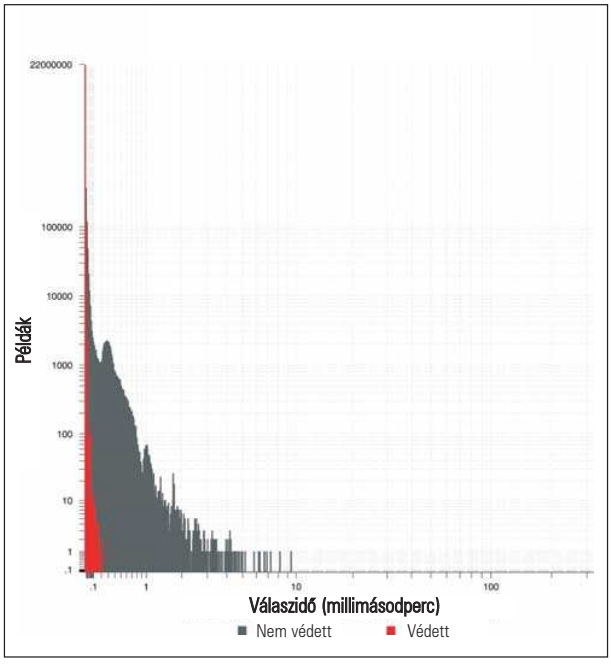
Védett processzorok: valós idejű teljesítmény szabványos Linux alatt

Tartsunk fenn egy processzort a nagy fontosságú feladatok számára, és máris magasabb szinvonalú valós idejű szolgáltatásokat biztosíthatunk.

A többprocesszoros rendszerekben a védett processzor olyan egység, amelyet a nagy fontosságú, valós idejű folyamatok futtatására tartanak fenn. A védettként kiválasztott processzor erőforrásai teljes egészében a nagy fontosságú folyamatok futtatását szolgálják. A védett processzorok futtatási környezete biztosítja a valós idejű alkalmazások üzemeltetéséhez szükséges előre jósolhatóságot.
Másként fogalmazva, ha rendelkezünk egy védett processzorral, akkor garantáltan gyors választ tudunk adni a külső megszakításokra, és előre meghatározható működésű környezetet tudunk biztosítani a valós idejű folyamatok futtatásához.

Linux és védettség

Korábban védett processzort csak szimmetrikus többprocesszoros rendszerekben lehetett kijelölni. A hiperszálas (***hyperthreading***), vagyis logikailag egynél többnek látszó processzorok megjelenésével azonban már egyprocesszoros rendszerben is kijelölhetünk védett processzort.
A védett processzoros megközelítéssel a fejlesztők a valós idejű futtatási környezetekéhez hasonló eredményeket érhetnek el.
Az eredmények összehasonlíthatóak az RTAI vagy RT/Linux környezetben mértekkel, ahol a Linux egy valós idejű futtatási környezetben önálló folyamatként kel életre.
Ha tisztán linuxos környezetet használunk az alkalmazások fejlesztéséhez, akkor jó néhány előnyt élvezhetünk az ilyen jellegű futtatási környezetekhez képest. Például a Linux számos illesztőprogramot támogat, így a teljes értékű alkalmazási megoldások megvalósításának költsége lényegesen alacsonyabb. A magas szintű programozási nyelvek széles választékával az alkalmazások elkészítése is lényegesen hatékonyabb.
Kereskedelmi alkalmazásoknál ez fontos tényező, és bár a valós idejű rendszerek tervezésénél nem feltétlenül a programozás hatékonysága a legfontosabb, a fejlesztési fázisban mégis komoly segítséget jelenthet, valamint a végső termék szolgáltatásainak bővítéséhez is hozzájárulhat.
Mindemellett a Linux összetett protokollkészletekkel – mint például a CORBA –, kiterjedt grafikai képességekkel és fejlett alkalmazásfejlesztői eszközökkel rendelkezik. A normál Linux-terjesztésekben elérhető szolgáltatásokon túl a Linux tudása – köszönhetően a mögötte álló közösség erejének – napról napra nő.
Ha az alkalmazások tervezésekor a Linuxot választjuk alapnak, a felhasználó a jövőben lényegesen több választási lehetőséggel élhet majd.



