µClinux-ismeretek Linux-programozók számára

Mit szólnál, ha a programjaid memóriakezelés nélküli processzorokon is futnának? Azt, hogy sok munka kellene hozzá? Szó sincs róla.

Az µClinux népszerűsége az utóbbi időben hatalmasat nőtt, és egyre több fogyasztói készülékben láthatjuk viszont. Forgalomirányítókban (*1. kép*), webkamerákban, DVD-lejátszókban való használata önmagában is igazolja sokoldalúságát. Az olcsó, a µClinux futtatására alkalmas 32 bites processzorok terjedése további választási lehetőségeket kínál a µClinux használatán gondolkodó gyártóknak. A µClinux immár a 2.6-os rendszermag része, így borítékolható, hogy ismertsége tovább fog nőni.

1. kép A SnapGear LITE2 VPN forgalomirányítón uClinux fut

Egyre több fejlesztő játszadozik a µClinux használatának gondolatával, ezért egy olyan útmutató, amely összefoglalja a normál Linuxtól való eltéréseit, valamint az alkalmazásánál felmerülő buktatókat és csapdákat, rendkívül nagy értékkel bír. A következőkben áttekintjük, hogy az µClinux használatára magát elszánó fejlesztőnek milyen módosítások elvégzésére kell felkészülnie, valamint maga a környezet hogyan hat a fejlesztési folyamatra.

Memóriakezelés nélkül

Meghatározó és szembeszökő különbség a µClinux és az egyéb Linux-rendszerek között, hogy előbbi nem képes memóriakezelésre. Linux alatt a memóriakezelés virtuális memória (***virtual memory***, VM) használatával folyik, a µClinux viszont olyan rendszerekre készült, amelyek VM-támogatással nem rendelkeznek. Mivel a VM-kezelés általában az úgynevezett memóriakezelő egység (***memory management unit***, MMU) segítségével történik, a µClinux kapcsán sokszor hallani az MMU hiányára utaló NOMMU (nincs MMU) kifejezést.

VM használatakor minden folyamat ugyanarról a – valójában virtuális – címről fut, és a VM alrendszer feladata annak figyelése, hogy ezekhez a virtuális címekhez melyik fizikai memóriaterület tartozik. Lehetséges tehát, hogy az egy folyamat által látott virtuális memóriaterület összefüggő, ám az a fizikai rész, amelyre leképeződik, széttöredezett; sőt, még akár a merevlemezen található csereterületen is lehet. Mivel az önkényesen lefoglalt memória a folyamat címterén belül bárhová leképezhető, már futó folyamathoz minden gond nélkül lehet további memóriát hozzáadni.

VM nélkül minden folyamatot arra a memóriaterületre kell helyezni, ahonnan futni tud. A legegyszerűbb esetben ennek a memóriaterületnek összefüggőnek kell lennie. Bővítésére általában nincs lehetőség, mert előtte és utána más folyamatok lehetnek. Egy µClinux alatt futó folyamat tehát nem tudja a hagyományos Linux alatt futó folyamatokhoz hasonlóan növelni a rendelkezésére álló memória méretét. Ugyan a futtatás érdekében minden programot át kell helyezni, ez a művelet a fejlesztő számára észrevétlen. Szükségessége a VM hiányából fakad, és persze szálka minden µClinux fejlesztő szemében. Közvetlen következmény, hogy semmilyen memóriavédelemmel nem számolhatunk – a rendszermag és a folyamatok a rendszer tetszőleges területét összezavarhatják. Egyes processzortípusok lehetővé teszik ugyan bizonyos be- és kiviteli, utasítás- és memóriaterületek felhasználóktól való védelmét, de garanciát semmire sem kapunk. A rendszer összeomlását okozó hibáknál már csak azok rosszabbak, amelyeket nem veszünk észre, márpedig a folyamatok közötti véletlenszerű memória-felülírások felderítése rendkívül nehéz feladat.

