Memóriafoglalás a rendszermagban

Robert írása röviden bemutatja a rendszermag-memóriafoglalás lépéseit, valamint azt is megtudhatjuk, hogy milyen változások történtek a 2.6-os rendszermagban.

A rendszermagfejlesztők bánatára a rendszermagban egyáltalán nem olyan egyszerű memóriát foglalni, mint a felhasználói térben. A nehézségek több okra vezethetők vissza, többek közt:

- A rendszermag körülbelül 1 GB virtuális és fizikai memóriakorláttal bír.
- A rendszermemória nem lapozható.
- A rendszermag fizikailag általában folytonos memóriát igényel.
- A magnak gyakran pihenés nélkül kell memóriát foglalnia.
- A rendszermagban elkövetett hibákért sokkal magasabb árat fizetünk, mint bárhol másutt.

Bár a nagy memóriatömegek elérése nyilvánvalóan nem kifejezetten a rendszermagnak kijáró fényűzés, a nehézségek tisztázásával sokat tehetünk azért, hogy a folyamat viszonylag fájdalommentes legyen.

Általános célú foglaló (allocator)

A memóriafoglalás általános célú felülete a rendszermagban a `kmalloc()`:

```
#include <linux/slab.h>
void * kmalloc(size_t size, int flags);
```

Ismerősnek tűnhet – végtére is éppen olyan, mint a felhasználói térben megszokott `malloc()` – kivéve, hogy egy második, `flags` nevű kapcsolót is vár. Most egyelőre tekintsünk el tőle és lássuk, mit ismerünk! Először is a méret (size) itt is ugyanazt jelenti, mint a `malloc()`-változatban: a foglalás méretét adja meg bájtban. Sikeres visszatérés esetén a `kmalloc()` a memória `size` méretű területére ad vissza mutatót. A legfoglalt memória kiosztása bármilyen objektumtípus tárolásához és eléréséhez megfelelő. Akárcsak a `malloc()`, a `kmalloc()` is lehet sikertelen, ezért ellenőriznünk kell, hogy a visszatérési érték nem `NULL`-e. Lássunk egy példát!

```
struct falcon *p;

p = kmalloc(sizeof (struct falcon), GFP_KERNEL);
if (!p)
  /* a foglalás sikertelen - itt kezeljük le*/
```

Zászlók (Flags)

A `flags` mező a memóriafoglalás viselkedését befolyásolja. A jelzőbiteket három csoportra bonthatjuk: műveletmódosítók, zónamódosítók és típusok. A műveletmódosító mondja meg a rendszermagnak, hogy miképp foglalja le a memóriát. Például ezek mutatják meg, hogy a rendszermag pihenhet-e (vagyis a `kmalloc()` hívás blokkolhat-e) a foglalás teljesítése közben. Ezzel szemben a zónamódosítók arról tájékoztatják a rendszermagot, hogy honnan kell a kérést kielégíteni. Például bizonyos kérések teljesítésekor olyan memóriát kell felhasználni, amit bármelyik eszköz a közvetlen memóriaelérés (Direct Memory Access, DMA) szolgáltatáson keresztül közvetlenül el tud érni. Végül a típuszászlók mutatják a foglalás típusát; ezek a megfelelő zóna- és

1. táblázat **A műveletmódosítók**

Zászló	Leírás
__GFP_COLD	A rendszermag a hideg gyorstárlapokat (cache cold pages) használja.
__GFP_FS	A rendszermag fájlrendszer ki–bemenetet indíthat el.
__GFP_HIGH	A rendszermag hozzáférhet a vésztartalékokhoz (emergency pools).
__GFP_IO	A rendszermag lemez-B-K-műveleteket végezhet.
__GFP_NOFAIL	A rendszermag megismételheti a foglalást.
__GFP_NORETRY	A rendszermag nem tér vissza, ha a foglalás nem sikerült.
__GFP_NOWARN	A rendszermag sikertelenség esetén nem ír figyelmeztetéseket.
__GFP_REPEAT	A rendszermag megismétli a foglalást, ha nem jár sikerrel.
__GFP_WAIT	A rendszermag alhat.

