Írjunk hibátlan programot!

Bár e gondolat túlzó, a jellemző memóriahibákra azonnal rámutat a KDE-csoport legújabb csodája, a Valgrind.

A dinamikus memóriakezelés alapvető fontosságú szerepet játszik napjaink alkalmazásaiban. Jelenleg elképzelhetetlen C/C++ programot írni a `malloc()` vagy `new()` használata nélkül. E függvények alkalmazásával nem nehéz elérni, hogy az egyetlen korlát az elérhető memória mérete legyen. Ám az áldásos tulajdonságok mellett hamar napvilágot láttak a gondatlanságból eredő káros mellékhatások is: a tömbök túlcímzése, lefoglalt területek többszöri felszabadítása vagy a mutatóik elvesztése, ezek mind-mind olyan gondok, amelyek nem is biztos, hogy azonnal kiderülnek. Sőt, még ha rá is jövünk, hogy baj van, a gond forrását akkor is órákba telhet megtalálni, mivel többnyire egyáltalán nem ott helyezkednek el, ahol észleltük őket.

A Valgrind egy nyílt forrású segédeszköz a memóriakezelési hibák felderítésére. Az x86-os processzorra fordított programokban figyeli a memóriaszivárgást és a helytelen elérést. ***Julian Seward*** fejlesztette. Jelenleg ez a projekt a ➲ http://www.developer.kde.org/~sewardj/ címen érhető el. A Valgrind telepítésének előfeltétele a legalább 2.2-es változatszámú rendszermag, illetve a 2.1-es vagy 2.2-es Glibc megléte.

A Valgrind valójában egy dinamikusan összeépített könyvtár, ez a parancsfájl végzi az összeépítést. A szükséges függvénykönyvtárak a */usr/local/lib* alatt találhatók. A Valgrind az alábbi hibákat képes felderíteni:

- olyan memóriaterület olvasása, amely nem kapott kezdeti értéket;
- felszabadított terület olvasása, illetve írása;
- túlcímzett terület olvasása, illetve írása;
- a verem nem megfelelő területeinek olvasása, illetve írása;
- memóriaszivárgás;
- rosszul használt `malloc/new/new[]` és `free/delete/delete[]` páros;
- a POSIX `pthread` API helytelen használata.

Ezek a hibák többnyire összeomláshoz vezetnek, ám sokszor nagyon sokáig rejtve maradnak a felhasználók elől. Csak bizonyos körülmények együttálláskor kerülnek felszínre, emiatt nagyon veszélyesek. A Valgrind közvetlenül a futtatandó állományt használja a vizsgálathoz. Eldönti, hogy a program módosítása szükséges-e a memóriakezelés szempontjából helyes használathoz, és rámutat a hiányosságokra. Mindezt valójában az x86-os processzor emulálásával teszi.

A fentebb említett tulajdonságok következtében a Valgrind használata meglepően egyszerű. Első megközelítésben elég csak a parancs elé beszúrni a `valgrind` szót. A futtatandó program és kapcsolói előtt a Valgrindnek is megadhatunk bizonyos értékeket, ezekkel befolyásolhatjuk a vizsgálatot. A továbbiakban jellegzetes programozói hibákat fogok elkövetni és a Valgrind segítségével derítem fel őket. A példák természetesen olyanok lesznek, amelyekről üvölt, hogy hibásak, de most tegyünk úgy, mintha nem is lenne olyan szemet szúróak. A hangsúly most a hiba típusán van. Készítsünk mi is olyan feltételt, amelyben egy kezdőérték nélküli változó áll. Íme:

```
#include <stdio.h>
int main () {
        signed char a;
        if (a > 0) printf ("hello\n");
        return 0;
}
```