VM hiányában gyakorlatilag csereterületet sem lehet használni. Igaz, a µClinuxot futtató rendszerek esetében ez nem jelent túl nagy gondot, hiszen merevlemezzel vagy a csereterület kihasználásához elegendően nagyméretű memóriával amúgy sem rendelkeznek.

A rendszermag eltérései

A rendszermagot fejlesztők számára a µClinux kevés meglepetést tartogat. Az egyetlen komoly különbség az, hogy az MMU által biztosított lapkezelési lehetőségekről le kell mondani. A gyakorlatban ez csak minimális mértékben érinti a rendszermag működését. A `tmpfs` például nem használható µClinux alatt, mert működése a VM alrendszerre alapul.

Hasonlóképpen le kell mondanunk a szabványos futtatható formátumokról, ezek ugyanis µClinux alatt el nem érhető VM szolgáltatásokat vesznek igénybe. Ehelyett egy új, egyszintű formátumot kell követni. Az egyszintű formátum olyan sűrített futtatható formátum, amely csak futtatható

kódot és adatokat tárol, valamint tartalmazza azokat az áthelyezéseket, amelyek révén a futtatható állomány a memória tetszőleges területére betölthető.
Az illesztőprogramokat sokszor kell módosítani a µClinuxra való áttéréskor, ám nem a rendszermag esetleges eltérése, hanem a támogatandó eszközök jellege miatt. Az SMC hálózati illesztőprogram például támogatja az ISA buszos SMC kártyákat. Ezek általában 16 bites kártyák, és valamilyen 0x3ff fölötti be- és kiviteli címen érhetők el. Az illesztőprogram könnyedén rávehető a lapka nem ISA buszos, beágyazott változatainak támogatására, de 8, 16 vagy 32 bites módban kell futnia, 32 bites be- és kiviteli címet kell használnia, és a megszakítás sorszáma is jóval nagyobb lesz. (Az ISA buszos eszközök megszakítása legfeljebb 16-os lehet.) Tehát az illesztőprogram nagyrészt változatlan marad, a gép jellegzetességei miatt kisebb átültetési munkákat el kell végezni. Elég gyakori, hogy a régebbi illesztőprogramok a be- és kiviteli címeket rövid egészként tárolják, beágyazott µClinux alapú rendszeren viszont, ahol az eszközök leképezett be- és kiviteli címeken érhetők el (memory-mapped I/O addresses), erre nincs lehetőség.
Az `mmap` rendszermagon belüli megvalósítása is merőben eltérő. Ugyan működése a fejlesztő számára nagyjából észrevétlen, mégis ismerni kell a lelkivilágát, különben előfordulhat, hogy a µClinux alapú rendszereken nagyon rossz hatékonyságú megoldásokat fogunk kidolgozni. Hacsak a µClinux `mmap`-ja nem képes közvetlenül a fájlrendszeren belül mutatni a fájlra, biztosítva ezzel annak soros elérését és folytonosságát, memóriát kell foglalnia, és át kell másolnia az adatokat a memóriába. Az `mmap` µClinux alatti hatékony használatához bizony különleges feltételeknek kell teljesülniük. Először is, jelenleg az egyetlen fájlrendszer, amely képes a fájlok összefüggő tárolását biztosítani, az a `romfs`. Mivel a memóriafoglalást el akarjuk kerülni, `romfs`-t kell használnunk. Másodszor, kizárólag a csak olvasható leképezéseket lehet megosztani, vagyis – ugyancsak a memóriafoglalás elkerülése miatt – a leképezések csak olvashatóklehetnek. Éppen ezért µClinux alatt a fejlesztők nem használhatják ki a másolás írás közben (***copy-on-write***) lehetőségeket. A rendszermagnak is figyelembe kell vennie, hogy a fájlrendszernek ROM-ban kell lennie, amely egy csak olvashatónak kinevezett terület lesz a processzor címterében. Erre akkor van lehetőség, ha a fájlrendszer valahol RAM-ban vagy ROM-ban található. Mindkét esetben közvetlen címezhetőséget kell biztosítani a processzor számára. Nulla méretű `mmap` memóriafoglalásra nincs lehetőség, ha a fájlrendszer merevlemezen található, hiába használunk akár `romfs`-t, ugyanis ilyenkor a processzor nem tudja közvetlenül megcímezni a tartalmat.