1. ábra A védett processzorral és az anélkül mért válaszidők összehasonlítása

A valós idejű működés megjósolhatóságot jelent, nem sebességet!

Valós idejűnek azt az alkalmazást nevezzük, amelynek a külvilág eseményeire kell válaszolnia, és a műveleteket adott határidőn belül kell befejeznie.
A határidő letelte után adott helyes válasz nem számít helyes válasznak. Maguk a határidők az alkalmazástól függnek,

néhány mikromásodperc és több másodperc között változhatnak. A szigorúan valós idejű alkalmazásoknál a határidőket be kell tartani. Kizárólag a rendszer által a legrosszabb esetben teljesített mutatók érdekelnek bennünket, ugyanis ezek azok az esetek, amelyekben a határidők elmulasztása előfordulhat.

Mivel a való világ eseményeiről a számítógépes rendszer megszakítások révén értesül, egy valós idejű operációs rendszernek a legrosszabb esetben is adott időn belül garantáltan válaszolnia kell a megszakításokra.

Ha ez sikerül, és át tudjuk adni a vezérlést a megfelelő valós idejű alkalmazásnak, már meg is tettük az első lépést a határidő teljesítése felé.

Ha a valós idejű alkalmazás már fut, a rendszernek előre meghatározható futási időt is garantálnia kell az alkalmazás számára. Ha a valós idejű alkalmazás futtatásának ideje túlságosan széles tartományban ingadozik, a határidőket túllépjük.

Ha a megszakításokra jó ütemben akarunk válaszolni, az operációs rendszernek képesnek kell lennie arra, hogy megszakítás fellépése esetén bármely futó programtól azonnal el tudja venni az erőforrásokat. Mivel a 2.4-es sorozatú Linux nem engedi a feladatoknak, hogy más, a rendszermagon belül futó feladatoktól elvegyék az erőforrásokat, ez a sorozat legrosszabb esetben meglehetősen gyenge válaszidőket biztosít. Létezik persze egy folt, amely lehetővé teszi a rendszermagon belül futó feladatok erőforrásainak elvételét. Sajnos ezt hiába telepítjük, maradnak olyan rejtett tényezők, amelyek a megszakításokra adott válaszok jelentős elhúzódását okozhatják.

Az operációs rendszer feladata az, hogy több, a rendszer erőforrásain osztozó feladat végrehajtását irányítsa. A megosztott erőforrások jellemzőit tartalmazó adatszerkezetek megsérülhetnek, ha egyszerre több program is használja őket.

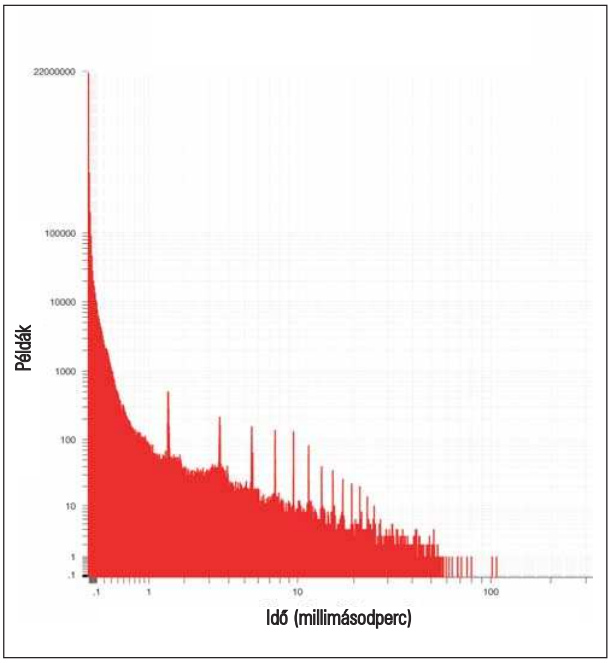
Ebből következően minden operációs rendszernek vannak olyan kritikus kódrészletei, amelyeket a programok csak egymás után érhetnek el.

