# PHP nyomkövetők (1. rész)

„Kössük össze a Föld számítógépeit egy hálózattá... A Föld forgásától a vezetékeken hamarosan megindul a szoftverek áramlása” (Eric S. Raymond: A katedrális és a bazár). Sorozatunkban a PHP nyelvű programok hibajavításáról, teljesítményelemzéséről, gyenge pontjainak felderítéséről lesz szó, valamint különböző alkalmazásokról, amik ezt megkönnyíthetik.

A *PHP* nyelv egyik fő előnye és sikerének kulcsa (valamint sebezhetőségének gyökere), hogy szabad volta és egyszerűsége miatt amatőrök is nekiállhatnak egy-egy szkript megírásának. Így voltam ezzel magam is. Komoly motivációt szolgáltat e nyelv megtanulásához, hogy az ingyenesen elhelyezhető/felhasználható webes programok gyümölcseit sokan élvezhetik (lásd ***Web 2.0, Linuxvilág #63.***). A pozitív visszajelzés a szabad szoftver „fizetőeszköze”. Szerencsés, ha elég idő áll rendelkezésre: a programok nyugodt megtervezését ugyanis semmi nem pótolja; ha segítőkész emberek és megfelelő feladatok vannak a (virtuális) környezetben, a kezdőkből idővel óhatatlanul haladók lesznek egy-egy témában – azzal együtt, hogy az autodidakta módszereknek megvannak a maguk hátrányai, maradnak bőven fehér foltok a módszerek terén.

Hogyan és milyen alkalmazások segítségével lehet elmélyedni, hibát keresni *PHP* programokban, azaz hogyan lehet „debuggolni” őket? A ***debug*** szótő valószínűleg a számítógépek (h)őskorszakáig vezet vissza, amikor egy hatalmas termet betöltő gépezet elektroncsövei között kellett hibá(sa)t keresni, és az egyik ilyen hiba oka egy odakerült bogár (***bug***) volt. A kényelmes nyomkövetés azonban nemcsak a szó szoros értelmében vett hibák felkutatására jó, hanem egy-egy „újrahasznosított” (például *PECL* vagy *PEAR*) program hatékonyabb, gyorsabb megismerésére is. Ebben az esetben (és a fejlesztés bizonyos szituációiban) lehet létjogosultsága annak, hogy a webszerver a helyi gépünkön legyen. Általában azonban olyan helyzeteket fogunk vizsgálni, amikor a webszerver egy távoli gépen helyezkedik el, ahová csak biztonságos (***SSH***) kapcsolaton át van lehetőségünk belépni, sőt, ahonnan esetleg a rendszergazda le is tiltotta a belső hálózatunk felé irányuló *SSH*-kapcsolatok nyitásának lehetőségét is. A nyomkövetők másik fontos szolgáltatása általában a teljesítményelemzés, azaz a szűk keresztmetszetek, időigényes részletek keresése, a *„profiler”* mérések végzése.

## A „házi” hibakeresés módszerei

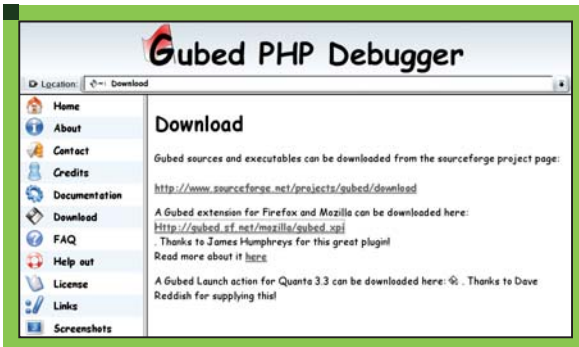
Legtöbbünknek ilyesfélék jutnak eszébe, amikor valami kisebb(nek hitt) hiba merül fel egy *PHP* programban: tegyünk be egy `echo 'itt tartunk';` részletet, hogy eljut-e odáig az interpreter. Írassunk ki egy-egy változót, például a `print_r($tomb);` vagy a `var_dump($egyeb);` segítségével. Ez utóbbi az értékek mellett típusokat is mutat, bár pont emiatt zsúfoltabb, kellemetlenebb is a látvány. Ha nem akarunk állandóan „oldal forrását” kérni, akkor célszerű odatenni ezen parancsok elé és után egy `<pre>` és `</pre>` *HTML*-kulcsszót. Ezek a kézi megoldások gyorsak, ha tudjuk, hol kell keresni a hibát, verziófüggetlenek, és nem igényelnek rendszergazdai jogokat.

