Memóriaszivárgások beágyazott rendszereknél

Cal írásában három, az általános memóriakezelési hibák felderítésére alkalmas, könnyen használható eszközről olvashatunk.

A beágyazott rendszerek fejlesztésével kapcsolatos gondok egyike a memóriaszivárgások felderítése. Három programot ajánlok erre a célra. Segítségükkel nem a rendszermag, hanem az alkalmazások memóriaszivárgásai deríthetők fel. Kettő közülük – az `mtrace` és a `dmalloc` – a MontaVista Linux Professional Edition 2.1-es terjesztésnek is részét képezi. A harmadik – a `memwatch` – az Internetről tölthető le (lásd a ***Kapcsolódó címeket***).

A C és C++ nyelvekben programozó fejlesztők dinamikus memória-hozzárendelést használnak. Kellő körültekintés nélkül alkalmazva azonban gondok merülhetnek fel a memóriakezelés környékén, csökkenhet a gép teljesítménye, és megjósolhatatlan lesz az alkalmazás futásának eredménye, vagy összeomolhat a rendszer.

A memóriaszivárgást okozó hibák egy része a lefoglalt területen kívül eső memória olvasásával vagy írásával, illetve a már felszabadított memória újbóli felszabadításával kapcsolatos. Memóriaszivárgás akkor történik, ha a memóriát lefoglaljuk, ám a használat után a felszabadítása nem történik meg; vagy ha a memóriafoglalás mutatóját töröljük, és emiatt a memóriaterületet többé nem lehet használni. A memóriaszivárgás a gyakoribbá váló lapozás miatt csökkenti a gép teljesítményét, illetve idővel a memória elfogyását és az alkalmazás összeomlását okozhatja. A hozzáférési hibák az adatok módosulását okozhatják, ezek miatt a program futásának eredménye megjósolhatatlan lesz, illetve maga a futás is megszakadhat.

Ha egy program kifut a memóriából, akár a Linux rendszermagot is összeomlaszthatja.

A beágyazott alkalmazások tervezését és megvalósítását tehát kellő körültekintéssel kell végezni. Egy alkalmazásnak képesnek kell lennie az összes lehetséges hiba kezelésére, ezeket a hibalehetőségeket pedig gondosan fel kell mérni és el kell hárítani – különösen akkor, ha memóriakezelésről van szó. Gyakran egy alkalmazás többször lefuttatható, mielőtt a soha fel nem szabadított memóriaterületek miatt rejtélyes módon összeomlana, rossz esetben a rendszert is magával rántja.

A hibákat a memóriaszivárgások észlelésére képes programok segítségével kereshetjük meg.

Ezek a programok a `malloc`, a `free` és az egyéb memóriakezelő függvények helyettesítésével működnek. Mindegyik rendelkezik olyan programrésszel, amely elfogja a `malloc` – és egyéb hasonló – függvények hívásait, és minden memóriakéréshez annak követésére alkalmas adatokat fűz. Egyesek memóriavédelemre is képesek, így elfogják a téves memóriahozzáférési kísérleteket.

Vannak olyan memóriaszivárgásokat érzékelő programok, amelyek rendkívül nagy méretűek, és a vizsgált program virtuális memóriaképével dolgoznak. Beágyazott rendszernél azonban az általuk támasztott követelményeket meglehetősen nehéz teljesíteni. Az `mtrace`, a `memwatch` és a `dmalloc` egyszerű programok, ám a legtöbb hibát ezekkel is megtalálhatjuk.

Mindhárom eszközt olyan C-példaprogrammal próbáltuk ki, amely általános memóriakezelési hibákat tartalmazott. A példaprogram a három említett hibakereső program támogatásával történő fordításához szükséges `Makefile` állományokkal együtt a 40. CD Magazin/Szivargas könyvtárában található. A hibakereső programokat különféle típusú célgépekkel is kipróbáltuk. A példaprogram natívan és keresztfordítva is használható.

mtrace

A három említett eszköz közül a legegyszerűbb az `mtrace`. Az `mtrace` a GNU C könyvtár egyik szolgáltatása, segítségével a `malloc/free` hívások egyensúlyának felbillenése észlelhető. Az `mtrace()` függvényhívás által használhatjuk, amely a hívások követését indítja el, és naplózza a `malloc` segítségével lefoglalt, valamint a felszabadított területek címét. Létezik egy szintén `mtrace` névre keresztelt Perl-parancsfájl is, amely a naplófájl kiegyensúlyozatlan hívásait jeleníti meg, és – amennyiben a forráskód hozzáférhető – a kód kérdéses `malloc`-hívást tartalmazó sorát is megjelöli. A program C- és C++-kód ellenőrzésére Linux alatt egyaránt használható. Az `mtrace` leginkább figyelemre méltó tulajdonságra a méretezhetősége. Általános hibakeresésre és modulárisan egyaránt használható.

