Az időosztásos rendszermag bemutatása

Ha csak magasabb pontszámot szeretnénk elérni a Quake-ben, netán audiorajongók vagyunk, esetleg simábban futó felületet szeretnénk – a rendszermag lappangási idejének csökkentése igen fontos cél.

A teljesítmény mérése két szempont: az áteresztőképesség (throughput) és a lappangás (latency) alapján történik. Az első olyan, mint az autópálya szélessége: minél szélesebb, annál több autó utazhat rajta. A második pedig a sebességkorláthoz hasonló: minél magasabb, annál hamarabb érnek az autók A pontból B-be. Feladatok esetében nyilvánvalóan mindkét érték fontos. Néhány munka azonban olyan lehet, hogy az egyik érték szükségesebbé válik a másiknál. Az autópályás hasonlatnál maradva a teherfuvarozás érzékenyebb lehet az áteresztőképességre, ugyanakkor a futárszolgálat számára a lappangás a fontosabb. E cikk fő témája pontosan ez a jó kis régivágású rendszermagfeladat: a lappangás csökkentése és a rendszer válaszidejének növelése.

Az audio-, illetve videofeldolgozás és -visszajátszás azon feladatok közé tartozik, amelyek sokat nyerhetnek az alacsonyabb lappangási idővel. A Linux számára egyre fontosabb, hogy az interaktivitás teljesítménye is növekedhet. Nagy lappangási idő esetén az egérkattintások és a hasonló felhasználói tevékenységek meglehetősen sokáig válasz nélkül maradnak, ami igen távol esik attól a pattogósságtól, amit a felhasználók elvárnak. A rendszer ugyanis a fontos folyamatokat nem tudja elég gyorsan elérni.

A gond – legalábbis a rendszermag vonatkozásában – az, hogy nem időosztásos módon működik. Általában ha valami fontos történik, például bekövetkezik egy interaktív esemény, a fogadóalkalmazás fontosságnövelést kap, következésképpen futni fog. Így működnek az időosztásos módon többfeladatosított (multitask) operációs rendszerek. Az alkalmazás addig fut, amíg egy adott időmennyiséget fel nem használ (ezt nevezik időszeletnek), vagy valamilyen más fontos esemény nem történik. A másik lehetőség az együttműködő többfeladatos módszer (cooperative multitasking), ahol – mielőtt egy új folyamat elkezdhetne futni – az alkalmazásoknak maguknak kell jelenteniük, hogy „kész vagyok!”. A gond az, hogyha valami a rendszermagban fut, az ütemezés gyakorlatilag ilyen együttműködő típusú lesz.

Az alkalmazások, akár a felhasználótérben futva hajtják végre saját kódjukat, akár a rendszermagban hajtanak végre rendszerhívásokat, vagy más módon dolgoztatják a rendszermagot, e két módszer valamelyike szerint működnek. Amikor egy folyamat a rendszermagban dolgozik, addig fut, amíg úgy nem dönt, hogy megáll – eközben az időszeleteket és a fontos eseményeket figyelmen kívül hagyja. Hiába válik egy még fontosabb folyamat futtathatóvá, akkor sem tud lefutni addig, ameddig a jelenleg futó folyamat ki nem lép, már ha épp a rendszermagban tartózkodik. Márpedig ez a folyamat milliszekundumok százait fogyaszthatja.

Lappangási megoldások

Az első és legegyszerűbb megoldás a lappangás problémájára, ha újraírjuk a rendszermag algoritmusait, hogy azok a lehető legkevesebb kötött időmennyiséget használják fel. Ezzel csak az a gond, hogy a korábbi cél is ez volt: a rendszerhívásokat olyanra írták, hogy lehetőleg gyorsan térjenek vissza a felhasználótérbe, s lám, mégis akadnak lappangási gondok. Vannak ugyanis olyan algoritmusok, amelyek egyszerűen nem méretezhetők jól.

