© Kiskapu Kft. Minden jog fenntartva

Processzorhoz kötés

Bizonyos folyamatok egyetlen processzorhoz kötése egy új rendszerhívás segítségével.

A Linuxnak az a képessége, hogy a segítségével folyamatokat köthetünk egyetlen processzorhoz, már nagyon régóta várt tulajdonság volt. Ezt a képességet processzorhoz kötésnek nevezzük, ami azt jelenti: „ez a folyamat csak ezen a bizonyos processzoron futhat”, vagy pedig „ezek a folyamatok csak ezeken a processzorokon futhatnak, de a nullás processzoron nem”. Az ütemező ennek megfelelően betartja a kikötéseket, és a folyamatok csak a számukra kijelölt processzorokon fognak futni.
Más operációs rendszerek, mint például a Windows NT, már régóta biztosítanak processzorhoz kötéssel kapcsolatos rendszerhívásokat. Ennek következtében a Linuxban is egyre inkább kívánatossá vált, hogy ilyen képességekkel bővítsék. Végül a 2.5-ös rendszermagban megjelentek a folyamatok processzorhoz kötését meghatározó és lekérdező rendszerhívások. A cikkből kiderül, hogy miért szükséges, hogy ismerjük a processzorhoz kötéssel kapcsolatos képességeket, és később azt is megtudhatjuk, hogyan használhatjuk ki ezeket az új képességeket. Ha nem vagy programozó, vagy csak akad egy olyan programod, amit nem vagy képes módosítani, bemutatok neked egy segédeszközt, amellyel egy adott PID-del rendelkező folyamatod processzorhoz való kötését állíthatod be. Végül pedig megvizsgáljuk a processzorhoz kötéssel kapcsolatos rendszerhívások megvalósítását.

Gyenge és erős processzorhoz kötés

Kétféle processzorhoz kötés létezik: a gyenge és az erős.
A gyenge, amit természetes kötésnek is hívunk, az ütemezőnek azt a tulajdonságát hivatott jellemezni, hogy egy folyamat esetében az ütemező mindig arra törekszik, hogy az adott folyamat a lehető leghosszabb ideig ugyanazon a processzoron fusson. De ez pusztán próbálkozás, mivel ha bármi közbejön, a folyamat rögtön költözik is a másik processzorra. A 2.5-ben található O(1) ütemező kiváló természetes kötési képességekkel bír, amelynek a 2.4-es rendszermag pontosan az ellenkezője, ahol is a processzorhoz kötés meglehetősen gyenge volt. Ha ez a természetes kötés gyenge, létrejön a pingponghatás. Ez annyit tesz, hogy az ütemező a folyamat minden egyes meghívásakor más-más processzoron futtatja a folyamatot. Az *1. táblázatban* egy rossz processzorkötésű ütemezés látható; a *2. táblázat* azt mutatja, hogyan néz ki a jó processzorkötésű ütemezés. Ezekkel ellentétben az erős processzorkötés az, amit a processzorkötéssel kapcsolatos rendszerhívások biztosítanak. Az erős kötés kötelesség, amit az ütemezőnek kötelező érvénnyel be kell tartania. Ha például egy folyamat a nullás processzorhoz van kötve, akkor minden esetben csakis azon futhat.

1. táblázat **A pingponghatás**

	time 1	time 2	time 3	time 4
Process A	CPU 0	CPU 1	CPU 0	CPU 1

2. táblázat **A jó processzorhoz kötés**

	time 1	time 2	time 3	time 4
Process A	CPU 0	CPU 0	CPU 0	CPU 0

Miért van szükség a processzorhoz kötésre?