Amennyiben gcc-t használjuk, a -Wall segítségével előcsalogathatjuk a fordítás alatt az összes figyelmeztető üzenetet. Az a szép ebben a csúnyaságban, hogy a gcc még a -Wall-lal is rendben találja, azaz szó nélkül lefordítja. Az, hogy az elkészült program kiírja-e, hogy hello, vagy sem, nagyban függ az ózonlyuk méretétől, és a New York-i tőzsde pillanatnyi állásától. Nézzük, mit mond rá a Valgrind:

```
==3220== Conditional jump or move depends on
➥uninitialised value(s)
==3220==    at 0x80483FA: main (elso.c:4)
==3220==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3220==    by 0x8048330: (within
➥/home/bw/prog/c/valgrind/elso)
```

A sor elején látható szám az indított folyamat azonosítója (PID). Az elkövetett hiba ismertetése után a hívás helye is látható. Amennyiben a `-g` kapcsolóval fordítottuk a programot (debug), akkor még a forrásállományban kérdéses sor sorszá-

mát is megadja. Ez az alapvető hiba más formában is testet ölthet. Például egy tömböt indexelünk a változóval egy számlálós ciklusban. Gondoljuk csak végig a következményeket. Lássunk példát a dinamikus memóriafoglalásra is! Ezúttal túlcímzünk egy tömböt:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
        int *p, a;
        p = (int *) malloc (sizeof (int) * 10);
        p[10] = 1;
        a = p[10];
        free (p);
        return 0;
}*
```

Mivel C-ben a tömbök indexelése 0-tól indul a fenti példában a 9-es a legmagasabb indexű elem. A `p[10]`-zel láthatóan túlcímeztük a dinamikusan foglalt tömbünket. Linux alatt elindítva egyetlen árva hibaüzenetet sem kaptam. Szó nélkül lefutott a program. Doktor Valgrind itt is segít *(1. lista)*.

```
==3269== Invalid write of size 4
==3269==    at 0x804844E: main (harmadik.c:6)
==3269==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3269==    by 0x8048370: (within
➥/home/bw/prog/c/valgrind/harmadik)
==3269==    Address 0x4107004C is 0 bytes after
➥a block of size 40 alloc'd
==3269==    at 0x40027B88: malloc
➥(vg_replace_malloc.c:153)
==3269==    by 0x804843F: main (harmadik.c:5)
==3269==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3269==    by 0x8048370: (within
➥/home/bw/prog/c/valgrind/harmadik)
==3269==
==3269== Invalid read of size 4
==3269==    at 0x804845A: main (harmadik.c:7)
==3269==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3269==    by 0x8048370: (within
➥/home/bw/prog/c/valgrind/harmadik)
==3269==    Address 0x4107004C is 0 bytes after
➥a block of size 40 alloc'd
==3269==    at 0x40027B88: malloc
➥(vg_replace_malloc.c:153)
==3269==    by 0x804843F: main (harmadik.c:5)
==3269==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3269==    by 0x8048370: (within
➥/home/bw/prog/c/valgrind/harmadik)
```

Javítsuk ki fenti programunkat. Ám mialatt jóízűen szürcsölgetjük kávénkat, egy vicces kolléga belenyúlt a munkánkba, és a biztonság kedvéért még egyszer felszabadította a memóriaterületet:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
        int *p, a;
        p = (int *) malloc (sizeof (int) * 10);
        p[9] = 1;
        a = p[9];
        free (p);
        free (p);
        return 0;
}
```

Erre már a Linux is dobott egy „Segmentation fault” üzenetet. Ez az a pont, amikor egy több ezer soros program esetén Valgrind nélkül nem marad más, mint a `gdb` (GNU debugger) alkalmazása. A változókövetés, töréspontok elhelyezése, és lépegetés a sorok között hadművelet több órát is elrabolhat drága időnkből. Ehelyett inkább a következőket tegyük:

```
==3296== Invalid free() / delete / delete[]
==3296==    at 0x40027E7A: free
➥(vg_replace_malloc.c:231)
==3296==    by 0x8048479: main (harmadik.c:9)
==3296==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3296==    by 0x8048370: (within
➥/home/bw/prog/c/valgrind/harmadik)
==3296==    Address 0x41070024 is 0 bytes inside
➥a block of size 40 free'd
==3296==    at 0x40027E7A: free
➥(vg_replace_malloc.c:231)
==3296==    by 0x804846A: main (harmadik.c:8)
==3296==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3296==    by 0x8048370: (within
➥/home/bw/prog/c/valgrind/harmadik)
```

A Valgrind hasonlóan jól felismeri, ha a `malloc()` mellett nem `free()`-t, hanem `delete()`-t használunk. A memóriafoglalással kapcsolatos függvények bármely helytelen variációja esetén jelez. Még abban az esetben is kapunk hibaüzenetet, ha a lefoglalt és felszabadított területek megegyeznek, de rossz volt a függvényhívás.

