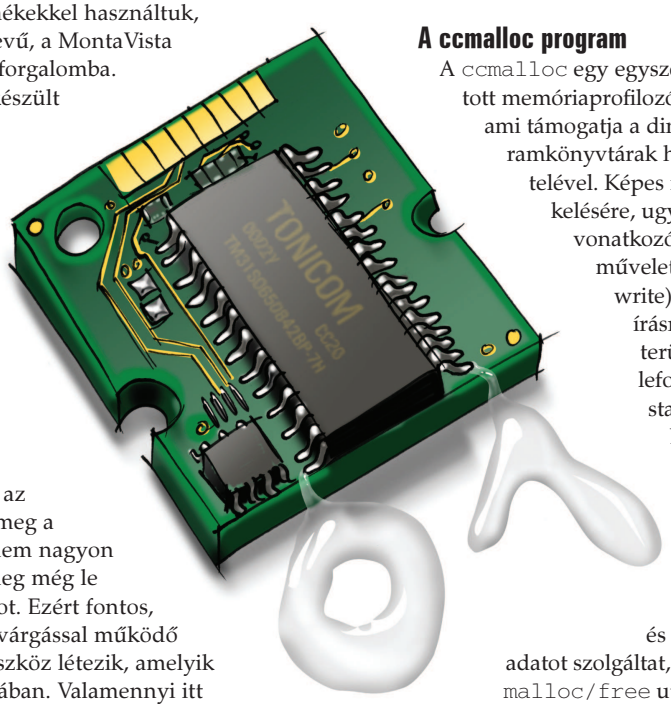
Memóriaszivárgás érzékelése C++-környezetben

Ne halogassuk a memóriaszivárgási gondok megoldását! Egy, esetleg több kényelmes eszközt is programfejlesztési folyamatunk részévé tehetünk.

Egyik korábbi cikkünk (Memóriaszivárgás érzékelése beágyazott Linux-rendszerekben, Linuxvilág, 2002. szeptember) a memóriaszivárgás érzékelésének kérdéseit tárgyalta C programozási nyelvet használó környezetben. Jelenlegi írásunk viszont a C++-programokban jelentkező memóriaszivárgás érzékelésével kapcsolatos kérdésekre keres választ, nem pedig a rendszermagbeli szivárgásokra. Valamennyi itt bemutatott eszközt a MontaVista Linux Professional Edition 2.1 és 3.0-s termékekkel használtuk, egyikük pedig, a `dmalloc` nevű, a MontaVista Linuxszal kerül kereskedelmi forgalomba.

A beágyazott rendszerekhez készült alkalmazói programok tervezőinek és programozóknak különös gondot kell fordítaniuk e programok erőforrás-felhasználására. A munkaállomásoktól eltérően a beágyazott rendszerek csupán véges tárterülettel rendelkeznek. Az ilyen rendszerekben nem található lapozási terület a működésüket szüneteltető programok számára. Amint a rendszer felhasználja az összes erőforrását, a pánikon meg a rendszer újraindításán kívül nem nagyon marad más választásunk, esetleg még le lehet állítani egy-két programot. Ezért fontos, hogy elkerüljük a memóriaszivárgással működő programokat. Számos olyan eszköz létezik, amelyik segít a szivárgások megtalálásában. Valamennyi itt bemutatott eszközhöz létezik saját tesztprogram.



Az egyik módszer sikeres alkalmazásfejlesztői használatát magam is láttam, ez a prototípuskód fejlesztéséhez és kipróbálásához készült, és amennyire csak lehetséges, egy munkaállomás használatát is magában foglalja. A munkaállomáson való kipróbálással az alkalmazói programok fejlesztői megkönnyíthetik a célprocesszorra való áttérést. A munkaállomások használata melletti döntő érv, hogy olcsók és minden érintett résztvevő birtokában van ilyen. Ezzel szemben a célgépek száma kevés, az iránt uk való kereslet pedig nagy.

A legtöbb memóriaszivárgás-érzékelő program teljes forráskód formájában érhető el. Ezeket jellemzően x86-alapú felületeken készítették. Nem x86-os felületeken történő használatuk új felületre való átültetést igényel (porting). Ez az átültetési művelet az egyszerű újrafordítástól, szerkesztéstől és futtatástól az egyik felületről a másiknak megfelelő gépi kód átalakításáig terjedhet. Némelyik eszközzel használati tanácsokat illetve javaslatokat kapunk a más felületekre történő programfordításhoz.