Mielőtt részletesebben megismerkednénk az új rendszerhívásokkal, előtte tisztázzuk, hogy egyáltalán miért van szükség rájuk. Elsődleges előnyük az, hogy a gyorstárazás teljesítménye növelhető. Mint mondtam, az O(1) ütemező erősen törekszik arra, hogy a folyamatok egy processzoron maradjanak, és lehetőség szerint ott is tartja őket. De bizonyos teljesítményigényes helyzetekben – mint amilyen mondjuk egy nagy adatbázis-kiszolgáló futtatása, vagy egy temérdek szállal dolgozó Java-kiszolgáló – nagyon fontos lehet, hogy ezt a processzorhoz való kötést a legszigorúbban kezeljük. A több processzoros számítógépek keményen megdolgoznak azért, hogy a processzorok gyorstára érvényben maradjon. Az adatok egyszerre csak az egyik processzor gyorstárában találhatók meg, máskülönben a gyorstár elveszíti összehangoltságát, és ez bizonytalansághoz vezet azt illetően, hogy melyik processzor gyorstára tartalmazza a rendszermemóriának leginkább megfelelő másolatot. Ennek következtében, ha az egyik processzor egy sornyi adatot helyez el a gyorstárában, akkor az összes többi processzornak a helyi gyorstárában érvénytelenítenie kell azt a bizonyos sort. Ez az érvénytelenítés meglehetősen kellemetlen és költséges. De az igazi gond akkor jelentkezik, ha egy folyamat ugrál a processzorok között, ami érvénytelenítések sorozatával jár, és a szükséges adat sosem található meg a gyorstárban. Így a gyorstár hibázási gyakorisága nagyon nagyra nő. A processzorhoz kötés ettől véd meg, és ezáltal növeli a teljesítményt.
A processzorhoz kötés másik nagy előnye első pillantásra a fentiek egyenes következményének tűnik. Ha több szál fér hozzá egyszerre ugyanahhoz az adathoz, akkor előnyös lehet, ha a szálak egy processzoron futnak. Így biztosított, hogy a szálak nem érvénytelenítik a különböző processzoron lévő adatokat, és nem okoznak hibákat a gyorstárban. Ez ugyanakkor SMP-rendszerek esetén csökkenti a többszálúságból következő teljesítménynövekedést. A gyorstárazásból nyert teljesítménynövekedés talán megéri azt a veszteséget, amit a szálak egymás utáni végrehajtása következtében el kell szenvednünk. A harmadik és egyben utolsó előny valós idejű, illetve időérzékeny alkalmazások esetében lehet számottevő. Ebben a megközelítésben minden egyes rendszerfolyamat a rendszerben bizonyos processzorokhoz van kötve. Egy bizonyos alkalmazás pedig a maradék processzorokhoz. Általános esetben egy kétprocesszoros rendszerben ez a bizonyos alkalmazás egy processzorhoz van kötve, míg minden más folyamat a másik processzorhoz. Így biztosított, hogy erre a bizonyos alkalmazásra összpontosul az adott processzor összes figyelme.

Az új rendszerhívások elérése

A rendszerhívások meglehetősen újak, ezért egyelőre nem minden rendszeren érhetők el. Legalább 2.5.8-pre3-as változatszámú rendszermag és 2.3.1-es glibc szükséges (bár a 2.3.0-s glibc támogatja a rendszerhívásokat, egy hibát is magában rejt). 2.4-es rendszereken az új rendszerhívások még nem érhetők el, de egy folt már rendelkezésre áll, ami a ➲ http://www.kernel.org/pub/linux/kernel/people/rml/cpu-affinitycímről tölthető le, illetve megtalálható az 51. CD Magazin/Processzor könytárában.
A frissebb terjesztések közül több is magában foglalja ezeket a szolgáltatásokat. A Red Hat 9-ben például már a rendszermag és a glibc is tartalmazza a rendszerhívásokhoz szükséges támogatást. A valós idejű megoldások közül például a MontaVista Linux támogatja az új felületet.

Kötődési maszkok

A legtöbb rendszeren – beleértve a Linuxot is – a processzorhoz kötést bitmaszkok segítségével lehet beállítani. A bitmaszk *n* számú bitből áll, ahol az egyes bitek be-, illetve kikapcsolt állapotától függnek bizonyos szolgáltatások. Például a processzorhoz kötést (32 bites számítógépeken) egy 32 bites bitmaszk határozza meg. Az egyes bitek azt jelölik, hogy az adott folyamat kötve van-e egy bizonyos processzorhoz vagy processzorokhoz. A biteket jobbról balra kell számolni, a 0-s bittől a 31-es bitig, és ennek megfelelően a nullás processzortól a 31. processzorig. Például:

```
11111111111111111111111111111111 = 4 294 967 295
```