Azonban ezeknek hátrányai is vannak: átfogó vizsgálathoz sok helyre kell beszúrni a fenti parancsokat, ami veszélyes lehet, ha elfelejtjük kiszedni őket, és a „kikommentezés” (megjegyzéssé változtatás) zavarja a kód képét. (Bár ez előny is lehet, mert a távoli jövőben is láthatjuk majd, hogy itt már kerestünk hibát korábban.) Nemigen lehet futás közben próbálkozni (pédául egy-egy ciklusbeli vagy bemeneti) változó megváltoztatásával. Ami talán a legnagyobb hátrány, hogy ezekkel a módszerekkel meghiúsulhat a *HTTP*-fejléc küldése, ami átirányításkor (redirect) vagy munkamenetek, illetve „sütik” (cookie-k) szervezésénél jelenthet gondot.

## „Célszerszámok” a PHP-n belül

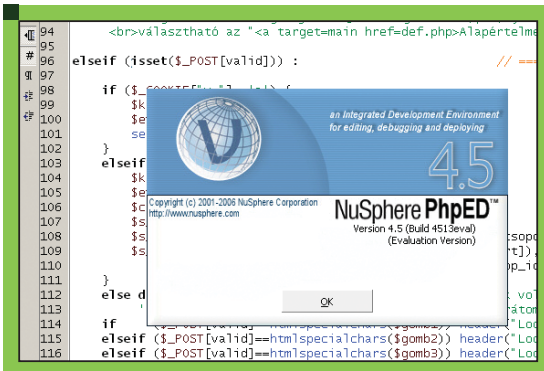
Körmönfontabb hibakeresésre is lehetőség van PHP függvények segítségével: `error_log()`, `set_error_handler()`, `user_error()`, `trigger_error()`, `assert()` stb. Ez utóbbi különösen jól használható. Érzékenysége (hogy megálljon-e a program egy-egy hibára és milyenre) jól skálázható a ***php.ini*** konfigurációs fájlban; meghálálja a megismerésére szánt időt. Célszerű a ***php.ini*** fájlt úgy beállítani, hogy fejlesztés, tesztelés közben kapjunk hírt mindenről (dönthetünk: képernyőre vagy naplófájlba?); az éles rendszer azonban legyen a lehető legcsöndesebb a hibákat illetően (és a hibákat feltétlenül naplófájl fogadja). Tegye mindezt természetesen úgy, hogy maga a *PHP* kód ugyanaz maradhasson, amikor a tesztkörnyezetből átkerül az éles környezetbe.

A *PHP*-t maximális hibajelentési szinten használva – az `error_reporting=E_ALL` beállításával – figyelmeztetést kapunk mindenről, ami gondot