A dmalloc használata

Amint arról a 2002. szeptemberi számban már részletesen beszámoltam, a `dmalloc` szerzőjének állítása szerint a program C++-ismerete korlátozott, így a C++ alapú memóriaszivárgások felismerése úgyszintén az.

Annak érdekében, hogy a `dmalloc`-ot a C++-szal és programszálakkal együtt lehessen használni, az alkalmazást `static`-ként kellett szerkeszteni.

A ccmalloc program

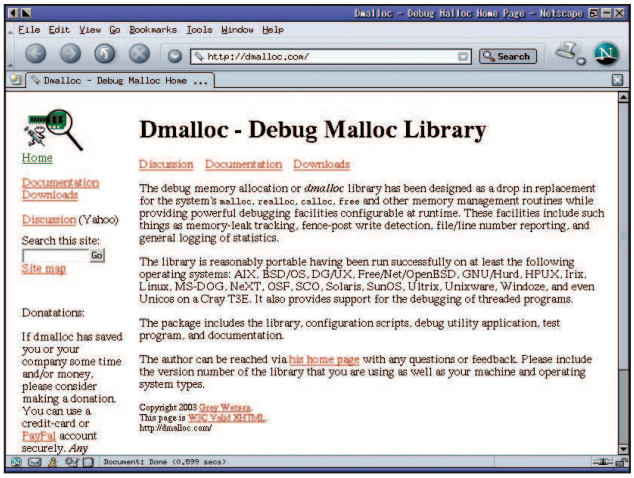
A `ccmalloc` egy egyszerű használati modellel ellátott memóriaprofilozó (memory-profiler) eszköz, ami támogatja a dinamikusan szerkesztett programkönyvtárak használatát, a `dlopen` kivételével. Képes memóriaszivárgások érzékelésére, ugyanazon adatállományokra vonatkozó többszörös felszabadítási műveletekre, hitelesítésre (underwrite), felülírásra (overwrite) és írásra (write) a már felszabadított területen. Megjelenít állomány-lefoglalási és -felszabadítási statisztikákat.

Egyaránt alkalmazható tökéletesített (optimized) és lecsupaszított (stripped) programkódra, és támogatja a C++ programnyelvet. A teljes hívási láncra vonatkozó állomány és programsor-számozási adatot szolgáltat, és nem egyes-egyedül a `malloc/free` utasítás közvetlen hívójára vonatkozóan, a C++ nyelv itt is támogatott.

A `ccmalloc` használatához nincs szükség a program újrafordítására: mindössze a szerkesztésnél (link) a `-lccmalloc -ldl` vagy a `ccmalloc.o -ldl` parancsokat kell használni. A `ccmalloc` képes a hívási láncolatokat hatékonyan ábrázolni, a hívási láncok testre szabható nyomtatását, hívási láncok kiválasztás szerinti nyomtatását, tömörített naplóállományt és a ***.ccmalloc*** névvel ellátott indítóállományt. A leírás fő része a ***ccmalloc.cfg*** nevű állományban található, a fejlesztőeszközhöz tartozó próbaállományok további leírást tartalmaznak. A rendszerváltozók eléréséhez az `nm` és `gdb` szükséges, míg a gzip a naplóállományok tömörítéséhez.

NJAMD

Az NJAMD, miként a program alkotója állítja, „nem csupán egy újabb malloc-hibakereső”. Mint az a legtöbb tárterület-foglalást ellenőrző hibakereső programnál megfigyelhető, a szabványos területfoglalási ellenőrző műveleteket újakkal helyettesítik, amik különféle ellenőrzéseket végeznek a tár használatbavétele során. Nevezetesen a program ellenőrzi a dinamikus



átmeneti tár alul-, illetve túlcsordulásait és érzékeli a tárterület felszabadítás utáni újbóli használatát. Az NJAMD-hez készített programkönyvtár LD_PRELOAD-olható vagy a programhoz szerkeszthető. Az első tárterület-foglaláskor nagy – 20 MB-os – tárhoz kapcsolódó átmeneti tárat hoz létre, és ebből szab ki annyi memóriát, amennyit a program igényel.
Az NJAMD használható önmagában, ügyféloldali programmal együtt, vagy a `gdb`-ből. Egy segédprogramja is van, ami a rendszerleállás utáni veremellenőrzést teszi lehetővé (postmortem heap analysis). Egy további szolgáltatás lehetővé teszi az alkalmazás hibakeresését, átugorva az újrafordítást: egyszerűen csak előre be kell tölteni a programkönyvtárat (preload).
Az NJAMD képes nyomon követni a szivárgásokat a `malloc` és `free` függvényeket burkoló könyvtári függvényekben, grafikus felhasználói felület (GUI) alapú helyfoglaló elemeknél, valamint a C++ `new` és `delete` függvényeknél.
Gyakran előfordul, hogy a memóriaszivárgás nem fedezhető fel azonnal, hanem lappang, hogy aztán később, a legváratlanabb pillanatban sújtson le. Ennek kiderítése sok időt vehet igénybe. Az NJAMD számos rendszerváltozóval bír, amelyek különböző érzékelési szintek beállítását teszik lehetővé.
Mint a legtöbb hibakereső eszköznél már megszokhattuk, a teljesítmény fontos kérdéssé válhat az NJAMD használatakor, ezért az eszközt leginkább csak a fejlesztési szakaszban szabad használni. A bekapcsolt állapotú eszköz mellett végzett telepítés a rendszer lelassulását eredményezheti.

