A 2.6-os rendszermag új munkasor kezelése

A rendszerteljesítmény egyik központi kérdése a megszakítási lappangási idő. A munkasorok használatával egy meghajtóprogramban kikerülhetjük a megszakításoktól védett időigényes kódrészletek használatát.

A legtöbb Unix-rendszerben – így a Linuxban is – az eszközmeghajtók általában két részre (félre) bontják a meghajtók feldolgozását. Az első részben (a felső félben) található az ismerős megszakításkezelő, amelyet a rendszermag az alkatrésztől kapott megszakítási jel észlelésekor hív meg. Sajnos a megszakítás valóban méltó a nevére: amikor az alkatrész kiadja a megszakítást, az bármilyen futó kódot megszakít. Tehát a megszakításkezelők (a felső felek) aszinkron módon futnak, figyelembe véve a jelenleg futó kódot. Minthogy a megszakításkezelők megszakítják a futó kódot (legyen az akár más rendszermagkód, akár felhasználói folyamat vagy másik megszakításkezelő), nagyon fontos, hogy a lehető leggyorsabban fussanak le.

Még nagyobb baj, hogy néhány megszakításkezelő (Linux alatt gyorsak a megszakításkezelők) a futás során kikapcsolják a megszakításokat a helyi processzoron. Ezt azért teszik, hogy a megszakításkezelő megszakítás nélkül, a lehető leggyorsabban fusson le. Továbbá az összes megszakításkezelő az összes processzoron kikapcsolja a saját megszakítási vonalát. Ez biztosítja, hogy azonos megszakítási vonalnak két megszakításkezelője nem futhat egyszerre. Egyúttal megakadályozza, hogy a meghajtóprogram készítőjének kezelni kelljen a rekurzív megszakításokat, ami bonyolítaná a programozást. Ha azonban a megszakítások le vannak tiltva, más megszakításkezelők sem tudnak futni. A megszakításlappangás (interrupt latency) – vagyis hogy mennyi időbe telik a rendszermagnak, amíg egy alkatrész megszakítási kérelmére válaszol – a teljesítmény egyik kulcskérdése. Ismétlem, a megszakításkezelők sebessége súlyos kérdés. Kisméretű gyors megszakításkezelők írásának az elősegítésére a második részt, avagy az alsó felet használjuk. Ennek a résznek a feladata, hogy a lehető legnagyobb mennyiségű, későbbre halasztható munkát átvegye a felső résztől. Az alsó rész bekapcsolt megszakítások mellett fut, ennek megfelelően az alsó fél futtatása nem akadályozza a többi megszakításkezelő futását – így nincs is hátrányos hatással a megszakítási lappangásra.

Csaknem minden eszközmeghajtó alkalmaz ilyen vagy olyan alsó felet. Az eszközmeghajtó a felső fél (a megszakításkezelő) segítségével válaszol az alkatrésznek, illetve itt végzi el a fontos időérzékeny műveleteket, például az eszközregiszterek alaphelyzetbe történő állítását vagy az eszközből a memóriába történő adatmásolást. A megszakításkezelő ezután megjelöli az alsó felet, utasítva a rendszermagot, hogy a lehető leghamarabb futassa le a szükséges kódot, majd kilép.

A legtöbb esetben a valódi munka csak ekkor kezdődik az alsó félben. Egy későbbi időpontban – gyakorta a megszakításkezelő visszatérése után azonnal – a rendszermag az alsó felet is végrehajtja. Az alsó fél lefut és elvégzi azokat a feladatokat, amelyeket a megszakításkezelő hátrahagyott. Az alsó és felső fél közötti munkamegosztás teljes egészében az eszközmeghajtó-készítő döntésén múlik. Az eszközmeghajtó-készítők a lehető legtöbb munkát általában megpróbálják az alsó részre hárítani.

A Linux – zavaró módon – elég sokfajta módszert ajánl fel az alsó rész megvalósítására. Jelenleg a 2.6-os rendszermag `softirq`-k, `tasklet`-ek és munkasorok (work queues) típusú alsórész-megoldásokkal bír. A korábbi rendszerekben másfajta alsó felek voltak elérhetők: például a BH-k és a feladatsorok (task queues interface). Ez a cikk kizárólag az új munkasorfelülettel (work queue) foglalkozik, amely a 2.5-ös fejlesztői sorozatban mutatkozott be a feladatsorok betegeskedő `keventd` részének a kiváltása végett.