Memóriafoglalás a rendszermag és az alkalmazások számára

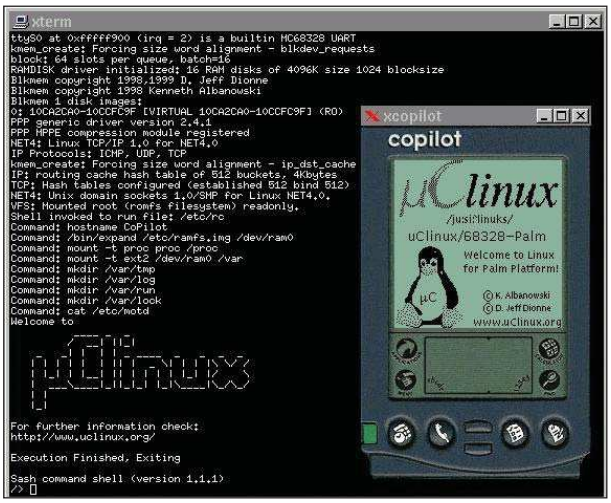
A µClinux kétféle magszintű memóriafoglalót (allocator) kínál. Első látásra talán nem egyértelmű, miért van szükség egy másik memóriafoglalóra is, de a kisméretű µClinux rendszereknél a különbség nyilvánvalóvá válik. A Linux alapértelmezett memóriafoglalója a kettő hatványaira épülő foglalási módszert alkalmaz. Működése ennek köszönhetően gyorsabb, és hamar talál a foglalási kérésnek megfelelő méretű memóriaterületet. Sajnos µClinux alatt az alkalmazásokat arra a memóriaterületre kell betölteni, amit a memóriafoglaló biztosít. A – különösen nagyméretű területek foglalásakor jelentkező – következményeket hamar megértjük, ha veszünk példaként egy 33 KB memóriát igénylő alkalmazást. A memóriafoglalásnál a kettő következő hatványa szerint ez 64 KB memóriát fog kapni. A feleslegesen lefoglalt 31 KB területet nem tudjuk hatékonyan felhasználni. Az ilyen jellegű memóriapazarlás a legtöbb µClinux rendszerben elfogadhatatlan. Az ebből fakadó gondok elkerülésére egy másik memóriafoglalót is készítettek a µClinux rendszermagokhoz. Általában `page_alloc2` vagy `kmalloc2` névvel illetik, a tényleges rendszermagváltozattól függően. A `page_alloc2` a kettő hatványai szerint történő memóriafoglalásból fakadó pazarlást szünteti meg. Az egy lapnyinál (egy lap mérete 4096 bájt, vagyis 4 KB) kevesebb memóriát igénylő alkalmazások számára ez is a kettő hatványai szerint foglalja a memóriát, e felett azonban a lefoglalt terület méretét a következő egész lapméretre kerekíti. Visszatérve az előző példához, egy 33 KB-ot igénylő alkalmazás 36 KB memóriát fog kapni, vagyis egy 33 KB-os alkalmazásnál azonnal megtakarítottunk 28 KB-ot.
A `page_alloc2` a memória töredezése ellen is megteszi a szükséges lépéseket. A két lapnyi (8 KB) vagy kisebb igényeket a memória elejéről, az ennél nagyobbakat a végéről elégíti ki. Ezzel elkerülhető, hogy az átmeneti, például hálózati pufferek számára végzett foglalások miatt a memória feltöredezzen, és később a nagyobb alkalmazások futtatása meghiúsuljon. Részletesebb memóriatöredezési példát lejjebb, az Alkalmazások és folyamatok című részben mutatok. A `page_alloc2` persze nem tökéletes, de a gyakorlatban remekül működik, ugyanis a µClinuxot használó beágyazott készülékeken alkalmazások egy viszonylag állandó csoportja és általában hosszú ideig fut. Miután a kedves fejlesztő túltette magát a rendszermag memóriafoglalási nyűgein, a valódi érdekességeket az alkalmazások terén tapasztalhatja. Itt mutatkozik meg igazán a µClinux VM-kezelésének hiánya. A legnagyobb különbség, ami az alkalmazások µClinux alatti működését gátolja, az a dinamikus verem hiánya. A VM-kezeléssel rendelkező Linuxok alatt, ha egy alkalmazás megpróbál a verem tetején túlra írni, kivételjelzést kapunk, és a vermet újabb memóriafoglalással bővíti a rendszer. µClinux alatt ilyesfajta kényelemre ne számítsunk, a vermet fordítási időben kell lefoglalni. Az a fejlesztő, aki alkalmazásának kapcsán eddig talán nem is törődött a verem használatának témakörével, most kénytelen lesz tisztázni az ilyen vonatkozású kérdéseket. Az első dolog, amivel a fejlesztőknek foglalkozniuk kell, amikor újonnan átültetett alkalmazásuk rejtélyesen összeomlik vagy rendellenesen kezd működni, az a lefoglalt verem mérete. Alapesetben a µClinux 4 KB-ot foglal a verem számára, ami egy korszerű alkalmazás esetében nagyjából a semmivel egyenlő. A fejlesztőnek az alábbi módszerek valamelyikével meg kell próbálnia növelni a verem méretét:

1. Az `FLTFLAGS=-s <veremméret>` hozzáadása és az `FLTFLAGS` exportálása az alkalmazáshoz tartozó `Makefile` állományba fordítás előtt.
2. Az `flthdr -s <veremméret>` parancs futtatása az alkalmazás lefordítása után.

A második jelentős eltérés, ami a µClinux alatti fejlesztést megnehezíti, az a dinamikus kupac (***heap***) hiánya – ez az

a terület, ahonnan a `malloc` és a hasonló C-függvények használatával bejelentett igények kielégítése történik.
A VM-kezeléssel rendelkező Linux-rendszerek alatt az alkalmazások növelhetik folyamatméretüket, vagyis dinamikus kupaccal rendelkezhetnek. Mindennek megvalósítása hagyományosan alacsony szintű `sbrk/brk` rendszerhívásokkal történik, melyekkel növelhető és megváltoztatható a folyamatok címterének mérete. A könyvtári függvényekkel – mint a `malloc` – történő kupackezelés annak a memóriaterületnek a felhasználásával történik, amelynek lefoglalására az alkalmazás nevében meghívott `sbrk()` függvény segítségével került sor. Ha egy alkalmazásnak bármikor több memóriára van szüksége, akkor csak újra meg kell hívnia az `sbrk()` függvényt. A csökkentésre a `brk()` segítségével nyílik mód. Az `sbrk()` a folyamat végén növeli meg a memóriaterületet. A `brk()` önkényesen dönt, hogy közelíti az elejéhez a folyamat végét, vagyis csökkenti annak méretét, vagy kijjebb tolja azt, vagyis növeli a folyamatot.
Mivel μClinux alatt a `brk` és az `sbrk` szolgáltatásai nem valósíthatók meg, egy átfogó memóriakészletből kell gazdálkodni, amely alapjában véve a rendszermag szabad memóriakészlete. Természetesen ennek a megoldásnak is vannak hátulütői. Például egy megszaladó folyamat akár a teljes rendszermemóriát felemésztheti. A rendszerkészletből való foglalás nem egyenértékű az `sbrk` és a `brk` használatával, ezek révén ugyanis a folyamat címterének végét toljuk ki. Egy normál `malloc` megvalósítás tehát nem felel meg, új megvalósításra van szükség.
Az átfogó készletes szemléletnek nyilván előnyei is vannak. Az első, hogy csak a ténylegesen szükséges memóriaterületre tesszük rá a kezünket, nem úgy, mint az előre foglalt kupacot használó beágyazott rendszereknél. Ez a μClinux rendszerek esetében rendkívül fontos tényező, hiszen memória tekintetében általában szűkösen állnak. A második előny, hogy amikor befejezzük egy memóriaterület használatát, azonnal visszatehetjük a készletbe. A megvalósítás során támaszkodhatunk a rendszermag beépített memóriakezelő szolgáltatásaira is, így csökkenteni tudjuk alkalmazásunk kódjának méretét. Az új felhasználók legtöbbször a memóriahiány gondjába futnak bele. A