*1. ábra* Gubed


*2. ábra* Quanta+




*3. ábra* Nusphere::PhpED

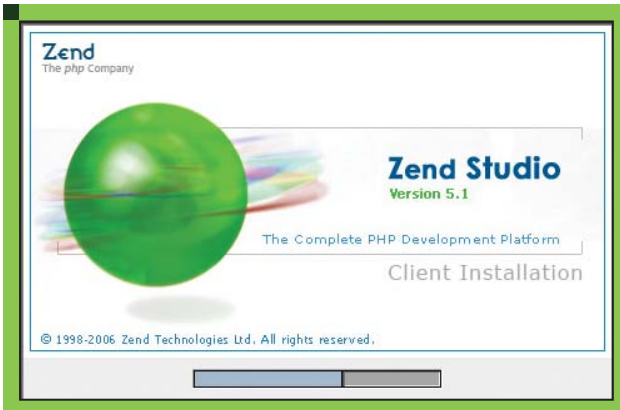

*4. ábra* ActiveState::Komodo


*5. ábra* Xored::TruStudio

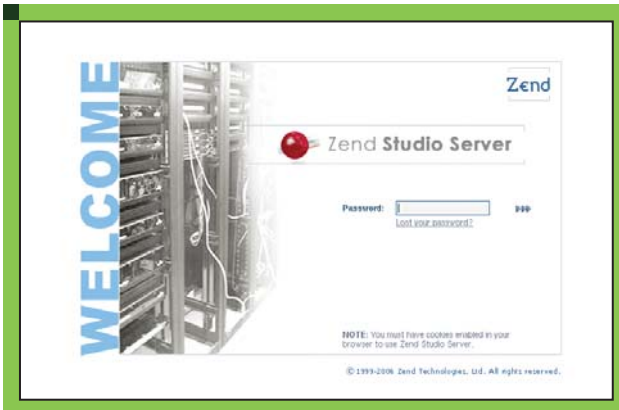
okozhat, például az előzetes érték nélküli, definiálatlan változókról, amikor is hibás adatokkal dolgozhat a program. Ez a sajátosság akár programból is állítható az `error_reporting()` függvénnyel – azonban veszélyes lehet, ha így felejtjük. Ha a program már kész bevezetésére, akkor az `E_NONE`-t használva teljesen leszigetelhető a kód a további vizsgálatoktól.

## Miért is jó nyomkövetőt használni?

Az interaktív nyomkövetők esetén kényelmesen kézben tartható a futtatás. Parancsról parancsra léptethetünk, a felesleges blokkokat átugorhatjuk, töréspontokat tudunk elhelyezni a kódban, akár feltételhez kötve őket. A program belső állapotváltozásainak lépésenkénti nyomon követése szinte csak így lehetséges. Könnyen inicializálható a bemenet (`$_GET`, `$_POST`, `$argv` változók), működés közben módosítható a változókörnyezet, a hibaüzenetek és naplók a kimenettől elkülönülnek. Az érintetlen kód gyümölcse az eredeti funkcionalitás – ezt sehogy máshogy nem lehet elérni „házi módszerekkel”. Hátrányként jelentkeznek az előkészítés nyűgös teendői: a webszerveren telepíteni kell valamit – általában webszerver-modul(oka)t, *.so* fájl(ok) formájában (rendszergazdaként), mint a *PhpED* és az *Xdebug* esetében, vagy egy kisebb *PHP*-programrendszert például a *Gubed* esetén (ennek előnye, hogy ehhez nem kell rendszergazdának lenni, és nem változik meg a *PHP*-értelmező a nyomkövetés miatt). Kliens alkalmazás is kell a helyi gépen. Az az ideális eset, ha ez a kliensprogram ugyanaz, mint a fejlesztőkörnyezet maga (például a *Komodo* vagy a *Quanta*), de lehetnek előnyei puritánabb kliensprogramoknak vagy akár parancssori klienseknek is. Egy kényelmes *PHP* fejlesztőkörnyezettől azt várjuk, hogy lehessen benne kódot szerkeszteni (szintaxis-színezéssel, navigálással, kódkiegészítéssel, esetleg az összetartozó egységek takarásával/kibontásával), lehessen nyomkövetni (helyi vagy távoli gépen), legyen benne teljesítményelemző (profiler) és kódelemző alkalmazás, lehessen fel- és letölteni a távoli webszerverre/ről fájlokat (*scp*-vel), tudjon kapcsolatot tartani egyéb alkalmazásokkal, mint amilyenek például a *CSS* vagy *HTML* érvényesítők, a *PHPdocumentor* stb. A kód lefedettségét, azaz a futás közben érintett (és nem érintett!) részek feltérképezését (*code coverage*) hasznos lehet látni nagyobb programoknál. Úri huncutság, de segíthet időnként az incidens vezérelt („*just in time*”, *JIT*)




**6. ábra** Zend Studio Kliens




**7. ábra** A Zend Studio Szerver webfelületen konfigurálható



nyomkövetés, azaz: csak akkor nyílik meg a megfelelő kódrészlet a fejlesztőkörnyezetben, ha valami hiba lépett föl a futás során. Előnyös lehet még a munkamenet alapú (*per session*) hibakeresés. Nem nagy baj az sem, ha a legszükségesebb (*PHP*, *HTML*, *SQL*) kézikönyvek egy kattintással előjönnek, valamint az oktatóanyagok is hasznosnak bizonyulnak a kezdők számára.

