf.ai.mit.edu/Patents/Gif/Gif.html oldalt és a ➲ www.unisys.com/unisys/lzw/, illetve ➲ www.unisys.com/unisys/lzw/lzw-license.asp oldalakat.
- image/jpeg – a JPEG képszabvány használatával kódolt kép.

© Kiskapu Kft. Minden jog fenntartva

Második lépésként a programunknak el kell küldenie egy pontosvesszőt (";"), amelyet két „kocsi vissza” karakternek kell követnie ("\n"). Harmadikként pedig el kell készítenie az átvitelre kerülő dokumentum törzsét. Ez lehet sima szöveg, HTML vagy olyan nyers adat, amely egy GIF vagy JPEG képet alkot.
A webkiszolgálót tehát nagyon egyszerű rávenni a válaszadásra, csak ezt kell írnunk:

```
printf("Content-type: text/html\n\n<HTML><HEAD>
        ➥</HEAD>"

"<BODY><H3>Tesztoldal</H3></BODY></HTML>\n");
```

Alapesetben mindössze ennyit kell tennünk, hogy a webkiszolgáló az ügyfélprogram kérésére válaszoljon. Természetesen az alapeset néhány trükkös kiegészítésével befolyásolhatjuk az átküldött dokumentum megjelenését. Az egyik ilyen kiegészítés a "charset=" módosító kifejezés a MIME-típus után, rögtön a „kocsi vissza” karakter előtt, ez utasítja a böngészőt a megfelelő karakterkészlet használatára (például „ISO-;9660-;1”, „ISO-;8859-;2”, „KOI-;8”, „WIN-;1250” stb.). Nézzük meg, hogyan küldhetünk át egy orosz nyelvű ellenőrzőoldalt:

```
printf("Content-type: text/html;
        ➥charset=KOI-;8\n\n"
       "<HTML><HEAD></HEAD><BODY><H3>"
       "<BODY><H3>Maya malinkayaproba
        ➥</H3></BODY></HTML>\n");
```

Folyamatos frissítések küldése a böngészőnek

Gyakran előfordul, hogy egy weboldalt olyan hosszan tartó folyamatok megfigyelésére használnak, melyek tovább tartanak annál, mint amit a webböngésző képes kivárni. Ezt a helyzetet is jól lehet kezelni a hagyományos programnyelvek segítségével. Az alapötlet az, hogy utasítjuk a webkiszolgálót a böngészővel való TCP-kapcsolat fenntartására, és bizonyos időközönként – ezt programunkban adjuk meg – új dokumentumot küldünk át.
Az itt megadott leírás az Apache webkiszolgálóra érvényes. Azért ezt választottuk, mert manapság ez a legnépszerűbb HTTP-démon a linuxos világban. Elképzelhető, hogy más HTTP-démonnal is működik a dolog, mindenkit bátorítok, hogy próbálja ki a saját kiszolgálójával, és számoljon be az eredményről. Ezek a lépések szükségesek hozzá:

1. Nevezzük át programunkat úgy, hogy az "nph-;" karakterekkel kezdődjön. Azaz ha a program neve "update.cgi" volt, változtassuk meg "nph-;update.cgi"-re.
2. Küldjük át a HTTP-fejlécet. Ezt szokásos esetben a webkiszolgáló küldené el a böngészőnek:
   `printf("HTTP/1.0 200 Okay\n");`
3. A dokumentum MIME-típusaként adjuk meg ezt: „multipart/x-mixed-replace”:

   ```
   printf("Content-;Type:multipart/x-;
          ➥mixed-;replace;"
         "boundary=SoMeRaNdOmTeXt\n");
   ```

4. Kezdeményezzük az első dokumentum átvitelét a „boundary”-ben megadott kulcsszó átadásával:

   ```
   printf("\n--SoMeRaNdOmTeXt\n");
   ```

5. Küldjük el a frissített dokumentumot. Ez egyszerűen egy dokumentum, amely addig marad a böngészőablakban, amíg ugyanazon a megnyitott kapcsolaton keresztül érkező másik frissítés fel nem váltja valamikor a jövőben. A frissítést ismét a „boundary”-ben megadott kulcsszó követi:

   ```
   printf("Content-type: text/html\n\n<HTML><HEAD>
          ➥</HEAD>"
   "<BODY><H3>%d. frissítés</H3></BODY></HTML>\n"
   "\n--SoMeRaNdOmTeXt\n", Count++);
   ```

6. Kényszerítsük ki a szabványos kimenet tárolójának kiírását:

   ```
   fflush(stdout);
   ```

7. Ismételjük meg az ötödiktől a hetedikig tartó lépéseket mindaddig, amíg az összes frissítést át nem küldtük. Az utolsó frissítéskor ne küldjük át a határoló kulcsszót, csak egyszerűen kényszerítsük ki a szabványos kimenet kiírását és lépjünk ki. Ezáltal az utolsó frissítés eredménye a böngészőablakban marad a kilépés után is.