YAMD

A YAMD, vagyis a Yet Another Memory Debugger szavakból képzett névvel ellátott program egy olyan újabb csomag, ami a lefoglalt tárterületek széleit vizsgálja. Ezt a processzor lapozási módszerének használatával végzi. A program képes érzékelni a határokon kívüli olvasási és írási műveleteket. A hibát a program már az azt előidéző utasításnál észleli, nem pedig később, amikor további hozzáférések jelentkeznek. A híváselfogásokat a program naplóban rögzíti, az állomány nevével, a programsor számával és nyomkövetési adatokkal együtt. A nyomkövetés azért fontos, mert a tárterület lefoglalása véges számú eljáráson keresztül történik meg.
A könyvtár a `malloc` és `free` hívásokat utánozza. E művelet végrehajtása sok közvetett `malloc` hívás elfogását jelenti, amilyeneket például a `strdup` kezdeményezett. Ezenkívül a `new` és `delete` műveleteket is elfogja, kivéve, ha azok túl vannak terhelve.
A YAMD, más rokon programokhoz hasonlóan, nagy mennyiségű látszólagos memóriát vagyis lapozási területet igényel a működéséhez, viszont beágyazott rendszeren ez működik. Ezt az eszközt munkaállomáson prototípus-hibakeresésre is használhatjuk. Ha ezzel a hibakereséssel végeztünk, az alkalmazás célprocesszorra történő ültetése során biztosak lehetünk abban, hogy a legtöbb vagy éppen az összes memóriaszivárgást megtaláltuk.
A YAMD a futtatáshoz egy héjprogramot, a `run-yamd`-t kínálja, ami a program végrehajtásának megkönnyítésére szolgál. A program bizonyos kiinduló állapotokból való helyreállításhoz több lehetőséget nyújt. Naplóállományt lehet készíteni, amikor az ellenőrzött program magtárterület-kiíratást (core dump) hajt végre. A hibakereső eszközt a YAMD által vezérelt programokban a hibák felderítésére lehet használni. Mindazonáltal hibák a hibakereső eszköz használata során akkor is jelentkezhetnek, ha a YAMD-t előre betöltjük, ahelyett, hogy statikusan össze lenne építve a programmal.