2. táblázat **A zónamódosítók**

Zászló	Leírás
__GFP_DMA	Kizárólag DMA-k által elérhető memória foglalása.
No flag	Onnan foglalunk, ahol van.

műveletmódosítókat foglalják össze egyetlen szimbólumban. Általában több zóna és művelet megadása helyett egyszerűen csak egy típust adunk meg.
Az *1. táblázat* a műveletmódosítókat sorolja fel, míg a *2. táblázatban* a zónamódosítókat találjuk. Több eltérő jelet is használhatunk; a rendszermag memóriafoglalása nem egyértelmű. A rendszermagban a memóriafoglalás számos jellemzőjét befolyásolhatjuk. Akkor járunk el helyesen, ha kódunk a típusjeleket és nem az egyedi művelet- és zónamódosítókat használja. A leggyakoribb zászlók: a `GFP_ATOMIC` és a `GFP_KERNEL`; szinte valamennyi rendszermag-memóriafoglalás e két zászló közül használja valamelyiket.
A `GFP_ATOMIC` jel arra utasítja a memóriafoglalót, hogy soha ne blokkoljon. Ezt a zászlót olyan helyzetekben használjuk, amikor nem szabad „aludni" – azaz atomi jellegűnek kell maradni –, például a megszakításkezelőkben, az alsó részben (bottom half) és az olyan folyamat jellegű kódokban, amelyek zárakat tartalmaznak. Minthogy a rendszermag nem blokkolhatja a foglalást és megpróbálja felszabadítani a kérelem kielégítéséhez szükséges memóriát, a `GFP_ATOMIC` zászlót megadó kérelemnek kisebb esélye van a sikerre, mint azoknak, amelyek nem igénylik ezt a zászlót. Ugyanakkor megoldásunk az alvással nem egyeztethető össze, tehát ez az egyetlen lehetőség. A `GFP_ATOMIC` használata nagyon egyszerű:

```
struct wolf *p;

p = kmalloc(sizeof (struct wolf), GFP_ATOMIC);
if (!p)
    /* hiba */
```

Ennek megfelelően a `GFP_KERNEL` zászló a normál rendszermagfoglalást jelenti. Ezt a zászlót használjuk az olyan folyamatokban, amelyekben nincsenek zárak. Az ilyen zászlóval meghívott `kmalloc()` alhat; ezért ezt a zászlót csak akkor szabad használnunk, ha nem okoz galibát.

3. táblázat **A típusok**

Zászló	Leírás
GFP_ATOMIC	A foglalás nagy fontosságú és nem alszik. Ezt a jelet használjuk a megszakításkezelőkben, az alsó részben (bottom half) és egyéb helyzetekben, ahol az alvás nincs megengedve.
GFP_DMA	A DMA-k által elérhető memória foglalása. Azok az eszközmeghajtók, amelyek DMA-elérhetőségű memóriát alkalmaznak, ezt a zászlót használják.
GFP_KERNEL	Normál, blokkolható foglalás. Folyamatkörnyezetben ezt a zászlót használjuk, amennyiben az alvás biztonságos.
GFP_NOFS	Ez a fajta foglalás blokkolható és lemezműveletek kezdeményezését is megengedi, de a fájlrendszerműveleteket nem. Ezt a jelet használjuk a fájlrendszerkódokban, ahol másik fájlrendszerművelet indítása nem kívánatos.
GFP_NOIO	Ez a foglalás blokkolható, de nem kezdeményezhet blokkos ki- és bemenetet. Ezt a jelet használjuk a blokkos rétegben, amelyben nem szeretnénk további blokkos B-K-műveleteket indítani.
GFP_USER	Normál, blokkolható foglalás. Ezt a zászlót használjuk a felhasználói folyamatok memóriafoglalásához.

A rendszermag szükség esetén az alvás képességét a memória felszabadítására használja, ezért aztán az ilyen zászlót használó foglalásoknak nagyobb az esélyük a sikerre. Amennyiben például nem áll elegendő memória rendelkezésre, a rendszermag blokkolhatja a kódot és a lemezre menthet néhány nem működő lapot, csökkentheti a memóriagyorstár méretét, kiírhatja a kimeneti átmeneti tárat (buffer) és így tovább.
Néha, például amikor ISA eszközvezérlőt készítünk, biztosítanunk kell, hogy a lefoglalt memória képes együttműködni a kiszolgáló DMA-rendszerrel. Az ISA eszközök esetében ez a fizikai memória első 16 MB-nyi területét jelenti. Ha bizonyosak akarunk lenni benne, hogy a rendszermag ebből a területből foglal memóriát, a `GFP_DMA` zászlót kell használnunk. Ezt általában a `GFP_ATOMIC` vagy a `GFP_KERNEL` zászlóval együtt használjuk; a jeleket bináris `OR` művelettel kapcsolhatjuk össze. Például, ha a rendszermagot DMA-ra alkalmas memóriaterület foglalására akarjuk utasítani és engedélyezzük az alvást, az utasítás a következő módon fest:

```
char *buf;

/* DMA-ra alkalmas memóriát szeretnénk és szükség
   ➥esetén alhat*/
buf = kmalloc(BUF_LEN, GFP_DMA | GFP_KERNEL);
if (!buf)
    /* hiba */
```