A kiszolgálóoldali kényszerített frissítést mutatja be az *1. listán* olvasható egyszerű példa, amely egytől tízig számol a böngészőablakban. Minden frissítés között egy másodperc telik el.
Ennek magyarázatához meg kell érteni a kiszolgáló háttérben végzett tevékenységét. Eddig a programunk kimenetét leellenőrizte a kiszolgáló, például a megfelelő MIME-típus, a megfelelő elválasztójel stb. tekintetében, majd mielőtt elküldte volna a böngészőprogramnak, átküldött egy HTTP-fejlécet. Azért, hogy jobban befolyásolhassuk a webkiszolgáló és a böngészőprogram párbeszédét, meg kell kérnünk a webkiszolgálót, hogy ne végezze el az ellenőrzéseket, és ne küldjön HTTP-fejlécet. Ezért adtuk az „nph-”-t a program fájlnevéhez, mert ez azt jelenti: No Parsed Headers, azaz nem dolgozza fel a fejlécet. Ha a programunk neve „nph-”-val kezdődik, a webkiszolgáló úgy veszi, hogy a program maga gondoskodik a szükséges ellenőrzések elvégzéséről és a fejléc átküldéséről, ami pedig szokásos esetben a webkiszolgáló feladata lenne. A webkiszolgáló csak a böngészővel felépített TCP-kapcsolat fenntartásáért felelős, és a programunk szabványos kimeneti folyamát egy az egyben átküldi böngészőnek ezen a TCP-csatornán keresztül. Most már világos, miért kellett a második lépésben elküldeni a HTTP-fejlécet.
Következő lépésben közölnünk kell a böngészővel, hogy folyamatos frissítésekre számítson, ne csak egyszeri adatcsomagra, ezért nem szabad lezárni a TCP-csatornát az első dokumentum átvitele után. Ezt a dokumentum MIME-típusának megfelelő beállításával (`multipart/x-;mixed-;replace`) érhetjük el. Ezenkívül a böngészővel tudatni kell, hogyan különítheti el az egyes dokumentumokat a sok dokumentumból álló folyamban. Ezt a MIME-típus megadását követő `boundary=SoMeRaNdOmTeXt` módosító kifejezéssel oldhatjuk meg. Ezzel azt mondjuk a webböngészőnek, ha bármikor találkozik a `--SoMeRaNdOmTeXt` bájtsorozattal a bemeneti folyamban, meg kell állnia, és feltételezni, hogy az ezt követő adatok egy új dokumentum részei, amelyek az előzőt váltják fel a böngészőablakban.
A dokumentum végét és a következő dokumentum elejét elválasztó karakterláncot határoló kulcsszónak (vagy röviden határszónak) nevezik, és általában sokkal bonyolultabb és hosszabb, mint az itt megadott példa. Többnyire 50–60 bájt hosszú számokat és betűket tartalmazó véletlen karakterláncot szoktak választani erre a célra.
A karakterláncnak elég hosszúnak és elég véletlenszerűnek kell lennie ahhoz, hogy elhanyagolható legyen annak az esélye, hogy a dokumentum törzsében valahol véletlenül előfordul.
Végül, miután a dokumentumot kiírtuk a szabványos kimenetre, és a határszót is kiírtuk utána, ki kell kényszeríteni a szabványos kime-

4. lista CGI keretrendszer hagyományos programnyelven

```
void GenerateHtmlDocument () {

char  *TextField     = GetFormStringValue
                         ("TextField");

int     NumericField = GetFormIntegerValue
                         ("IntegerField");

float   FloatField    = GetFormFloatValue
                         ("FloatField");

printf("<HTML><HEAD><HEAD><BODY><H3><CENTER>"

       "TextField     = [%s] <BR>"

       "NumericField = [%d] <BR>"

       "FloatField    = [%f] <BR>"

       "<CENTER><H3><BODY><HTML>\n",

       TextField, NumericField, FloatField);

} /* end function GenerateHtmlDocument() */

int main (int argc, char **argv) {

       printf("Content-Type: text/html;\n\n");

       GenerateHtmlDocument();

       ReleaseFormData();

} /* end of main() function */
```

net tárolójának tényleges kiírását, mert csak így érhetik el az adatok a böngészőprogramot. Ha ezt nem tennénk, a démon az adatokat nem küldené el mindaddig, amíg a folyam átmeneti tárolója – ennek mérete függ a használt operációs rendszertől – be nem telik, és a kiürítést el nem végzi az operációs rendszer.