Ez az alapértelmezett processzorhoz kötési maszk minden folyamat esetében. Mivel minden bit be van kapcsolva, a folyamat bármelyik processzoron futhat.
Ugyanakkor, ha a bitmaszk

```
00000000000000000000000000000001 = 1
```

akkor sokkal szigorúbb megkötések érvényesek. Minek utána csak a 0. bit van bekapcsolva, a folyamat csak a nullás processzoron futhat, vagyis a kötési maszk a folyamatot a nullás processzorhoz köti. Remélem, érthető.
Mit jelent a következő két maszk tízes számrendszerben értelmezve? Mi történik, ha ezt a két maszkot egy folyamat kötési maszkjaként használjuk fel?

```
10000000000000000000000000000000
00000000000000000000000000000011
```

Az első maszk 2 147 483 648-cal egyezik meg, mivel a 31. bit van bekapcsolva, és a folyamatot a 31. processzorhoz köti.
A második bitmaszk értéke 3, és a folyamatot a nullás és az első processzorhoz köti.
A Linux processzorkötési felülete a fentiekhez hasonló bitmaszkot alkalmaz. A C nyelv azonban sajnos nem támogatja a bináris állandókat, így mindig a decimális vagy hexadecimális, azaz tízes vagy tizenhatos számrendszerbeli állandókat kell használni. A fordító talán figyelmeztetni fog, ha olyan nagy állandót próbálsz használni, amelyik a 31. processzorhoz köti a folyamatot, de ne aggódj, működni fog így is.

Az új rendszerhívások használata

Az új rendszermag és glibc esetén az új rendszerhívások használata meglehetősen egyszerű:

```
#include <sched.h>

long
sched_setaffinity(pid_t pid, unsigned int len,
                  unsigned long *user_mask_ptr);

long
sched_getaffinity(pid_t pid, unsigned int len,
                  unsigned long *user_mask_ptr);
```

Az első rendszerhívást arra használjuk, hogy egy folyamat kötéseit beállítsuk, míg a második a már beállított kötéseket kérdezi le.
Mindegyik rendszerhívás estében a `PID` kapcsoló annak a folyamatnak a PID-jét jelenti, amelynek a tulajdonságait le akarod kérdezni, vagy változtatni akarsz rajtuk. Ha a PID nulla, akkor a hívó folyamatra vonatkoznak a beállítások.
A második kapcsoló a processzorkötési maszkban található bitek számát jelöli, amely jelenleg 4 bájt (32 bit). Ez a kapcsoló azért szükséges, mert ha a jövőben a rendszermagban változni fog az erre a célra fenntartott bitek száma, akkor a már elkészült programok az új környezetben is helyesen fognak működni. Nem lenne jó, ha meglévő programjaink egyszer csak működésképtelenné lennének. A harmadik kapcsoló egy hivatkozást tartalmaz, ami magára a bitmaszkra mutat.
Nézzük csak meg, hogyan kérdezhetjük le egy folyamat processzorhoz kötési bitmaszkját:

```
unsigned long mask;
unsigned int len = sizeof(mask);

if (sched_getaffinity(0, len, &mask) < 0) {
    perror("sched_getaffinity");
    return -1;
    }

printf("a processzorkötési maszkom:
       ➥%08lx\n", mask);
```

Afféle kényelmi szolgáltatásként a visszaadott bitmaszk `AND`-olva van a rendszerben található működőképes processzorok bitmaszkjával, így biztosan csak a működőképes processzorokhoz tartozó biteket látjuk bekapcsolva. Például egy egyprocesszoros rendszer a fenti hívásra mindig 1-gyel tér vissza (a 0. bit kivételével minden bit nulla).
A maszk beállítása éppen ilyen egyszerű:

```
unsigned long mask = 7;
➥/* processors 0, 1, and 2 */
unsigned int len = sizeof(mask);

if (sched_setaffinity(0, len, &mask) < 0) {
    perror("sched_setaffinity");
}
```

A példában a folyamatot a rendszerben lévő első három processzorhoz kötjük. Ezt követően meghívhatod a `sched_getaffinity()` függvényt, hogy megbizonyosodj róla, a fenti példa valóban működött-e. Mivel tér vissza a `sched_getaffinity()`, ha valójában csak két processzorral rendelkezel? Mivel tér vissza, ha csak egy processzorral? A rendszerhívás sikertelen, ha a bitmaszkban megjelölt processzorok legalább egyike nem létezik. A nullás maszk