rendszer ugyan nagy mennyiségű szabad memóriával rendelkezik, az alkalmazás mégsem képes adott méretű puffer számára memóriát foglalni. A hiba oka ez esetben a memória töredezése, és jelenleg sajnos minden μClinux alapú megoldás szenved tőle. A VM-kezelés hiánya miatt μClinux alapú környezetben a memória teljes kihasználása a töredezés miatt gyakorlatilag lehetetlen. Legjobb, ha nézünk erre egy példát. Tegyük fel, hogy rendszerünk 500 KB szabad memóriával rendelkezik, és egy alkalmazás betöltéséhez 100 KB-ra van szükségünk. Ez egy egészen hétköznapi eseménynek mondható. Ne feledjük azonban, hogy az igény kielégítéséhez 100 KB-nyi összefüggő memóriaterülettel kell rendelkeznünk. Tegyük fel, hogy a memóriatérkép a következőképpen alakul. Minden karakter körülbelül 20 KB-nyi területet szimbolizál, az X-ek a rendszermag vagy más programok által lefoglalt vagy használt részeket jelzik:

```
0     100   200   300  400   500   600   700  800   900   1000
 -+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
  |xxxxx|xxxxx|---xx|--x--|-x---|xx---|-x---|-xx--|-x---|xxxxx|
```



2. ábra uClinux futtatása Xcopilot alatt (Palm emulátor)

Mint látjuk, van ugyan 500 KB-nyi szabad területünk, ám a legnagyobb összefüggő szakasz is mindössze 80 KB-os. Ilyen helyzet többféle módon is kialakulhat. A leggyakoribb ok, hogy egy program lefoglal valamennyi memóriát, majd nagy részét felszabadítja, és ekkor egy kisebb lefoglalt rész marad egy nagyobb szabad blokk közepén. A rövid ideig futó programok ugyancsak befolyásolhatják a memóriafoglalások helyét és módját. A μClinux `page_alloc2` memóriafoglalója rendelkezik egy kapcsolóval, amely pontosan az ilyen gondok elkerülését segíti. Ez egy új `/proc` bejegyzést hoz létre, a `/proc/mem_map`-et, amely a lapokról és foglalási csoportosításukról tájékoztat. Részletes ismertetésétől helyhiány miatt eltekintek, de a `page_alloc2.c`-hez a rendszermag forrásában kielégítő leírást lehet találni.
Gyakori kérdés, hogy miért nem töredezettség-mentesítjük a memóriát, hogy ily módon be tudjuk tölteni a 100 KB-os alkalmazást. A baj az, hogy nincs VM-kezelés, vagyis nem tudjuk áthelyezni a programok által használt memóriaterületeket. A programok általában különféle hivatkozásokat tesznek a lefoglalt területeken belüli címekre, és VM-kezelés nélkül a memóriának mindig a megadott helyen kell elérhetőnek lennie. Ha átmozgatjuk a programokat, egyszerűen összeomlanak. A μClinux ezt a gondot nem tudja megoldani. A fejlesztőknek tudniuk kell róla, és lehetőség szerint kisebb foglalási blokkokat kell használniuk.