Biztonsági kérdések és a webkiszolgáló bemenetének értelmezése

A legtöbben a biztonsági kérdések miatt rettennek vissza webes programok hagyományos programnyelveken történő megírásától. Valószínűleg a biztonság megteremtése volt a Perl és a PHP nyelvek fejlesztésének egyik legfontosabb célja. Nehéz olyan alkalmazást írni, amely elég ügyes ahhoz, hogy az összes lehetséges támadást kivédje, miközben az adatokat csak környezeti változókban és a szabványos bemeneti folyamban adhatja át a böngésző a programnak.

Az első megoldásra váró nehéz feladat az adatok egyszerű és memóriatakarékos feldolgozása, tehát hogy egyszerűen kiválaszthassuk a keresett mezőt, és az adatot egy lépésben megszerezhessük. Ezenkívül gondolnunk kell bizonyos biztonsági gondokra, például az adattúlcsordulásra, amit a rosszindulatú böngészőprogram okoz, felülírva az alkalmazás memóriáját a túlcsordult adattal vagy akár meg is bénítva a szolgáltatást.

Bemutatunk egy függvénycsaládot, amely pontosan az említett egylépéses adatelérést valósítja meg a POST adatok feldolgozására és a biztonságra egyaránt ügyelve. A példa C nyelvű, de könnyen átvihető Fortranra vagy C++-ra:

```
char   *TextField     = GetFormStringValue
                          ("TextField");

int     NumericField = GetFormIntegerValue
                          ("IntegerField");
float   FloatField    = GetFormFloatValue
                          ("FloatField");
```

A függvények forrását a 2. lista mutatja be, az ezeket támogató függvényeket pedig a 3. lista tartalmazza. Terjedelmi okokból ezeket a listákat nem közöljük az újságban, de letölthetők az ➲ ftp://ftp.ssc.com/pub/lj/listings/issue82 címről. Az összes megadott függvényt kipróbáltuk Unix és Windows környezetben is, jól működtek, és ellenállóak a tárolók alulcsordulásával vagy túlcsordulásával szemben. Minden függvény az első meghívásakor lefoglal egy memóriaterületet, és itt elhelyezi a feldolgozott POST adatokat. Ez megmarad a memóriában, és a következő függvényhívás alkalmával egyszerű lineáris kereséssel végigmegyünk ezen a memóriaterületen a kívánt mezők után kutatva. A memóriafoglalásra csak egyszer kerül sor, és a különleges karakterek átalakítása ezen a memóriaterületen zajlik sorfolytonosan, ugyanis erre a célra nem használunk más ideiglenes tárterületet.

Mivel a bemutatott példa C nyelvű, melyben nincs a C++-ból ismert automatikus megszüntető (destructor), a program kilépése előtt egy takarítófüggvényt szükséges meghívni. Ez a ReleaseFormData(), mely nélkülözhetetlen a futási időben lefoglalt tárterület felszabadításához. Ha ezeket a függvényeket C++ osztályokká alakítjuk, egyszerűen csak meg kell hívni ezt a függvényt a POST adatok elérését megvalósító osztály megszüntetőjében. A *4. listán* bemutatunk egy egyszerű keretrendszert egy hagyományos programnyelven megírt CGI-programhoz.

Miről lesz szó a jövőben?

Természetesen csak felületesen érintettük, mi történik akkor, ha a gyors és hatékony nyelveket (pl.: C/C++) használjuk webes alkalmazások fejlesztésére, és megszabadulunk attól a tehertételtől, amit az értelmezett nyelvek feldolgozása jelent. Könnyen belátható, miért kell még az alábbiakról is szót ejtenünk:

- A helyi fájlrendszer használata a CGI-programjaink állapotának tárolására.
- Miért tárolható az állapot a helyi fájlrendszeren Linux alatt lemeztúlterhelés veszélye nélkül, ami más operációs rendszerek használata esetén felléphet.
- Sütik létrehozása, módosítása és megsemmisítése az ügyfél böngészőprogramjában CGI-programok segítségével.
- Biztonsági beállítások, hogy csak a mi CGI-programunk érhesse el az állapotadatokat a helyi fájlrendszeren, és senki másé.
- Pehelysúlyú szálak és a gyors CGI.

Dan Teodor tanácsadóként dolgozik a PriceWaterhouseCoopersnél a texasi Houstonban. Megszállott Linux-rajongó középiskolás évei és a Slackware 0.x rendszermag-kiadások óta. Nagyratörő terveket szövöget a webalkalmazások elterjesztéséről és a sajátos üzleti modellekről, miközben arról ábrándozik, hogy Új-Mexikóban síel, és minden évben alapít egy csődbe menő dot-com vállalatot.

© Kiskapu Kft. Minden jog fenntartva