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

1. lista A bind forrása

```
/* bind - egyszerű parancssoros eszköz
 * az alkalmazások processzorhoz kötésének
 * beállítására.
 */

#define _GNU_SOURCE

#include <stdlib.h>
#include <stdio.h>
#include <sched.h>

int main(int argc, char *argv[])
{
    unsigned long new_mask;
    unsigned long cur_mask;
    unsigned int len = sizeof(new_mask);
    pid_t pid;

    if (argc != 3) {
      fprintf(stderr,
              "usage: %s [pid] [cpu_mask]\n",
              argv[0]);
      return -1;
    }

    pid = atol(argv[1]);
    sscanf(argv[2], "%08lx", &new_mask);

    if (sched_getaffinity(pid, len,
                          &cur_mask) < 0) {
      perror("sched_getaffinity");
      return -1;
    }

    printf("pid %d's old affinity: %08lx\n",
           pid, cur_mask);

    if (sched_setaffinity(pid, len,
        ➥&new_mask)) {
      perror("sched_setaffinity");
      return -1;
    }

    if (sched_getaffinity(pid, len,
                          &cur_mask) < 0) {
      perror("sched_getaffinity");
      return -1;
    }

    printf(" pid %d's new affinity:
             ➥%08lx\n",
           pid, cur_mask);

    return 0;
}
```

mindig sikertelen hívást eredményez. Csakúgy, ha a 7. processzorhoz szeretnél kötni egy folyamatot, és nincs hét processzorod, a rendszerhívás ugyancsak sikertelen lesz.

A rendszerben minden folyamat processzorkötési maszkja lekérdezhető, azonban a maszk csak azokban a folyamatokban változtatható meg, amelyeknek te vagy a tulajdonosa, illetve bármely folyamat bitmaszkját megváltoztathatod, ha rendszergazdai jogosultságokkal rendelkezel.

Egy eszközt akarok!

Ha nem vagy programozó, illetőleg nem vagy képes módosítani a forrást, akkor is megváltoztathatod a folyamatok kötési tulajdonságait. Az első listában egy egyszerű parancssoros eszköz forráskódját találod, amellyel bármely folyamat kötési maszkját egyszerűen beállíthatjuk, ha tudjuk a folyamat PID-jét. Mint említettem, ehhez neked kell lenned az adott folyamat tulajdonosának, vagy pedig rendszergazdai jogosultságokat kell birtokolnod.
Az eszköz használata egyszerű, ha megtanulod a tízes számrendszerbeli kifejezésmódot:

```
bind pid maszk
```

Tegyük fel, hogy egy kétprocesszoros rendszerünk van, amelyben az 1600-as PID-del futó Quake-folyamatot a második processzorhoz szeretnénk kötni. Egyszerűen csak a következőt kell begépelnünk:

```
bind 1600 2
```

Legyünk igazán ravaszok

Az előző példában a Quake-et a rendszerünkben található egyik processzorhoz kötöttük. Hogy igazán gyors képet kapjunk, a rendszerünkben minden más folyamatot a másik processzorhoz kell kötnünk. Ezt megteheted kézzel is, vagy ha írsz egy trükkös parancsfájlt, de egyik sem igazán hatékony. Ehelyett használd ki, hogy a processzorhoz kötés a `fork()` során öröklődik, vagyis egy folyamat minden gyermeke ugyanahhoz a processzorhoz fog kötődni, mint a szülője.
Ennek a legtisztább módja, ha magát az `init`-et dolgozzuk meg egy kicsit, olyanformán, hogy a kívánt processzorkötési maszkot a rendszermag parancssorán keresztül adjuk át. Célunkat azonban egyszerűbben is elérhetjük, anélkül, hogy módosítanunk kellene vagy újra kellene fordítanunk az `init`-et. Helyette egyszerűen megszerkeszthetjük a rendszerindító parancsfájlt. A legtöbb rendszeren ez a parancsfájl a */etc/rc.d/rc.sysinit* vagy a */etc/rc.sysinit* fájlok egyike, amit az `init` először futtat le. Helyezzük el a `bind` példaprogramot a */bin* könyvtárban, és az *rc.sysinit* elejéhez adjuk hozzá a következő két sort:

```
/bin/bind 1 1
/bin/bind $$ 1
```