Az áteresztőképesség próbája

15 „dbench 16” átlaga

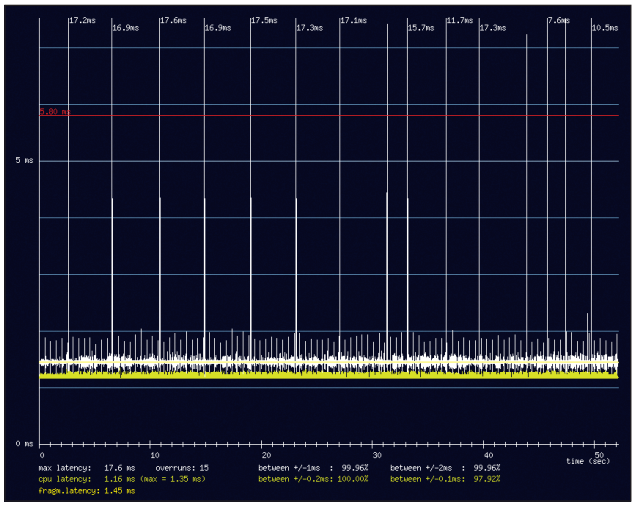
Rendszermag	Áteresztőképesség
2.5.2	24,3813 MB/mp
2.5.2-preempt	28,5920 MB/mp

A második megoldás, hogy közvetlen módon időzítőpontokat szúrunk a rendszermagba. Ezen elképzelés szerint (melynek ötlete a kis lappangási foltokból származik), meg kell keresni a rendszermag problémás pontjait, és minden ilyen helyre be kell illeszteni egy olyan kódot, amelynek „Futni szeretne valaki? Ha igen, hadd fusson!” hatása van. Ezzel a megoldással azonban az a helyzet, hogy nemigen remélhetjük, hogy az összes lehetséges problémás helyet megtaláljuk és kijavíthatjuk. Ennek ellenére – a próbák tanúsága szerint – ezek a foltok meglehetősen jó munkát végeznek. Nekünk azonban nem gyors javításra, hanem a gond hosszútávú megszüntetésére van szükségünk.

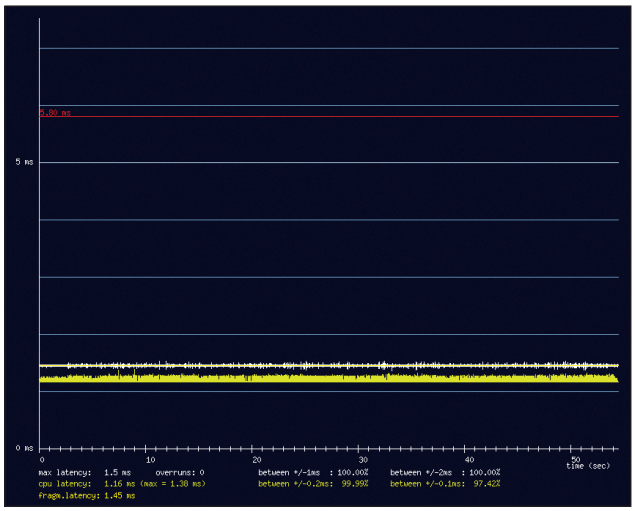
Az időosztásos rendszermag

Sokkal jobb megoldás, ha a rendszermagot időosztásosan készítjük el, hiszen ezzel a gondot teljes egészében megszüntetjük. Így ha valami fontosabbnak kell futnia, le is fog futni, függetlenül attól, hogy a jelenlegi folyamat éppen mit csinál. Az egyetlen buktató és egyben az ok, amiért Linux nem így lett már a kezdetektől megírva, az, hogy a rendszermagnak újrahívhatónak kell lennie. Szerencsére az időosztásosság gondjait a már létező SMP- (Symmetric Multiprocessing) támogatás megoldotta. Az SMP-kód előnyeit kihasználva és néhány egyszerű módosítással kiegészítve a rendszermag időosztásossá tehető.