Valgrind

A Valgrind viszonylag új, nyílt forrású tárterület-hibakereső az x86-os felületen használható GNU/Linux-rendszerekhez. Több képességgel van megáldva, mint a korábbi eszközök, viszont kizárólag x86-os gépeken fut. Ha egy program végrehajtása a Valgrind alatt zajlik, akkor az összes tárból való olvasási vagy odaírási művelet az összes `malloc`, `free`, `new` és `delete` eljárás ellenőrzésével zajlik. A Valgrind képes érzékelni a használatlan tárterületet, a memóriaszivárgásokat, az előkészítetlen memóriára vagy rosszul címezhető tárterületre való hivatkozást, a Posix-szálak hibás használatának néhány fajtáját, és a `malloc-free` és a `new-delete` műveletpárok típushibáit.
A Valgrind a hibaszűrésre a `gdb`-vel együtt is használható, ami lehetővé teszi a programozó számára, hogy a hiba jelentkezésének helyén a `gdb`-t használja. Ilyen módon a programozó közvetlenül a hibaforrást nézheti meg, és hamarabb találhat rá megoldást is. Némelyik esetben javítófolt is készíthető, a hibakeresés pedig folytatódhat. A Valgrindet úgy tervezték, hogy kicsi és nagy alkalmazásokon egyaránt képes legyen dolgozni, például a KDE 3-on, a Mozillán és az OpenOffice-on.
A Valgrind egyik szolgáltatása az, hogy képes részletes adatokat szolgáltatni az átmeneti tár használatának jellegéről (profiling). A program képes a processzor (CPU) L1-D, L1-I és egyesített L2 átmeneti tárának részletekbe menő utánzására, és a felhasználási eredményeket a vizsgált program minden egyes sorára vonatkozóan ki tudja számítani. A Valgrindnak jó a HOGYAN-ja, amelyben rengeteg példa található. A Valgrind honlapján temérdek leírás szerepel, bejárása mégis egyszerű. Számos különféle lehetőséget magában foglaló összeállítás érhető el, és a felhasználókra van bízva, hogy a számukra legkedvezőbbet kiválasszák.
A Valgrind hibamegjelenítő képernyőjén szerepel a vizsgált program azonosítója (PID), amit a hiba leírása követ. A címekkel együtt megjelennek a forrásállományok nevei és a programsorok számai. A teljes visszakeresés végig megjelenítődik.
A Valgrind működésének megkezdésekor beolvas egy indítóállományt, ami olyan utasításokat tartalmazhat, amelyek megmondják, hogy milyen hibaellenőrző üzenetek ne jelenjenek meg. E lehetőség használatával mód nyílik rá, hogy figyelmünket egy meghatározott programra összpontosítsuk, és ne a programkönyvtárakra, amiket különben sem lehet módosítani.
A Valgrind úgy végzi az ellenőrzést, hogy az alkalmazást szimulált processzorkörnyezetben futtatja. Ez a dinamikus szerkesztőt, illetve betöltőt (linker/loader) arra kényszeríti, hogy elsőként a szimulációs programot töltse be, s csak azután tölti

be a hozzátartozó könyvtárakat a szimulátorprogramba. A program futásának idején mindenféle adat gyűjtése zajlik. Amint a program végrehajtását megszakítjuk, a naplózott adatok megjeleníthetők, vagy akár kiírhatók a naplóállományba.

mpatrol

Az mpatrol programkönyvtár a tárterület lefoglalásának ellenőrzésére a vizsgálni kívánt programmal összeszerkeszthető. A program olyan módon lett megírva, hogy több különböző operációs rendszer felületén is futtatható. Határozottan nagy előnye, hogy több különböző processzorra át lett ültetve (porting), vagyis MIPS-re, PowerPC-re, x86-ra és néhány MontaVista fogyasztó által a StrongARM célgépekre.
Az mpatrol nagymértékben testreszabható: beállítható, hogy a verem használata helyett rögzített méretű statikus tömbből foglaljon le területeket. Statikus, osztott és szálbiztos könyvtárként lehet összeépíteni. Ugyanakkor a program megjelenhet egyetlen nagyméretű tárgykódú állomány formájában is, amit az alkalmazással össze lehet építeni, nem pedig könyvtárban kell elhelyezni. Ez a működési sajátosság a felhasználónak nagyfokú rugalmasságot biztosít.
A program által létrehozott kód negyvennégy különböző tárterület-foglalási és lánckezelési művelet pótlását tartalmazza. A kampók (hooks) adottak, így ezek az eljárások közvetlenül a `gdb`-ből hívhatók. Ez az adottság teszi lehetővé az mpatrolt használó programok számára a hibák keresését.
A könyvtárbeállítások és a veremhasználat a program működése során időszakosan megjeleníthető. A program futása idején gyűjtött összes statisztikai adat a program leállításakor megjelenik. A program előre beállított értékekkel rendelkezik, amelyek a környezeti változókkal felülbírálhatók. E változók futásidőben történő megváltoztatása szükségtelenné teszi a könyvtár újraépítését. A különféle tesztek hangolása dinamikusan végezhető el. Valamennyi naplózási tevékenység eredménye a pillanatnyi könyvtárban elhelyezkedő állományokba gyűjthető, de átirányítható a `stdout`-ra, a `stderr`-re vagy egyéb állományokba.
A program futása közben gyűjteni lehet a veremhívási visszakeresési adatot, és ezt állományba is lehet rögzíteni. Amennyiben a programot és járulékos könyvtárait a változók és programsorszámozás bekapcsolásával építették össze, ezeket az adatokat a naplóállományban ugyancsak meg lehet jeleníteni. Ha a programozó egy bizonyos ponton úgy dönt, hogy a terhelési tesztet egy kisebb tártartományon (smaller memory footprint) szeretné kipróbálni, az mpatrolt a tárterület korlátozására lehet utasítani. Ez a lehetőség olyan próbafeltételek megteremtését jelenti, amiket géptermi körülmények között készen talán sehol sem lehet megtalálni. A felhasználói környezetben végzett terhelési próba vagy egy gép próbaüzemeltetése ezzel a szolgáltatással könnyebbé válik. A fentebb említetteken kívül a hibajavítási eljárások ellenőrzése érdekében a próbaprogrammal tárterületfoglalási hibákat is elő lehet idézni. A programnak eme képessége hasznosnak bizonyulhat a C++ nyelvű kivételkezelésben. A veremről pillanatfelvételek készíthetők, hogy képet kaphassunk a tárhasználat alsó és felső értékeiről.