A futtatás közben történő kódmódosítás még a jövő zenéje. Az is az igazsághoz tartozik, hogy nyomkövetés közben jóval lassabb a futás – ez azonban többnyire nem okoz gondot, és ezen valamennyit lehet segíteni jól elhelyezett (feltételes) töréspontokkal.

A hibák kijavításához kell némi gyakorlat, de ez viszonylag hamar megszerezhető. Érdemes először minden releváns hibát megszüntetni, és csak utána rágódni a kisebbeken. Arra is ráérez az ember egy idő után, hogy mekkora lépésekben érdemes a kódot hizlalni, hogy az új részletek hibái még átlátható mennyiségűek legyenek. Jó önismerettel megfelelő középutat találhatunk, hogy ne pazaroljuk az időnket a túl gyakori kipróbálgatással, de ne állítsunk elő akkora programot sem, aminek a hibái már átláthatatlan mértéket öltenek, és a kijavításukhoz szükséges idő igencsak megnő. Ilyenkor a részletes hibajelentés nagyon sokat ér.

Hasznos, ha az érvényes és hibás tesztadatokból teszthalmazokat tudunk létrehozni, és az ezekkel való tesztelést megfelelően dokumentáljuk. (Nagyobb programok dokumentálatlan tesztelése többnyire egyenértékű azzal, mintha semmit sem csináltunk volna.)

## Áttekintés a nyomkövetőkről

A *PHP3* még tartalmazott hálózat-alapú nyomkövető-támogatást, de a *PHP4* és *PHP5* már nem. Ehelyett külső nyomkövetők használata ajánlott:

- A legegyszerűbb a „debuG" tükörszavaként elnevezett *Gubed* (például a *Quanta Plus* fejlesztőkörnyezet által felajánlott integrált kliensprogrammal) ➲ gubed.mccabe.nu, ➲ quanta.sourceforge.net
- jó lehetőségeket kínál a *DBG*, de a rá épülő legjobb fejlesztőkörnyezet, a *NuSphere::PhpED* sajnos a *Microsoft Windows*-os verzióban (és ehhez tartozó funkcionalitásban) jóval előrébb tart, mint a Linuxos változatban – és ráadásul időkorlátos ➲ dd.cron.ru/dbg
- a teljesítményelemzésben jár élen az *Advanced PHP Debugger* (*APD*) ➲ pecl.php.net/package/apd
- az (időkorlátos) *ActiveState::Komodo* fejlesztőkörnyezetet használhatjuk kliensprogramként az *Xdebug*-hoz, mely működését tekintve a PHP 3-hoz tartozó hibakereső utódjának tekinthető; az *Xdebug* honlapja mintaszerűen egyszerű és igényes. ➲ www.xdebug.org, ➲ www.activestate.com/Products/Komodo
- szintén *Xdebug*-ot (és *Eclipse*-et) használ java alapon a *Xored::TruStudio*; sajnos licenc után érdeklődik néhány percenként, és csak CGI interpreterrel ajánlja fel a nyomkövetést (azaz, amennyire láttam, nem tud távoli szerveren nyomkövetni) ➲ www.xored.com/trustudio/download
- a legjobban kidolgozott és a legszélesebb igényeknek is megfelel az időkorlát nélküli, *„Personal Licence"*-cel használható, java alapú *Zend Studio* és a hozzá tartozó (Apache-modulként működő, *.so* fájlokat tartalmazó) szerver ➲ www.zend.com/store/products/zend-studio

A ➲ www.php-editors.com értékelése szerint maximális pontot kapott a *Nusphere::PhpED*, az *ActiveState::Komodo* és a *Zend Studio*. Saját tapasztalatom is e hárommal kapcsolatban volt a legjobb, pontosabban az utóbbi kettővel.

A következő részekben a fent felsoroltakat fogjuk részletesen megvizsgálni.

*Szabó Zoltán*
Három gyermekével és feleségével Pannonhalmán él. Tíz éve kísérletezik a Linuxszal. Matematikát és informatikát tanít, diákotthonban keseríti a rábízottak életét. Szívügye a PHP és a PostgreSQL.
(szz@freemail.hu)

### KAPCSOLÓDÓ CÍMEK

Írásom közben felhasználtam Papp Győző előadását:
➲ phpconf.hu/2004/media/eloadasok/chemotox.pdf

➲ hu.php.net/assert-options

➲ www.sitepoint.com/article/bug-detection-php-assertions