Alkalmazások és folyamatok

A VM-kezeléssel rendelkező Linuxok és a μClinux között további fontos különbség a `fork()` rendszerhívás hiánya. Ha egy fejlesztő `fork()` hívást alkalmazó programot szeretne átültetni, bizony sokat kell dolgoznia vele. μClinux alatt az egyetlen lehetőség a `vfork()` használata. Ugyan a `vfork()` sok tekintetben megegyezik a `fork()` függvénnyel, éppen a különbségek számítanak a legtöbbet. Ha valaki nem ismerné a `fork()` és a `vfork()` rendszerhívást: segítségükkel egy folyamat két folyamatra, egy gyermekre és egy szülőre osztódhat. Egy-egy folyamat tetszőleges számú alkalommal osztódhat, ha több gyermeket is létre kell hoznia. Amikor egy folyamat elvégzi a `fork()` hívását, a gyermek minden tekintetben a szülő másolata lesz,

ám semmin nem osztozik vele, és mind ő, mind szülője függetlenül futnak tovább. A `vfork()` esetében más a helyzet. Először is, a szülő felfüggesztésre kerül, futása leáll, amíg a gyermek ki nem lép vagy meg nem hívja az `exec()` függvényt, az új alkalmazás indítására szolgáló rendszerfüggvényt. A gyermek a `vfork()` visszatérése után a szülő vermén fut, illetve a szülő memóriáját és adatait használja. A gyermek tehát akár meg is rongálhatja szülőjének vermét vagy adatszerkezeteit, ami nyilván hibát okoz. A bajt úgy kerülhetjük el, hogy biztosítjuk, a `vfork()` meghívása után a gyermek soha ne térjen vissza a jelenlegi veremkeretből, és munkájának végén az `_exit` függvényt hívja meg. Az `exit` azért nem használható, mert módosítja a szülő adatszerkezeteit. A gyermeknek tartózkodnia kell az átfogó adatszerkezetekben vagy változókban tárolt adatok módosításától, ezek a változtatások ugyanis ellehetetleníthetik a szülő működését. Egy alkalmazás átírása a `vfork` használatára a `fork` helyett a legtöbb esetben vagy gyermekien egyszerű, vagy borzalmasan nehéz. Általánosan elmondható, hogy ha az alkalmazás a `fork()` hívása után nem hívja meg gyakorlatilag azonnal az `exec()` függvényt, akkor a `fork()` – `vfork()` csere elvégzése előtt gondosan át kell vizsgálni. A µClinux egyszintű futtatható formátuma, bár közvetlenül nem érinti az alkalmazásokat és működésüket, enged néhány olyan műveletet, amelyet a normál linuxos ELF futtatható állományok nem. Az egyszintű formátum kétféle változatban létezik, teljesen áthelyezett és helyfüggetlen kód (***position-independent code***, PIC) változatban. A teljesen áthelyezett változat a kódjához és az adatokhoz egyaránt rendelkezik áthelyezésekkel, míg a PIC változat csak az adatokhoz használ néhányat. A beágyazott rendszerek fejlesztői számára az egyik legelőnyösebb szolgáltatás a helyben futtatás lehetősége (***execute-in-place***, XIP). Használatakor az alkalmazás közvetlenül Flash memóriáról vagy ROM-ból fut, valóban csak minimális, adatainak elhelyezésére elegendő memóriát igényelve. Ezzel a megoldással az alkalmazás kódja vagy szöveges részei több példány között is megoszthatók. Az XIP támogatása nem minden µClinuxos gépen megoldott, ugyanis a fordító részéről is megfelelő támogatást igényel, valamint PIC formátumú futtatható fájlok használatát teszi szükségessé. Amíg tehát adott géptípus eszközkészlete képtelen a PIC kezelésére, addig az XIP támogatásáról sem lehet szó. Jelenleg csak m68k és ARM rendszereken létezik az XIP használatához szükséges fejlettségű támogatás az egyszintű formátumhoz. A `romfs` az egyetlen fájlrendszer, amely képes az XIP µClinux alatti támogatására, ugyanis az XIP használatához az alkalmazásokat összefüggően kell tárolni a fájlrendszerben. Az egyszintű formátum az alkalmazások vermének méretét is megadja a fejállományban (***flat header***) mező formájában. Ha növelni kell egy alkalmazás vermének méretét, akkor csupán ezt a mezőt kell átírni. Erre az `flthdr` parancs használható, a következő módon:

```
flthdr -s egyszintű_futtatható_állomány
```