Bevezetés a munkasorok világába

A munkasorok két dolog miatt is érdekesek: először is az összes alsófél-megoldás közül ezeket a legegyszerűbb használni. Másodszor ez az egyetlen olyan alsófél-megoldás, ami folyamatkörnyezetben (process context) is képes futni, így a munkasorok jelentik az egyetlen lehetőséget, amivel az eszközmeghajtó-készítő akkor élhet, ha az alsó résznek aludnia kell. Továbbá a munkasorok képessége vadiúj, és szerintem ami új, az király.

Nézzük meg, mivel is jár az, hogy a munkasorok a folyamatkörnyezetben futnak! Ez jelentős különbség más alsófél-megoldásokhoz képest, amelyek – ettől eltérő módon – megszakítási környezetben futnak. A megszakítási környezetben futó kód nem alhat vagy blokkolhat, minthogy a megszakítási térnek semmilyen újraütemezhető háttérfolyamata nincsen. Ezért, mivel a megszakításkezelők nincsenek folyamathoz rendelve, az ütemezőnek nincs mit alvó állapotba helyeznie, és ami talán még fontosabb: nincs

mit felébresztenie sem. Következésképpen megszakításkörnyezetben nem lehet elvégezni olyan műveleteket, amelyek a futó környezetet a rendszermagban alvó állapotba helyeznék. Ilyen művelet a jelző lekapcsolása (downing a semaphore), a felhasználói tér memóriájába vagy onnan történő másolás, vagy a nem önműködően történő memóriafoglalás. Minthogy a munkasorok folyamatkörnyezetben futnak (mint azt látni fogjuk, a rendszermagszálak hajtják őket végre), teljes mértékben képesek az alvásra. A rendszermag lényegében pontosan úgy ütemezi az alsó felek futását a munkasorokban, mint a rendszer bármely más folyamatát. Akárcsak a többi rendszermagszál (kernel threads), a munkasorok is alhatnak, meghívhatják az ütemezőt és így tovább.

Normál esetben a munkasorokat az alapértelmezett rendszermagszálak kezelik. Processzoronként egy ilyen alapértelmezett szál fut. Ezeket `events/n` alakú névvel jelöljük, amelyben ***n*** annak a processzornak a száma, amelyhez a szál kötődik. Például az egyprocesszoros gépeknek csak egy `events/0` rendszermagszála lenne, míg a kétprocesszoros gépeknek már egy `events/1` nevű szála is lenne. Munkasorainkat azonban akár saját rendszermagszálon is futtathatjuk. Amikor az alsó felünk elindul, az alapértelmezett szál helyett a saját egyedi szálunk ébred fel és kezd el dolgozni. Egyedi munkasorszálat (unique work queue thread) készíteni csak néhány teljesítményfüggő esetben érdemes. Általában az alapértelmezett szálat használó alsó feleket érdemes előnyben részesíteni. Ennek ellenére később azt is meg fogjuk nézni, hogy miként lehet munkasorszálat készíteni.

A munkasorszálak alsó részünket munkasorkezelőnek nevezett különleges függvényként hajtják végre – a munkasorkezelőben készíthetjük el az alsó részünket. A munkasorfelület használata nagyon könnyű; az egyetlen nehéz rész magának az alsó résznek (azaz a munkasorkezelőnek) az elkészítése.

A munkasor illesztőfelülete

A munkasorok használatához először is létre kell hoznunk egy munkasorszerkezetet. A munkasorszerkezetet a ***linux/workqueue.h*** fájlban megadott `struct work_struct` határozza meg – szerencsére a munkasorszerkezetek előállítását két makró könnyíti meg. Amennyiben munkasorszerkezetünket statikusan szeretnénk meghatározni (azaz úgynevezett globális változóként), a következő kifejezéssel közvetlenül megadhatjuk:

```
DECLARE_WORK(name, function, data)
```

Ez a makró létrehozza a `struct work_struct` szerkezetet, majd a megadott munkasorkezelő és függvény felhasználásával alaphelyzetbe állítja. Munkasorkezelőnknek a következő prototípusnak kell megfelelnie:

```
void my_workqueue_handler(void *arg)
```

Az `arg` értéket a rendszermag a munkasorkezelő minden egyes meghívásakor átadja – ezt a `DECLARE_WORKQUEUE()` makróban található `data` érték határozza meg. Az eszközmeghajtók az érték használatával több munkasort is kezelni tudnak egyetlen munkasorkezelő segítségével. A `data` értéket a munkasorok megkülönböztetésére is felhasználhatjuk. Amennyiben munkasorszerkezetünket közvetlen módszer helyett inkább dinamikusan szeretnénk létrehozni, arra is lehetőségünk nyílik. Ha a munkasorszerkezetre csak közvetett hivatkozásunk van (például azért, mert a `kmalloc()` függvénnyel foglaltuk le), a következő módon állíthatjuk alaphelyzetbe:

```
INIT_WORK(p, function, data)
```

Ebben az esetben a `p` a `work_struct` szerkezetre mutat majd, a `function` a munkasorkezelőnk, a `data` pedig az az érték, amit a rendszermag a meghíváskor át fog adni. A munkasorszerkezet létrehozását normál esetben csak egyszer végezzük el, például a meghajtónk alaphelyzetbe állító eljárásában. A rendszermag a munkafolyam-szerkezetet használja a rendszeren futó különféle munkasorok nyomon követésére. Érdemes figyelemmel kísérnünk a szerkezetet, mert később még szükségünk lesz rá.