A *3. táblázat* a zászlótípusokat sorolja fel, míg a *4. táblázatban* azt mutatjuk be, hogy az egyes zászlók milyen művelet- és zónamódosítóknak felelnek meg. A `<linux/gfp.h>` fejléc azonosítja a zászlókat.

Memória felszabadítása

Miután a `kmalloc()` segítségével lefoglalt memória használatát befejeztük, vissza kell adnunk a rendszermagnak a lefoglalt területet. Ezt a feladatot a `kfree()` végzi el, amely a felhasználói tér `free()` könyvtárhívásának felel meg.
A `kfree()` meghatározása így fest:

```
#include <linux/slab.h>

void kfree(const void *objp);
```

A `kfree()` használata azonos felhasználó térbeli megfelelőjével. Tegyük fel, hogy `p` a `kmalloc()` által lefoglalt

4. táblázat **A típusjelek összehasonlítása**

Zászló	Érték
GFP_ATOMIC	__GFP_HIGH
GFP_NOIO	__GFP_WAIT
GFP_NOFS	(__GFP_WAIT | __GFP_IO)
GFP_KERNEL	(__GFP_WAIT | __GFP_IO | __GFP_FS)
GFP_USER	(__GFP_WAIT | __GFP_IO | __GFP_FS)
GFP_DMA	__GFP_DMA

memóriaterület mutatója. Ebben az esetben a következő parancs szabadítaná fel a memóriablokkot:

```
kfree(p);
```

Akárcsak a felhasználói tér `free()` függvénye esetében, ha a `kfree()`-t egy már korábban felszabadított memóriaterülettel hívjuk meg vagy olyan mutatót adunk át, amely nem a `kmalloc()` függvénytől származik, hibázunk, ez pedig memóriahibákhoz vezethet. Fontos, hogy kiegyensúlyozottan hívjuk a foglalásokat és a felszabadításokat, ügyelve arra, hogy a `kfree()`-t mindig pontosan egyszer hívjuk meg a megfelelő mutatóhoz. A `kfree()` meghívása a `NULL` értékkel biztonságos, ugyanis az érték létezését ellenőrzi – mindazonáltal nem éppen értelmes ötlet.
Vessünk egy pillantást a teljes foglalásfelszabadítás ciklusra:

```
struct sausage *s;

s = kmalloc(sizeof (struct sausage), GFP_KERNEL);
if (!s)
    return -ENOMEM;
/* ... */

kfree(s);
```

Virtuális memória foglalása

A `kmalloc()` függvény fizikai és következésképpen virtuálisan folytonos memóriaterületet ad vissza. Ebben különbözik a felhasználói tér `malloc()`s függvényétől, amely szintén virtuális, de nem szükségszerűen folytonos fizikai memóriaterületet foglal le. A fizikailag folytonos területek két elsődleges előnnyel bírnak: először is sok alkatrész nem képes virtuális memóriát címezni. Ezért aztán, hogy egy memóriaterületet elérhessenek, a memóriablokknak fizikailag folytonos memóriaterületen kell léteznie. Másodszor a fizikailag folytonos memóriablokk egyetlen nagy lapterületet használ. Ez nagymértékben csökkenti a TLB (translation lookaside buffer) memóriacímző munkáját, hiszen mindössze egyetlen TLB-bejegyzésre van szükség.
A fizikailag folyamatos memóriának akad egy nagy hátránya: gyakran igen nehéz fizikailag folytonos memóriaterületet találni, s különösen igaz ez a nagyméretű foglalásokra. A csak virtuálisan folytonos memória sikeres foglalására jóval nagyobb esélyünk van. Amennyiben nincs szükségünk fizikailag folytonos memóriára, használjuk a `vmalloc()` függvényt:

```
#include <linux/vmalloc.h>

void * vmalloc(unsigned long size);
```

A `vmalloc()` által lefoglalt memóriát a `vfree()` segítségével adhatjuk vissza a rendszernek:

```
#include <linux/vmalloc.h>

void vfree(void *addr);
```