Az egyszintű formátum kétféle tömörítés használatát is lehetővé teszi. A teljes futtatható állományt tömörítve tudunk a legjobban takarékoskodni a ROM-mal. Azzal a kellemes mellékhatással is jár, hogy az alkalmazás teljes egészében egyetlen összefüggő RAM-területre töltődik be. Dönthetünk a csak az adatszegmensre kiterjedő tömörítés használata mellett is. Erre akkor lehet szükség, ha takarékoskodni szeretnénk a ROM-területtel, de XIP-et is használni szeretnénk. Az

```
flthdr -z egyszintű_futtatható_állomány
```

paranccsal teljesen tömörített futtatható állományt készíthetünk, míg a

```
flthdr -d egyszintű_futtatható_állomány
```

paranccsal az adatszegmensre korlátozhatjuk a tömörítést.

Megosztott könyvtárak

A megosztott könyvtárakról csak érintőlegesen szólhatok, de annyit mindenképpen érdemes tudni róluk, hogy µClinux alatt teljesen más a használatuk. A jelenleg elérhető megoldások a fordítóprogram részéről módosításokat, a fejlesztő részéről pedig kiemelt figyelmet igényelnek. Ha megosztott könyvtárat akarunk létrehozni, legjobb, ha egy példával kezdünk. A jelenlegi µClinux terjesztések az ***uC-libc*** és az ***uClibc*** könyvtárakhoz egyaránt kínálnak megosztott könyvtárakat. Megosztott könyvtárat létrehozni nem nehéz, és mindkét könyvtár jó és könnyen érhető példát mutat rá. Mielőtt bárkinek is túl nagy várakozásai lennének, megemlíteném, hogy a GCC `-shared` kapcsolójának használata nem része a megosztott könyvtárak létrehozásának, tehát senki ne gondolja, hogy tapasztalatból meg tudja oldani a feladatot. µClinux alatt a megosztott könyvtárak ugyanolyan egyszintű futtatható állományok, mint az alkalmazások, és tényleges megosztásukhoz helyben futtathatóra kell fordítani őket. A helyben futtatás lehetősége nélkül a megosztott könyvtárak minden őket használó alkalmazáshoz külön példányban indulnak el, ami rosszabb, mint ha befordítanánk őket alkalmazásainkba.

Összegzés

µClinuxra váltani Linuxról sokszor nem csupán a kétféle rendszer közötti különbségek kezeléséről szól. A µClinux rendszerek egyre mélyebben beágyazott rendszerek, sokszor meglehetősen kevés memóriával, kisebb ROM-mal és szokatlan eszközökkel kiegészítve. A merevlemez elmaradása és a szigorú erőforráskorlátok, párosulva a memóriavédelem hiányával és az eltérések szövevényes rendszerével sokszor a vártnál nehezebbé teszik a µClinux világában tett első kirándulást. Kezdetben a legjobb az, ha megismerkedünk a µClinux emulátoraival (*2. ábra*) vagy olcsó eszközöket szerzünk be. Remélem, hogy a fontosabb kérdésekre rámutatva sikerül elérnem, hogy az óvatosabb fejlesztők felkészültebben kezdjenek bele a µClinux megismerésébe, és el tudják kerülni a leggyakoribb csapdákat és félreértéseket.

Linux Journal 2004. július, 123. szám

David McCullough vezető programmérnök és veterán beágyazott-program fejlesztő. A SnapGear és a Lineo előtt a Stallion Technologies alkalmazásában állt, ahol programozási és mérnöki vezető volt, és SCO és BSD UNIX alapú termékek fejlesztésében vett részt.