MontaVista-i programozók mutatták be először a rendszermag időosztásos megvalósítását. Első lépésként a `spin lock` meghatározását változtatták meg úgy, hogy a „nem időosztható” tartományokat tartalmazza. `Spin lock` alatt ugyanis nincs szükségünk erre a tulajdonságra, éppen úgy, ahogy SMP alatt sem érünk el egy adott zárolt területet azonos időben több helyről. Természetesen egyprocesszoros rendszereken valójában semmi másra nem használjuk a `spin lock`-okat, csakis az időosztásosság jelzésére. Másodszor a kódot úgy módosították, hogy a kényes részeken vagy magában az ütemezőben semmiképpen ne legyen időosztásos. Végül a megszakításból való visszatérés kódfáját úgy változtatták meg, hogy a pillanatnyi folyamatot – amennyiben szükséges – ütemezze újra.



1. kép
Lappangási próba mérési eredményei hagyományos rendszermag esetén



2. kép
Lappangási próba mérési eredményei időosztásos mag esetén

UP (uniprocesszor, vagyis egyprocesszoros) módban a `spin_lock` `preempt_disable`-ként, a `spin_unlock` pedig `preempt_enable`-ként lett meghatározva. SMP módban ezen kívül a normál zárolást is elvégzik. Mit is csinálnak ezek az új eljárások?
A `preempt_disable` és `preempt_enable` egymásba ágyazható időosztásjelzők a `preempt_count`-on változtatnak, amely új `integer` a `task_struct` szerkezetekben.
A `preempt_disable` lényegében a következő:

```
+current->preempt_count;
barrier();
```

a `preempt_enable` pedig:

```
--current->preempt_count;
barrier();
if (unlikey(!current->preempt_count
&& current->need_resched))
    preempt_schedule();
```

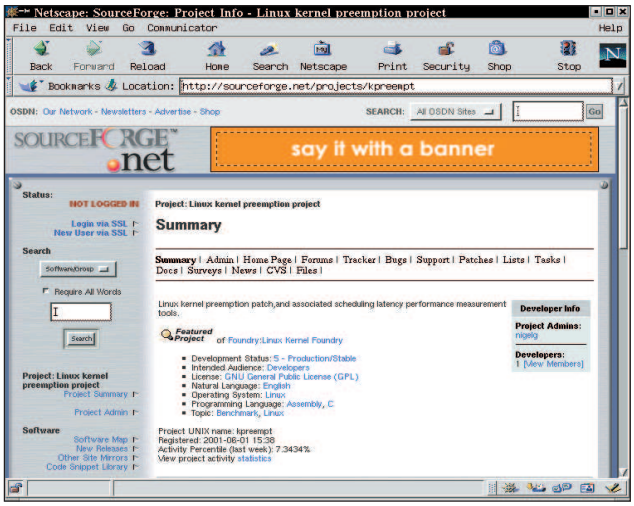
Ennek eredményképpen nem fogunk időosztást alkalmazni, amikor a számláló nullánál nagyobb, mivel a `spin lock`-ok már eleve a megfelelő helyeken vannak, hogy megóvják az SMP-gépek kritikus részeit, az időosztásos rendszermag pedig most már szintén rendelkezik ezzel a védelemmel.
A `preempt_schedule` magába az ütemezőbe való belépést valósítja meg. Beállítja a működő folyamat jelét (flag), hogy jelezze: ez a folyamat időosztásos volt, majd meghívja az ütemezőt, végül visszatéréskor leszedi a jelet:

```
asmlinkage void preempt_schedule(void)
{
    do {
        current->preempt_count +=
        ➥PREEMPT_ACTIVE;
        schedule();
        current->preempt_count -=
        ➥PREEMPT_ACTIVE;
    } while (current->need_resched);
}
```

A `preempt_schedule` másik bejárata a megszakítások visszatérő ösvényén keresztül vezet. Amikor a megszakításkezelő visszatér, ellenőrzi a `preempt_count` és a `need_resched` változókat, ugyanúgy, ahogy a `preempt_enable` is teszi (bár az ***entry.S***-ben található megszakítás visszatérő kód assembly nyelven íródott). Az lenne az eszményi, ha itt is időosztást idéznénk elő, hiszen egy megszakításról van szó, amely a `need_resched`-et általában valamilyen eszközesemény miatt állítja be. A megszakítást sajnos nem mindig tudjuk azonnal időosztásossá tenni, hiszen még zárolva lehet. Ez az oka annak, hogy az időosztás állapotát a `preempt_enable`-ben is ellenőrizzük.