Ha egy nagy fontosságú feladat – megszakítás fellépése miatt – hirtelen futtathatóvá válik, a processzor nem adható át neki azonnal, ha az éppen futó program pont egy ilyen kritikus szakaszban van.

A hosszúra nyúló kritikus szakaszok tehát jelentős mértékben befolyásolják a rendszer megszakításokra való válaszadási képességeit. Az alacsony késleltetéseket biztosító foltok megfelelő algoritmusok segítségével a hosszú kritikus szakaszok egy részét eltüntetik a Linux rendszermagból, illetve lerövidítik azokat.

Általában elmondható, hogy minél bonyolultabb egy alrendszer, annál hosszabbak a kritikus szakaszok. Mivel a Linux számos összetett alrendszert támogat, köztük fájlrendszereket, hálózati és grafikai alrendszereket, kritikus szakaszai rendkívül hosszúak, legalábbis egy kisméretű, valós idejű operációs rendszerhez hasonlítva.

Az erőforrások elvételét lehetővé tévő folt és az alacsony késleltetéseket garantáló foltok jelentős mértékben javítottak a Linux válaszidőin. A kritikus szakaszok ennek ellenére több tíz msec időtartamot is felölelhetnek, márpedig sok valós idejű alkalmazás számára az ilyen válaszidők elfogadhatatlanok.



2. ábra Válaszidők (kernel.org 2.4.21-pre4)

Mi az a védett processzor?

Mint korábban már említettem, a védett processzort a nagy fontosságú feladatok futtatására és a hozzájuk tartozó megszakítások kezelésére tartjuk fenn.

Védett processzor megadásának előfeltétele, hogy az operációs rendszer képes legyen a folyamatok és a megszakítások affinitásának (futtató processzorának) beállítására. A 2.4-es sorozatszámú Linux képes a megszakítások affinitásának beállítására, és ugyanez a képessége folyamatokra a megfelelő nyílt forrású foltok segítségével biztosítható. (Lásd: *„Processzorhoz kötés”*, Linuxvilág, 2003. szeptember).

A védett processzor előnyei

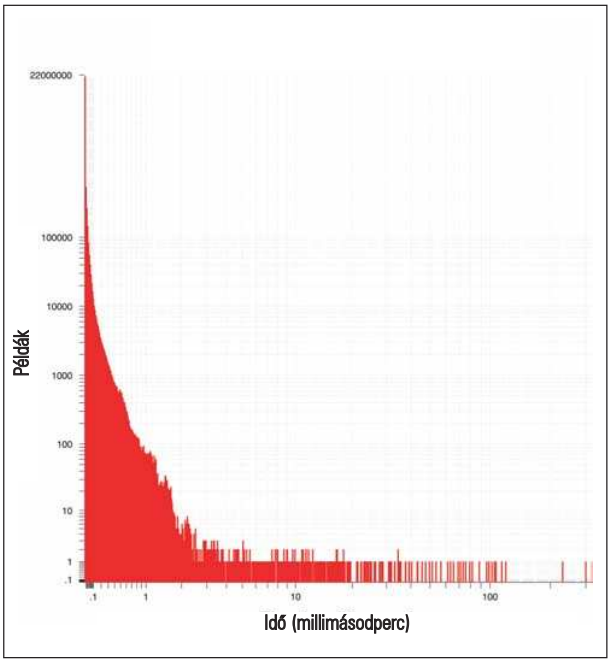
Mivel a védett processzorok nem futtatnak háttérfolyamatokat, a védett processzorokra kerülő nagy fontosságú feladatoknál nincs meg annak a veszélye, hogy azért ne tudnának válaszolni egy megszakításra, mert egy másik, éppen kritikus szakaszban lévő folyamat köti le a processzort.

A megszakítások mindig nagyobb fontosságot kapnak, mint bármely folyamat, és mivel fellépésük előre nem jósolható meg, a nem valós idejű megszakítások jelentős bizonytalansági tényezőt hozhatnak a folyamatok előre megbecsült futtatási idejébe.