Az `mtrace` használatának három alapvető eleme van: az ***mcheck.h*** állomány beillesztése a kódba, a `MALLOC_TRACE` környezeti változó beállítása és az `mtrace()` függvény meghívása. Ha a `MALLOC_TRACE` nincs beállítva, az `mtrace()` semmit sem csinál.

Az mtrace kimenete az alábbihoz hasonló:

```
- 0x0804a0f8 Free 13 was never alloc·d
/memory_leak/memory_leaks/mtrace/my_test.c:193
```

(a megjelölt című memóriarész nem volt lefoglalva, ezt követi majd a kipróbált program hibás sorának a száma).

Az üzenet azt jelzi, hogy a memóriát ugyan felszabadítottuk, csakhogy soha nem foglaltuk le, továbbá egy ***Memory not freed*** (Fel nem szabadított memória) rész jelzi azokat a címeket, a kérdéses memória méretét és a hibás kódsorokat, amelyek `malloc`-hívásához nem tartozik memóriafelszabadítás.

memwatch

A `memwatch` nemcsak a `malloc`, illetve `free` használatából fakadó hibákat, de a túlcsordulásokat is felismeri. Túlcsordulás alatt ez esetben azt értjük, amikor az adatok írása – a `malloc` segítségével – lefoglalt memóriarészbe kezdődik, ám a lefoglalt terület határain túl nyúlva történik.

A `memwatch` ugyanakkor nem ismeri fel a már felszabadított területre történő írást, illetve a lefoglalt területen kívül eső memóriarészek olvasását sem.

A `memwatch` központi eleme a ***memwatch.c*** állomány, ez tartalmazza a címellenőrzésekhez használt burkolókat és kódrészeket. A `memwatch` használatához a ***memwatch.h*** állományt a forráskódba be kell illeszteni. A `MEMWATCH` és az `MW_STDIO` változók értékeit a fordításkor parancssorban kell megadni (`-DMEMWATCH` és `-DMW_STDIO`). A fordításkor

természetesen a ***memwatch.c*** fájlra is szükség van.
A ***memwatch.c*** fájlból a fordítás során előálló objektummodult összefűzéskor az alkalmazásba kell építeni. A program futtatásakor bármely rendellenesség észlelése esetén az `STDOUT` kimeneten hibaüzenet jelenik meg. A felfedezett hibákkal kapcsolatos adatokat a ***memwatch.log*** állomány tartalmazza. Minden hibaüzenet megadja a kérdéses programkódot tartalmazó sor számát és a megfelelő állomány nevét.
A ***memwatch.log*** és az `mtrace` által készített naplófájl tartalmát összehasonlítva ugyanazokat a hibákat találtuk. A `memwatch` emellett egy túlcsordulást is észrevett, amelynél a lefoglalt terület elejét és végét jelző memóriacímek módosultak – ezáltal a `memwatch` kiterjedtebb tudása bizonyítást nyert.
A `memwatch` hátránya, hogy nem méretezhető, csak a teljes alkalmazással futtatva használható.

dmalloc

A harmadik eszköz egy könyvtár, amelyet a `malloc`, `realloc`, `calloc`, `free` és egyéb memóriakezelő függvények kiváltására készítettek. Beállításai futási időben is módosíthatók. Képes a memóriaszivárgások követésére, valamint az írásoknál felmerülő túlcsordulások észlelésére. A hibákat fájlnévvel és a kódsor számával jelenti, valamint általános kimutatásokat is készít. A könyvtárat ***Gray Watson*** készítette, átültetései számos, a Linuxtól eltérő operációs rendszerre is létrejöttek.
A csomagot megfelelő beállításokkal a többszálú és C++-programok támogatására is rá lehet venni. Megosztott és statikus könyvtárként is használható. A beállításokat a programfordítás közben lehet megadni. Az eredmény egy olyan könyvtárkészlet, amelyet az alkalmazás összefűzésekor használhatunk fel.
Az ellenőrizendő alkalmazás programkódjába a ***dmalloc.h*** állományt kell beilleszteni. Az ellenőrzés módját, illetve a naplózási beállításokat a könyvtár és a beillesztett állomány használata mellett egy környezeti változó segítségével kell megadni. Az alábbi sort a már említett `dmalloc` példaprogram segítségével végzett ellenőrzéséhez használtuk:

```
export \ DMALLOC_OPTIONS=debug=0x44a40503,
➥inter=1,log=napl fÆjl_neve
```

A beállítások jelentése a következő:
1. a ***log*** adja meg annak a naplófájlnak a nevét, amely a pillanatnyi könyvtárba kerül;
2. az ***inter*** a könyvtár önellenőrzésének gyakorisága;
3. a ***debug*** egy hexadecimális szám, amelynek bitjei a kívánt ellenőrzéseket választják ki.