A saját munkasorkezelőnk

Munkasorkezelőnk lényegében bármit csinálhat, végtére is a saját alsó részünkről van szó! Az egyetlen megszorítás, hogy kielégítse a megfelelő prototípust. Minthogy munkasorkezelőnk folyamatkörnyezetben fut, szükség esetén aludni is tud.

Megvan a munkasor-adatszerkezetünk és a munkasorkezelőnk, de hogyan tudjuk futtatni? Ha azt szeretnénk, hogy az adott munkasorkezelő a legközelebbi, a rendszermag számára is kényelmes helyen fusson le, hívjuk meg a következő függvényt és adjuk át neki a munkasorszerkezetünk címét:

```
int schedule_work(struct work_struct *work)
```

Ha a besorolás sikeres volt, a függvény nullától eltérő értéket ad vissza, hiba esetén azonban nullát. A függvény folyamat- vagy megszakításkörnyezetben egyaránt hívható. Néha előfordul, hogy a munkasort nem azonnal akarjuk beütemezni, csak egy bizonyos idő eltelte után; ilyen esetekben a következő utasítást használjuk:

```
int schedule_delayed_work(struct work_struct *work,
                          unsigned long delay)
```

Ilyenkor az adott munkasorszerkezethez rendelt munkasorkezelő a megadott ideig még biztosan nem fog lefutni. Például ha van egy `my_work` nevű szerkezetünk és öt másodpercre szeretnénk időzíteni, a következő formát használjuk:

```
schedule_delayed_work(&my_work, 5*HZ)
```

Normál esetben munkasorkezelőnket a megszakításkezelőnkből időzítjük, de semmi sem akadályoz meg bennünket, hogy ezt most tetszés szerinti helyről tegyük meg. Normál esetben az alsó fél és a megszakításkezelő csapatként együtt dolgozik, közösen oldják meg az eszközzel kapcsolatos feladatokat. A megszakításkezelő mint a

Munkasor-függvényreferencia

Statikus munkasorszerkezet-készítés:

```
DECLARE_WORK(név, függvény, adat)
```

Dinamikus munkasorszerkezet:

```
INIT_WORK(p, függvény, adat)
```

Új munkaszál készítése:

```
struct workqueue_struct
*create_workqueue(const char *név)
```

Munkaszál törlése:

```
void
destroy_workqueue(struct workqueue_struct *wq)
```

Munka adott munkaszálhoz rendelése:

```
int
queue_work(struct workqueue_struct *wq,
           struct work_struct *work)
```

Munka ütemezése adott idő eltelte után adott munkaszálhoz:

```
int
queue_delayed_work(struct workqueue_struct *wq,
                   struct work_struct *work,
                   unsigned long delay)
```

Függőben lévő munkák kivárása az adott munkaszálon:

```
void
flush_workqueue(struct workqueue_struct *wq)
```

Munka ütemezése az alapértelmezett munkaszálon:

```
int
schedule_work(struct work_struct *work)
```

Munka ütemezése adott idő eltelte után az alapértelmezett munkaszálon:

```
int
schedule_delayed_work(struct work_struct *work,
                      unsigned long delay)
```

Függőben lévő munkák kivárása az alapértelmezett szálon:

```
void
flush_scheduled_work(void)
```

Az adott késleltetett munka leállítása:

```
int
cancel_delayed_work(struct work_struct *work)
```

megoldás felső fele általában előkészíti a megmaradt munkát, majd beütemezi az alsó felet. Az is elképzelhető, hogy a munkasorokat az alsófél-feldolgozáson kívül másra szeretnénk használni.