A `vfree()` használata ismét azonos a felhasználói tér `malloc()` és `free()` függvényének a használatával:

```
struct black_bear *p;

p = vmalloc(sizeof (struct black_bear));
if (!p)
    /* hiba */

/* ... */

vfree(p);
```

Ebben az esetben a `vmalloc()` alhat.
A rendszermagban nagyszámú foglalás használhat `vmalloc()` függvényt, mivel csak kevés foglalásnak kell egybefüggőnek látszania az eszközök számára. Ha olyan memóriát foglalunk le, amit csak programok érnek el, például felhasználói folyamatokkal kapcsolatos adatokat tárolunk, a memóriának nem szükséges fizikailag folytonosnak lennie. Ennek ellenére a rendszermagban kevés foglalás használja a `vmalloc()` függvényt. A legtöbb a `kmalloc()`-ot választja, mégha szükségtelen is, részben történelmi, részben teljesítménynövelési okokból kifolyólag. Minthogy a fizikailag folytonos lapok jóval kevesebb munkát adnak a TLB-nek, az ebből fakadó teljesítménynövekedést gyakran sokra tartják. Ennek ellenére, ha megabájtok tucatjait akarjuk lefoglalni a rendszermagban, a `vmalloc()` a legjobb választás.

Apró, állandó méretű verem

A felhasználói folyamatokkal ellentétben a rendszermagban futó kódnak nincsen nagy, dinamikusan növekvő verme – ehelyett a rendszermag minden folyamata saját apró, állandó méretű veremmel rendelkezik. A verem pontos mérete kiépítésfüggő; a legtöbb architektúra két lapot foglal le veremnek, a verem tehát a 32 bites gépeken 8 KB.
Az apró verem miatt kerülendők a nagy, önműködő, veremben tárolt foglalások. Ha megnézzük, soha sem találunk ilyesmit a rendszermagkódban:

```
#define BUF_LEN        2048

void rabbit_function(void)
{
    char buf[BUF_LEN];
    /* ...  */
}
```

A helyes változat a következő:

```
#define BUF_LEN  2048

void rabbit_function(void)
{
    char *buf;

    buf = kmalloc(BUF_LEN, GFP_KERNEL);
    if (!buf)
        /* error! */

/* ... */
}
```

Felhasználói térben ritkán látunk hasonlót, hiszen nincs túl sok értelme dinamikusan foglalni memóriát, ha a kód írásakor már ismerjük a lefoglalandó méretet. A rendszermagban azonban mindig dinamikus foglalást alkalmazunk, amennyiben a foglalás néhány bájtnál nagyobb. Így elkerülhetjük a veremtúlcsordulást, ami egyébként számos gondot okozhatna.

Összegzés

Fellebbentettük a fátylat a rendszermag memóriakezeléséről, és láthattuk, hogy nem sokkal nehezebb, mint a felhasználói térben található megfelelője. Néhány ökölszabály, amit érdemes betartanunk:

- Döntsük el, hogy alhatunk-e (azaz a `kmalloc()` blokkolható-e) vagy sem. Ha egy megszakításkezelő közepén vagyunk és az alsó részben (bottom half) egy zárat tartunk, akkor ezt nem tehetjük meg. Amennyiben folyamatszinten vagyunk és nem tartunk zárakat, valószínűleg megtehetjük.
- Ha alhatunk, a `GFP_KERNEL`-t adjuk meg.
- Ha nem, válasszuk a `GFP_ATOMIC` zászlót.
- Amennyiben DMA által használható memóriára van szükségünk (például ISA vagy hibás PCI eszközhöz), adjuk meg a `GFP_DMA` zászlót.
- Mindig ellenőrizzük és kezeljük a `kmalloc()` által esetleg visszaadott `NULL` értéket.
- Ne folyassuk el a memóriát; ne felejtsük el a `kfree()` függvényt valahol meghívni.
- Biztosítsuk, hogy a `kfree()`-t nem hívjuk meg többször, és sehol se próbáljuk meg a már felszabadított memóriarészeket elérni.

Linux Journal 2003. december, 116. 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.

TOVÁBBI ÉRDEKESSÉGEK

További adatokat a rendszermagfa fájljaiban találunk:

- include/linux/gfp.h: a foglalási zászlók gyűjteménye.
- include/linux/slab.h: a `kmalloc()` és társainak meghatározásai.
- mm/page_alloc.c: lapfoglalási függvények.
- mm/slab.c: a `kmalloc()` és társainak megvalósítása.