Eredmények

Ezzel az időosztásos rendszermagfolttal a feladatokat nemcsak akkor ütemezhetjük újra, ha a felhasználótérben vannak, hanem azonnal, amint futtatni kell őket. Milyen hatása lesz mindennek?
A folyamatszintű válaszidő néhány esetben akár a huszadára is csökkenhet (lásd az *1. képen* bemutatott hagyományos rendszermagot szemben a *2. képen* látható időosztásos rendszermaggal). Ezek a grafikonok ***Benno Senoner*** igen hasznos lappangási próbaprogramjával készültek, amely a terhelés alatti audioátmenetitár-kezelést utánozza. Az ábra vörös vonala azt a lappangási értéket jelzi, amely felett az audiokihagyások már az emberi fül számára is észrevehetők. Látható, hogy az 1. képen több tüske is megfigyelhető szemben a 2. kép folyamatos, alacsony grafikonjával.
A lappangási próba szerint tehát a legrosszabb és az átlagos lappangás tekintetében egyaránt javulás mutatkozik. A további próbák kimutatták, hogy a rendszer átlagos lappangási ideje különféle terheléstípusok mellett most az 1–2 ms-os tartományba esik.
A leggyakoribb ellenvetés az időosztásos rendszermagokkal szemben a növekvő összetettség. Az összetettség pedig, mondják az ellenzők, csökkenti az áteresztőképességet. Szerencsére az időosztásos rendszermagfolt sok esetben még az áteresztőképességet is növeli (lásd az *1. táblázatot*). Ennek elméleti háttere az, hogy amikor az I/O-adat elérhetővé válik, az időosztásos rendszermag az I/O-hoz kapcsolt folyamatot sokkal gyorsabban életre tudja hívni. Az eredmény kellemes ráadásaként az áteresztőképesség is magasabb lesz. A főbb eredmények:



simábban futó grafikus felület, terhelés alatt kisebb audiokihagyások, jobb alkalmazás-válaszidő, és sokkal igazságosabb időosztás a magas elsőbbségű (priority) folyamatok számára.

A változások programozói szemszögből nézve

A rendszermagbetörők (hacker) valószínűleg elgondolkodnak rajta: hogyan hat mindez a kódomra? Mint korábban már említettük, az időosztásos rendszermag a már létező SMP-támogatáson alapul. Ez teszi lehetővé, hogy az időosztásos rendszermagfolt aránylag egyszerű legyen, és hogy a kódolási módszerekre gyakorolt hatása viszonylag kicsi maradjon. Egy változtatás azonban mindenképpen szükséges: jelenleg a processzoronkénti adatok (olyan adatok, amelyek minden processzor esetében egyediek) nem igényelnek zárolást. Mivel minden processzor esetében egyediek, a másik processzoron futó folyamat nem ronthatja el az első adatait. Az időosztás használatával azonban egyazon processzoron futó folyamatok is időosztássossá tehetők, s ilyenkor a második folyamat belegázolhat az első adataiba. Ez ellen normál esetben a meglévő SMP-zárak védik, de a processzoronkénti adatoknak nincs szükségük ilyen zárra. Azokat az adatokat, amelyek nem rendelkeznek zárral, mert természetüktől fogva védettek, „implicit módon zároltnak" nevezzük. Az implicit módon zárolt adatok és az időosztás nem fér meg együtt. A megoldás szerencsére egyszerű: az adat elérése körül ki kell kapcsolni az időosztást, például:

```
int catface[NR_CPUS];
preempt_disable();
catface[smp_processor_id()] = 1;   /* CPU
➥szerint indexelj k a  catface-t */
/* mßveletek a catface-en */
preempt_enable();
```

A jelenlegi Preemption-folt már tartalmazza a meglévő implicit módon zárolt adatok elérésének megfelelő módosítását. Szerencsére az ilyesmi viszonylag ritka. Azonban minden új kódnak védelemre van szüksége, amennyiben időosztásos rendszermagban akarjuk őket használni.