Egy védett processzor nem foglalkozhat a megszakítások kezelésével, hacsak az adott megszakítás nem egy nagy fontosságú, a védett processzorhoz rendelt feladathoz tartozik.

Védett processzoros rendszer megvalósítása

Ha folyamatok és megszakítások affinitását egyaránt meg tudjuk adni, akkor olcsón meg tudjuk oldani a kiszemelt processzor védelmét is. Ez a megoldás azonban csak akkor működőképes, ha egyik folyamat sem változtatja meg saját affinitását úgy, hogy a védett processzort is igénybe vegye.



3. ábra Válaszidők (Red Hat 8.0)

Általánosságban tehát ennél erőteljesebb védelemre van szükség – alábbiakban egy ilyen megoldást fogok ismertetni.
Processzor védelmét a */proc* felületen keresztül írhatjuk elő, itt a rendszergazda egy maszkkal választhatja ki a védett processzorokat, valamint egy megfelelő paranccsal a maszk módosítását is megoldhatja.
A felület lehetővé teszi a processzorok dinamikusan történő védetté nyilvánítását.
Ha egy processzort védettként jelölünk meg, akkor egyik folyamat sem módosíthatja saját affinitását úgy, hogy a védett processzort is igénybe vegye, feltéve, hogy a tiltás figyelembe vételével marad olyan processzor, amelyen a folyamat futhat.
A felhasználóknak tehát kifejezetten ki kell választaniuk a védett processzort folyamataik futtatására, ha igénybe akarják venni annak erőforrásait.
Saját affinitásmaszkjához csak privilegizált folyamat adhat hozzá processzort.
Ennél a megvalósításnál módosítani kell a folyamatok affinitását beállító programkódot. A `sys_sched_setaffinity()` függvénnyel processzoraffinitást lehet beállítani. A függvény módosításával eltávolíthatjuk a védett processzort minden felhasználóra specifikus maszkból, amikor az affinitást beállítottuk :

```
p->cpus_allowed_user = new_mask;
if (new_mask & ~shielded_proc)
    new_mask &= ~shielded_procs;
set_cpus_allowed(p, new_mask);
```

Megjegyzem, a védett processzort jelölő bitek nem törlődnek, ha ezáltal a folyamat futtatása mindegyik processzoron tilossá válna. A `cpus_allowed_user` egy új mező a `task` adatszerkezetben, amely az eredeti, a felhasználó által megadott affinitást tárolja.

Amikor a védett processzorok maszkja megváltozik, a fenti kódot a rendszer összes folyamatára meg kell hívni.
Ehhez kell tudni a felhasználó által eredetileg beállított processzoraffinitást. A védett processzor maszkjának megváltoztatására alkalmas kód a következőképpen néz ki:

```
for_each_task(p) {
   new_mask = p->cpus_allowed_user &
cpu_online_map;
   if (new_mask & ~shielded_proc)
      new_mask &= ~shielded_procs;
   if (new_mask != p->cpus_allowed)
      set_cpus_allowed(p, new_mask);
}
```