Insure++

A Parasoft cég által készített Insure++ termék nem tartozik a GPL-esített programok közé és nem is ingyenes, viszont jól használható eszköz a memóriaszivárgás és a kódlefedettség érzékelésére – nagyon hasonlít az mpatrolra.
Az Insure++ azonban az mpatrolnál ténylegesen több munkát végez a kódlefedettség területén, valamint adatgyűjtő és megjelenítő eszközökkel is szolgál. A program próbapéldányai letölthetők, és meghatározott ideig nem linuxos munkaállomásokon használhatók.
A program könnyen telepíthető Linuxra, de csomóponthoz kötött azon a gépen, amelyre telepítették. Az Insure++ programmal együtt átfogó leírást és számos bővítési lehetőséget kaphatunk. A kódlefedettséget vizsgáló eszköz különálló program, de már az alapcsomagban is szerepel.
Az Insure++ sok adattal szolgál a feltárt hibákról. Az Insure++ programot a használatbavételhez a felhasználói felülettel együtt kell újrafordítani. Ez a felhasználói program képezi az Insure++ könyvtári eljárásainak használatához szükséges programkódot. A fordítási szakaszban az érvénytelen típusátalakítások és a hibás értékátadások egyaránt felszínre kerülnek. A nyilvánvaló tárhasználati rendellenességeket ugyancsak jelenti a program. Futási időben a hibák a `stderr`-re kerülnek, de grafikus eszközzel is megjeleníthetők. Alkalmazás készítésekor a parancssor vagy a `makefiles` használható, ami nagy méretű alkalmazások létrehozását és komoly feladatok megoldását teszi lehetővé.
A program végrehajtása egyszerű: az Insure++ semmilyen különleges végrehajtandó parancsot nem igényel, úgy működik, mintha egy közönséges program lenne. Valamennyi hibakereső- és elfogó programkód az Insure++ programkönyvtáraiban található, amelyek össze lettek szerkesztve a programmal.
Az Inuse nevű bővítmény valós időben mutatja meg, hogyan használja a program a tárat. Képes pontos képet rajzolni a memóriafelhasználásról, hogy az mennyire válik töredezetté, és az apró, ám idővel egyre jelentősebbé váló kódszivárgásról. Egy ügyfélnél tapasztaltam, hogy egy bizonyos C++-osztálynál kis területű memória szivárgott, ami egy munkaállomásról nézve egészen aprónak látszott. Egy várhatóan hónapokig, sőt évekig futó beágyazott rendszeren a szivárgás meglehetősen felduzzadhat. Ezzel az eszközzel a szivárgásnak gyorsan nyomára bukkanhatunk, megtalálhatjuk és kijavíthatjuk. Más hozzáférhető eszközök még nem voltak képesek ezt a szivárgást észrevenni.
A kódlefedettséget egy további eszköz, a TCA vizsgálja. Amikor a program az Insure++ működése közben fut, adatokat gyűjt, és amint a TCA elemzi azokat, pontosan megmutatja, hogy milyen programkód lett végrehajtva. A TCA grafikus felhasználói felülete finomítja a kódlefedettségről alkotott képet.

Linux Journal 2003. június, 110. szám

Cal Erickson (cal_erickson@mvista.com)
Vezető Linux-tanácsadóként dolgozik a MontaVista Software cégnél. Ezt megelőzően a Mentor Graphics Embedded Software Division vezető tanácsadó mérnöke volt. Cal több mint harminc éve dolgozik a számítógép-ipar területén.

KAPCSOLÓDÓ CÍMEK

dmalloc ➲ http://dmalloc.com
ccmalloc ➲ http://www.inf.ethz.ch/personal/biere/projects/ccmalloc
Insure++
➲ http://www.parasoft.com/products/insure/index.htm
mpatrol ➲ http://www.cbmamiga.demon.co.uk/mpatrol
NJAMD ➲ http://sourceforge.net/projects/njamd
Valgrind ➲ http://developer.kde.org/~sewardj
YAMD ➲ http://www3.hmc.edu/~neldredge/yamd