Ezek a sorok az `init`-et (aminek a PID-je 1) és az éppen futó folyamatot a nullás processzorhoz kötik. Minden jövőbeli folyamatnak ezek a folyamatok lesznek a szülői, így örökölni fogják a kötési maszkokat. Ezt követően a saját folyamatodat (legyen ez egy valós idejű nukleáris irányító rendszer vagy a Quake) a másik processzorhoz kötheted. Ezt követően minden folyamat a nullás processzoron fog futni, kivéve a saját folyamatunkat és annak gyermekeit, amelyek az egyes processzoron futnak. Ez azt jelenti, hogy így a teljes processzor csak a mi saját folyamatunkat fogja kiszolgálni.

Rendszermagbeli processzorkötés megvalósítása

Jóval azelőtt, hogy Linus felvette volna a processzorhoz kötési rendszerhívásokat, a rendszermag már támogatta és figyelembe vette a processzorhoz kötési maszkokat. Nem létezett azonban semmilyen felület, amin keresztül ezeket a maszkokat a felhasználó módosíthatta volna. Minden folyamathoz tartozó maszk a `task_struct` szerkezetben tárolódik `unsigned long`-ként, `cpus_allowed` néven. A `task_struct` szerkezetet a folyamat leírójának nevezzük. Ez tárol minden adatot az egyes folyamatokkal kapcsolatban. A processzorkötési felület egyszerűen csak kiolvassa és módosítja a `cpus_allowed` változót.

Bármikor, ha a rendszermag egy folyamatot át akar költöztetni egy másik processzorra, először ellenőrzi, hogy az adott folyamat futhat-e azon a másik processzoron. Ha a másik processzorhoz tartozó bit nincs bekapcsolva, akkor a processzor a folyamatot nem helyezi át. Ezen túlmenően, ha a processzorhoz kötési maszk megváltozik, és a folyamat éppen egy számára nem engedélyezett processzoron fut, akkor a folyamat soron kívül áthelyeződik egy számára engedélyezett processzorra. Ez biztosítja, hogy egy folyamat csak a számára engedélyezett processzoron kezdheti meg a futását, illetve csak olyan processzorra költözhet át, amelyen nincs tiltva a futása. Természetesen, ha csak egy processzorhoz van kötve a folyamat, akkor nem költözik sehova.

Összegzés

A 2.5-ösben megjelent processzorhoz kötési felület – és egyéb helyekre átültetett változatai – egyszerű, de mégis hasznos eszközt kínálnak arra, hogy meghatározhassuk, hogy az egyes folyamatok mely processzoron fussanak. A többprocesszoros géppel rendelkező felhasználók biztosan örülni fognak, hogy újabb lehetőség teremtődött arra, hogy rendszerüket még hatékonyabbá varázsolhassák, vagy arra, hogy a legfontosabb valós idejű alkalmazások mindig kapjanak processzoridőt. De az egyprocesszoros számítógéppel rendelkező felhasználóknak sem kell magukat mellőzve érezniük, ők is használhatják az új rendszerhívásokat, de ezáltal sem tesznek szert túl sok haszonra.

Linux Journal 2003. július, 111. szám

Robert Love
A Florida Egyetemen folytat matematikai és számítástudományi tanulmányokat, egyúttal a MontaVista Software-nél dolgozik rendszermag-fejlesztőként. Szeret fényképezni.

KAPCSOLÓDÓ CÍMEK

A Schedutils ütemezővel kapcsolatos eszköztár: ➲ http://tech9.net/rml/schedutils. A csomagban megtalálható a `taskset` nevű program, amellyel hatékonyan állítgathatjuk, a folyamatok processzorhoz kötését.

Processzorhoz kötési foltok a rendszermaghoz és a glibc-hez: ➲ http://www.kernel.org/pub/linux/kernel/people/rml/cpu-affinity

© Kiskapu Kft. Minden jog fenntartva