Utoljára, a rám legnagyobb hatást gyakorolt tulajdonságot, a memóriaszivárgás felismerését hagytam. Akadnak olyan C-ről szóló könyvek, amelyekben azt tanácsolják, hogy ha „nem tudjuk, hol vagyunk éppen”, inkább ne használjuk a `free()`-t, mert a többszöri felszabadítás rosszabb, mintha egyszer sem tennénk. Ez Linux alatt még működhet is, mert többé-kevésbé eltakarítja a fel nem szabadított területeket. Ám más rendszerek esetén ez a memória szabályos felfalásához, rosszabb esetben a teljes összeomláshoz vezethet. Azért, mert a rendszertől nem kapunk hibaüzenetet, amiért nem szabadítottuk fel a memóriát, még nem járunk el helyesen.

Készítsünk egy memóriafalót: ugyanazt a mutatót fogjuk használni többszöri `malloc()` hívásra, természetesen `free()` nélkül. Érdekes kipróbálni, hogy egy terminálablakban futtatjuk memóriafalónkat, mellette pedig egy `xosview`-t figyelünk. Érdemes elüldögélni előtte.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
     int hamm = sizeof (int) * 1024, *p, i;
     for (i = 1; i <= 5; i++) {
        if (NULL == (p = (int *) malloc (hamm)))
➥break;
        printf ("Lenyelve: %d byte.\n", i * hamm);
     }
     return 0;
}
```

Nem kell megijedni, ugyanis ha Linux alatt futtatjuk, több mint valószínű, hogy visszanyerjük a memóriát. A Valgrind mindenesetre nem állja meg szó nélkül.
A `--leak-check=yes` beállítás használata esetén az alábbi jelentést kapjuk:

```
==3400== 4096 bytes in 1 blocks are possibly lost
➥in loss record 1 of 2
==3400==    at 0x40027B88: malloc
➥(vg_replace_malloc.c:153)
==3400==    by 0x804845B: main (negyedik.c:6)
==3400==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3400==    by 0x8048370: (within
/home/bw/prog/c/valgrind/negyedik)
==3400==
==3400== 16384 bytes in 4 blocks are definitely
➥lost in loss record 2 of 2
==3400==    at 0x40027B88: malloc
➥(vg_replace_malloc.c:153)
==3400==    by 0x804845B: main (negyedik.c:6)
==3400==    by 0x4025614E: __libc_start_main
➥(in /lib/libc-2.2.5.so)
==3400==    by 0x8048370: (within
➥/home/bw/prog/c/valgrind/negyedik)
```

Miként az látható, lesznek feltehetően és biztosan elvesztett területek. Ötször futott le a ciklusunk, ez azt jelenti, hogy négyszer írt felül egy olyan mutatót, ami értékes területre mutatott. Ez a négy biztosan elvesztett terület. A program végére még maradt egy mutató az adatterületre, így nem biztos, hogy ez elveszett.
Mindent egybevetve nagyon hatékony és jól használható eszköz a Valgrind. Nem hallottál még róla, és azt gondolod, senki nem használ egy ilyen „névtelen” programot? Látogass el a honlapjára és nézd meg, mely nagy alkalmazások készítésénél vették igénybe a Valgrindet. Az OpenOffice.org-tól a Mozillán keresztül számtalan nagy csomag esetén használják hibakeresésre. Itt az idő, hogy megismerj egy olyan eszközt, melynek megtanulása tíz perc, de órányi keserves keresgélést takaríthatsz meg vele. Sok szerencsét és kísérletezgető kedvet a programozáshoz.

Fülöp Balázs