Teljesítménypróbák

A megszakításokra adott válaszok megadási idejének mérésére az ***Andrew Morton*** weboldalán talált Realfeel teljesítménymérő programot használtuk.
Azért erre a programra esett a választásunk, mert a valós idejű óra (***Real Time Clock***, RTC) illesztőprogramot használja, és ezt több Linux-változatban is alkalmazzák megszakítások létrehozására.
A program az RTC illesztőprogram által létrehozott megszakításra adott válaszok megadási idejét méri. Az RTC-t megszakítások periodikus előállítására használjuk, pontosan 2048 Hz-es frekvenciával.
Egy `read` rendszerhívást is támogat, amely a következő megszakítás létrehozásakor tér vissza. A válaszidő mérésére az IA-32 TSC időzítőt érdemes használni, ennek felbontása a processzor órajelétől függ.
A megszakításokra adott válaszok létrejöttének idejét úgy tudjuk mérni, hogy először kiolvassuk a TSC értékét, majd ismételt `read` hívásokat indítunk a */dev/rtc*-re.
Amikor minden `read` hívás véget ért, a program kiolvassa a TSC aktuális értékét.
A két egymást követő TSC érték különbsége adja meg, hogy az RTC megszakításra várva a folyamat mennyi ideig volt blokkolva. A várt válaszidő 1/2048 másodperc.
Minden a várt válaszidőn túli idő lappangási időnek számít a megszakítás kezelését tekintve.
Ha mérni akarjuk, hogy legrosszabb esetben mennyi idő alatt válaszol a rendszer a megszakításokra, akkor aktív háttérterhelést kell rónunk rá.
A háttérterhelésnek akkorának kell lennie, hogy a rendszer csak késve tudjon reagálni a megszakításokra, és a futtatás előre jósolhatóságát rontó versengést kell okoznia az erőforrásokért. A háttérterhelés előidézésére a Red Hat ***stress-kernel*** csomagja tökéletesen megfelel.
A csomagból a következő programokat választottuk ki: TTCP, FIFOS_MMAP, P3_FPU, FS és CRASHME.
A TTCP nagyméretű adatcsomagokat küld és fogad a hurokeszközön keresztül.
A FIFOS_MMAP több próba kombinációja, felváltva cserél adatokat két folyamat között FIFO elven, illetve végez műveleteket egy `mmap`-pal kezelt fájlon. A P3_FPU próba lebegőpontos mátrixokon végez különféle műveleteket.
Az FS próba egy maréknyi fájlon változatos műveleteket végez, például nagyméretű, középen üres fájlokat hoz lét-

re, majd csonkolja és bővíti ezeket. Végül a CRASHME próba véletlenszerű adatokat tartalmazó puffereket hoz létre, majd ezekre az adatokra ugrik, és megpróbálja futtatni őket.
Bár Ethernet-forgalom nem keletkezik a rendszeren, a hálózati kapcsolat fennmarad, és a gépnek a próbák futtatása közben is kezelnie kell a normál szórásos forgalmat.
A ***stress-kernel*** csomagban szereplő NFS_COMPILE próba egy újabb változatát használtuk, az eredeti erőforrás-felszabadítási hibái miatt ugyanis a próbát nem lehetett hosszabb ideig futtatni.
Az NFS_COMPILE parancsfájl újra és újra lefordítja a rendszermagot, miközben egy a helyi hurokeszközön keresztül elérhetővé tett NFS fájlrendszert használ.
A próbák futtatására egy 1 GB memóriával és SCSI merevlemezzel felszerelt kétprocesszoros Pentium 4 Xeon gépet használtunk.

Eredmények

A védett processzor megszakításokra adott válaszainak idejét a ***Concurrent Computer Corporation*** által készített RedHawk Linux 1.3-as változata alatt mértük. A RedHawk 2.4.21-es rendszermagra épülő Linux rendszer. Érdemes megjegyezni, hogy a RedHawk Linux rendszermag különleges szolgáltatásai közül csak az egyik a védett processzorok kijelölésének lehetősége.
Az alább közölt eredmények eléréséhez – kisebb-nagyobb mértékben – az egyéb fejlesztések is hozzájárultak. A rendszermag például több nyílt forrású foltot is tartalmaz, köztük ***Robert Love*** erőforrások elvételét lehetővé tévő foltját, ***Andrew Morton*** alacsony késleltetést biztosító kiegészítését valamint a 2.5-ös Linux-ág O(1) ütemezőjét.
A teljesítménypróbák eredményét további módosítások is befolyásolhatták, így például azok az algoritmus-változtatások, amelyek a Linux rendszermag kritikus szakaszainak lerövidítését szolgálják; és az, amelyik az alsóbb megszakítások állítható fontosságú és ütemezési házirenddel szabályozható rendszermag démonon belüli kezelését teszi lehetővé.