Munkasorkezelés

Amikor besorolunk egy munkát, az a munkaszál következő ébredésekor fut majd le. Néha előfordul, hogy folytatás előtt biztosítanunk kell a munkasor befejeződését a rendszermagban. Ez különösen a modulok esetében fontos, amelyeknél a modul kivétele előtt az összes alsó félnek be kell fejeződnie. Az igény kielégítésére a rendszermagban egy függvényt találunk, amely megvárja, amíg a függőben lévő munkaszálon az összes munka lefut:

```
void flush_scheduled_work(void)
```

Minthogy ez a függvény a munkaszálon található összes függőben lévő munkafolyamatot kivárja, a dolog viszonylag hosszú időt vehet igénybe. Miközben a függvény a munkaszálak befejezésére vár, a hívás alszik. Következésképpen ezt a függvényt kizárólag folyamatkörnyezetben hívhatjuk meg. Ne használjuk a függvényt, ha ütemezett munkánk befejezését nem feltétlenül kell biztosítanunk. A függvény nem üríti ki a függőben lévő időzített munkákat. Amennyiben időzítéssel ütemezzük a munkát és az időzítés még nem járt le, a munkasor ürítése előtt le kell állítanunk az időzítést:

```
int cancel_delayed_work(struct work_struct *work)
```

Ez a függvény az adott munkasorszerkezethez tartozó időzítést állítja le – más munkasorokra nincs hatással. A `cancel_delayed_work()` függvényt kizárólag folyamatkörnyezetben hívhatjuk meg, mivel alhat. Ha valamilyen munkát leállított, nullától eltérő értéket ad vissza, egyéb esetben a visszaadott érték nulla.

Új munkaszál létrehozása

Ritkán az alapértelmezett munkasorszál nem elég. Szerencsére a munkasor-csatolófelület lehetővé teszi, hogy saját munkaszálat készítsünk és ezt használjuk alsó részünk futtatására. Új munkasorszál létrehozására a következő függvényt használjuk:

```
struct workqueue_struct *
create_workqueue(const char *name)
```

A rendszer elindítása előtt (system initialization) a rendszermag például a következőképpen készíti el az alapértelmezett szálakat:

```
keventd_wq = create_workqueue("events");
```

Ez a függvény készíti el a processzoronkénti összes munkaszálat. Visszaadott értéke a `struct workqueue_struct` szerkezetre hivatkozó mutató, ami a munkasor más munkasoroktól (például az alapértelmezettektől) való megkülönböztetésére szolgál. A munkaszál létrehozása után hasonlóképpen rendelhetünk hozzá munkákat, mint az

alapértelmezett munkaszálakhoz:

```
int queue_work(struct workqueue_struct *wq,
               struct work_struct *work)
```

Itt a `wq` a `create_workqueue()` függvénnyel készített, a munkasort azonosító mutató, a `work` pedig a munkasorszerkezetünkre hivatkozó mutató. Másik megoldásként a munkát időzítve is ütemezhetjük:

```
int
queue_delayed_work(struct workqueue_struct *wq,
                   struct work_struct *work,
                   unsigned long delay)
```

E függvény működése azonos a `queue_work()` függvényével, kivéve, hogy a besorolást a `delay` kapcsolóban megadott pillanatig elhalasztja. A két függvény megfelel a `schedule_work()` és `schedule_delayed_work()` függvényeknek, eltekintve attól, hogy az adott munkát az alapértelmezett helyett a megadott munkasorba helyezik el. Mindkét függvény siker esetén nullától eltérő értéket, hiba esetén pedig nullát ad vissza. Mindkét függvény folyamat- vagy megszakításkörnyezetből egyaránt hívható.
Végül egy adott munkasort a következő utasítással üríthetünk ki:

```
void flush_workqueue(struct workqueue_struct *wq)
```

A függvény addig vár, amíg `wq` munkaszálon az összes ütemezett munka be nem fejeződik, és csak azután tér vissza.

Összegzés

A munkasor-csatolófelület a 2.5.41-es változattól része a rendszermagnak. Akkoriban igen sok meghajtó és alrendszer kezdte el munkahalasztási módszerként használni. Felmerül a kérdés: biztos, hogy ez az eszményi alsó rész számunkra? Amennyiben az alsó részünket folyamatkörnyezetben kell futtatnunk, valóban a munkasor a megfelelő választás. Továbbá ha esetleg rendszermagszálakat szeretnénk létrehozni, valószínűleg úgyszintén a munkasorok alkalmazása lesz a legjobb megoldás. De mit tegyünk, ha nincs szükségünk olyan alsó részre, ami aludni is tud? Ez esetben valószínűleg a `tasklet`-ek használata a jobb választás. Ezeket is könnyű alkalmazni, de nem használnak rendszermagszálakat. Minthogy nem folyamatkörnyezetben futnak, végrehajtásuk során környezetváltási pluszmunka sincs – így számunkra is kevesebb többletmunkát okoznak.

Linux Journal 2003. november, 115. 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 punkzenét hallgat.