Továbbfejlesztési lehetőségek

Maradt még néhány feladatunk: miután a rendszermagot időosztássossá tettük, hozzáláthatunk a sokáig tartó zárolások futásidejének csökkentéséhez. Mivel a rendszermag nem lehet időosztásos, amíg a zár zárva van, a leghosszabb zárolások ideje fogja meghatározni a rendszer legrosszabb lappangási idejét. Minden munka, ami jótékony hatással van az SMP-méretezhetőségre (a finomabb felbontású zárolásra) egyúttal a lappangási időt is csökkenteni fogja. Megpróbálhatjuk újraírni az algoritmusokat és a zárolási szabályokat, csökkentve ezáltal a zárolási időt. A BKL eltüntetése szintén segíthet.
A hosszú ideig tartó zárolásokat felderíteni legalább olyan nehéz, mint újraírni őket. Létezik azonban egy `preempt-stats` folt, amely a nem időosztásos időszakokat méri és okait jelenti. Ez az eszköz igen hasznos lehet, ha adott terhelés mellett (például a Quake alatt) akarjuk meghatározni a lappangás okát. Amire szükség van, azt el kell érni. A rendszermagfejlesztőknek minden olyan zárolási időt újra meg kell vizsgálniuk, ami egy adott határértéket túllép. Egy viszonylag korszerű rendszeren ez körülbelül 5 ms-ot jelent. Ezt észben tartva rámutathatunk a magas lappangási és zárolási idejű területekre és kijavíthatjuk őket.

Összegzés

A Linux-közösség népes és igen sokrétű, a Linux felhasználási területe pedig a beágyazott rendszerektől egészen a nagykiszolgálókig terjed. Az időosztásos rendszermagmódszer előnyei a valós idejű alkalmazások területén is túlmutatnak.
Az asztali felhasználók, a játékosok és a multimédiafejlesztők egyaránt élvezhetik a csökkentett lappangási idő előnyeit.
A 2.4 és a 2.5 rendszermagfákhoz megoldás egyaránt szükséges – és elképzelhető, hogy nem a legszerencsésebb, ha mindkét változathoz azonos megoldást készítenünk. Mivel a 2.5-ös még fejlesztés alatt áll, itt az ideje, hogy beillesszenek egy azonnali előnyökkel járó képességet, amely egyúttal a további fejlesztésekhez is keretként szolgálhat. Eredményül jobb rendszermagot kaphatunk.

Linux Journal május, 97. szám

Robert Love
(rml@tech9.net) matematika és számítógépes tudományok szakos hallgató a Floridai Egyetemen. Amikor éppen nem Linuxot elemez, autóversenyzik, thai ételeket eszik vagy punk zenét hallgat.

Kapcsolódó címek

Benno Senoner lappangáspróbája
➲ http://www.gardena.net/benno/linux/audio

dbench ➲ http://ftp://samba.org/pub/tridge/dbench

Időosztásos rendszermagfolt
➲ http://www.kernel.org/pub/linux/kernel/people/rml/preempt-kernel

A Preemptive Kernel Project honlapja
➲ http://kpreempt.sourceforge.net

Alacsony lappangási folt
➲ http://www.zipworld.com.au/~akpm/linux/schedlat.html

MontaVista ➲ http://www.mvista.com

preempt-stats folt
➲ http://www.kernel.org/pub/linux/kernel/people/rml/preempt-stats