Red Hawk és Red Hat

Az *1. ábrán* látható, hogy RedHawk Linux alatt védett processzor használatával illetve anélkül milyen eredményeket sikerült elérni. A két eredmény közötti eltérés szembeszökő. A rendszer a legtöbb esetben mindkét módban kevesebb mint 100 mikromásodperc alatt képes volt válaszolni az RTC megszakításokra.
Általában tehát elmondható, hogy a Linux kellő gyorsasággal válaszol a megszakításokra. Csakhogy, mint már említettem, a valós idejű rendszereknél a legfontosabb mutató a legrosszabb esetben adott válaszidő. Ennek oka az, hogy éppen ilyenkor fordulhat elő, hogy egy valós idejű alkalmazás túllépi a megadott határidőt.
Védett processzoros módban a legrosszabb esetben is 220 mikromásodperc alatt megérkezett a válasz az RTC megszakításra. Amikor nem használtunk védett processzort, akkor a felső határ 10 msec volt, ami egy nagyságrenddel nagyobb, mint a védett processzoros mód legrosszabb eredménye.
Ugyan a válaszidő az esetek kevesebb mint egy százalékánál volt nagyobb mint 200 mikromásodperc, több ezer esetben az 500 mikromásodperc időtartamot is meghaladta. Egy valós idejű rendszerben ezek az esetek nagy valószínűséggel egy-egy elmulasztott határidőt jelentenek.
Ugyanezt a megszakításpróbát egy módosítások nélküli 2.4.21-es kernel.org rendszermagon (*2. ábra*), valamint egy 8.0-s Red Hat terjesztésen (*3. ábra*) is lefuttattuk. Ez a Red Hat rendszermag nem tartalmazza az erőforrások elvételét lehetővé tévő foltot, de az alacsony késleltetéseket biztosítót igen, vagyis esetében túl hosszúra nyúló kritikus szakaszokról nem beszélhetünk.
Mivel védett processzort ezek közül a rendszermagok közül egyik alatt sem lehet kijelölni, esetükben eredményeket csak a nem védett módnál tüntettünk fel.
Ezek a rendszermagok a RedHawk rendszermaggal mértekhez hasonló, általában 100 mikromásodperc alatti válaszidőket mutatnak.
A legrosszabb eset azonban messze kedvezőtlenebb, mint a RedHawk Linux akár védett processzor nélküli teljesítménye, a ***kernel.org*** Red Hawk összeállítása 107 msec, a Red Hat pedig 323 msec alatt válaszol, ha minden kötél szakad.
Az eredmények cseppet sem meglepők, hiszen ezeket a rendszermagokat inkább az erőforrások egyenlő folyamatok közötti elosztására tervezték, és feladatuk inkább általánosan jó rendszerjellemzők, és nem valós idejű válaszidők biztosítása.

Összefoglalás

Egyértelmű, hogy védett processzor kijelölésével jelentős javulást tapasztalhatunk a linuxos rendszerek legrosszabb esetben adott válaszidőinek tekintetében.
A védett processzorok alkalmazása hatékony megoldás, mivel így a rendszer legfontosabb feladatai számára erőforrások tarthatók fenn.
Mindezt a Linux normál alkalmazásprogramozási felületének módosítása nélkül érhetjük el.
Írásomban csak az RTC megszakításra adott válaszokról esett szó. Az RTC kiválasztását az indokolta, hogy a legtöbb Linux-változatban alapvető szolgáltatásnak számít. Más megszakításforrások és jobban optimalizált illesztőprogramok használatával ennél jobb garantált válaszidőket is el lehet érni.
Aki szeretne részletesebben megismerkedni a védett processzoros megoldásokkal, illetve szeretné látni, hogy jobb válaszidőket garantáló illesztőprogrammal milyen eredményeket lehet elérni, az tanulmányozza át a ➲ http://www.ccur.com/isddocs/wp-shielded-cpu.pdf dokumentumot.

Linux Journal 2004. május, 121. szám

Stephen Brosky a Concurrent Computer Corporation Integrált megoldások részlegének vezető kutatója. Tagja volt a valós idejű alkalmazás- és programszálfelületekre vonatkozó Posix 1003.1b és 1003.1c szabványokat elkészítő IEEE bizottságnak.