A fenti példában gyakorlatilag az összes lehetséges hibát naplózzuk. A 40. CD Magazin/Szivargas/debug.txt állományában a választható ellenőrzéseket és a ***debug*** beállításban hozzájuk tartozó biteket soroljuk fel.

Ha a könyvtárat C++-program ellenőrzésére akarjuk használni, egy ***dmalloc.cc*** nevű forrásfájlra is szükség van. Ez a kódrész biztosítja a burkolófüggvényeket a `new` – `malloc` és `delete` – `free` hívásokhoz. A GDB nevű GNU hibakereső és a `dmalloc` együtt is használhatók. A csomag tartalmaz egy fájlt, amelyet a ***.gdbinit*** állományba illesztve a GDB tudomására hozhatjuk a `dmalloc` jelenlétét.
A könyvtár mellé egy `dmalloc` nevű segédprogramot is kapunk, amellyel a `DMALLOC_OPTIONS` változó értékeit módosíthatjuk. Készítettem egy parancsfájlt, amellyel a beállításokat az ellenőrzendő program futtatása előtt is megadhatjuk, így a kipróbálás változatlan körülmények között ismételhető meg.

Jelen cikkben csak az eszköz általános használatáról esik szó, ám leírásában további részleteket, illetve az egyéb szolgáltatások ismertetését is megtaláljuk. A példaprogramot az említett kiegészítések segítségével, a `DMALLOC` használatával futtattuk. Az eredmény akár meglehetősen kiterjedt is lehet – gondoljunk csak a túlcsordulásokra, amelyeknél a mutató túlszalad a lefoglalt területen, és ahol lehetőség van a kérdéses terület tartalmának naplózására. A naplófájl végén kimutatásokat, címeket, blokkméreteket és a megfelelő `free`-hívással nem párosítható `malloc`-hívások kódsorának számát találjuk.
A három említett eszköz különféle módokon támogatja a memóriaszivárgások felderítését és naplózását. Linux-munkaállomáson és keresztfordítással mindet egyaránt kipróbáltuk, és a futtatást különféle célgépeken végeztük. Egy alkalmazás fejlesztésénél a programozók mindhárom eszközt használták. Az `mtrace` segítségével találtak meg egy memóriaszivárgást okozó hibát egy külső fejlesztőtől átvett C++-könyvtárban, amelyben egy kivétel dobása, illetve elkapása okozott súlyosabb gondot. A `dmalloc` segítségével találták meg a memóriaszivárgásokat a linuxos `pthread`-átültetéssel futtatott alkalmazásokban. A `memwatch` roppant hasznosnak bizonyult egy átmeneti tárkezelő rendszer hibáinak felderítésében, amely hibásan végezte önmaga töredezettségmentesítését. Az eszközök kicsik, könnyen használhatók, a hibakeresés végeztével pedig játszi módon eltávolíthatók.
A példaprogram egy ***my_test.c*** nevű fájlból áll. Három külön könyvtárban található: ez a ***README*** és a `Makefile` állomány, illetve egy a próba futtatásához használható parancsfájl (40. CD Magazin/Szivargas). A `dmalloc`-próbához a környezeti változókat beállító parancsfájl mellékelve van. A kipróbálást a Red Hat- és SuSE-Linux terjesztéseken, illetve a MontaVista Linux keresztfordítási környezetében végeztük.

Linux Journal 2002. szeptember, 101. szám

Cal Erickson
jelenleg a MontaVista Software vezető Linux-tanácsadója. Mielőtt csatlakozott volna a MontaVista csapatához, támogatásvezető mérnök volt a Mentor Graphics beágyazott alkalmazások részlegénél. Cal több mint harminc éve dolgozik a számítástechnikai iparágban, tapasztalatát számítógépgyártóknál és felhasználói termékeket fejlesztő cégeknél szerezte. A cal_erickson@mvista.com címen érhető el.

Kapcsolódó címek

A `dmalloc` a ➲ http://dmalloc.com oldalon érhető el. Letölthetők a forráscsomagok, a leírás, a beállító parancsfájlok és egy példaprogram.

Az `mtrace` tekintetében további tájékoztatást az `info libc` parancs kiadásával, majd a *Memory Allocation* (Memóriafoglalás) fejezet *Allocation Debugging* (Memóriafoglalási hibák keresése) című részére ugorva találhatsz.
A `memwatch` a ➲ http://www.linkdata.se/sourcecode.html címen érhető el. Az oldalt John Lindh, a Link Data tulajdonosa tartja fenn.