© Kiskapu Kft. Minden jog fenntartva

A DMA használata

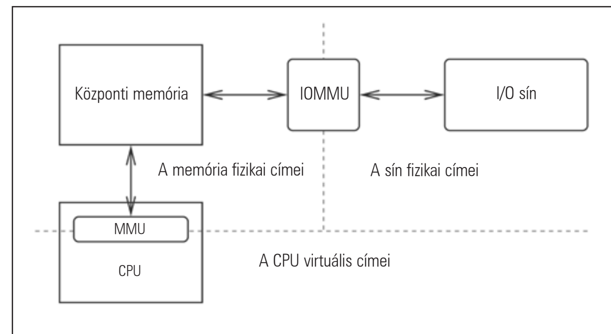
DMA segítségével az eszközök a processzor terhelése nélkül tudják írni és olvasni a memóriát, amivel felgyorsul a be- és kiviteli műveletek végrehajtása. Tekintsük át, hogy a rendszermag hogyan követi figyelemmel a processzor háta mögött történő eseményeket.

A DMA (***direct memory access***) a közvetlen memória-hozzáférés rövidítése, és segítségével az eszközök, rendszerelemek képessé válnak a központi memória tartalmának módosítására. Mindezt úgy, hogy az adatoknak nem kell áthaladniuk a processzoron. A DMA használatának előnye pontosan a processzor munkájának zavartalanságából fakad. A rendszer kérheti az adatok elhelyezését adott memóriaterületen, és a művelet befejezéséig nyugodtan foglalkozhat egyéb teendőivel. A DMA használatával kapcsolatos bajok nagy része ugyanakkor éppen abból ered, hogy a processzort kihagyjuk a játékból.

Ezek a bajok három csoportra oszthatók. Az első oka, hogy a processzor jó eséllyel memóriakezelő egységgel rendelkezik. Az a cím, amelyet a processzor a memória adott területének címzésére használ, nem feltétlenül egyezik meg a terület fizikai címével. Másodszor, a központi memóriába történő írások miatt előfordulhat, hogy a memória és a processzor közötti gyorstárak tartalma elavulttá válik. (Lásd: ***„A gyorstárazás rejtelmei"***, ***Linuxvilág, 2004. április***)

Harmadszor: a be- és kiviteli (I/O) sínen is lehet memóriakezelő egység (IOMMU). Ebből következően előfordulhat, hogy az eszköz által az adatok továbbításakor használt síncím nem egyezik meg a fizikai memóriacímmel vagy a processzor által használt képzetes memóriacímmel. Az ilyen megoldások az x86-os világ lakói számára meglehetősen idegennek hatnak. A GART-ok (***graphical aperture remapping table*** – grafikus ablakleképező tábla) AGP sínen való használatával ugyanakkor az x86-hívők IOMMU vonatkozású ellenállása gyengülni látszik.

A Linux rendszermag DMA-kezelő API-jának mindhárom problémaforrást figyelembe kell vennie, és a gondok tényleges fellépését meg kell előznie. Mindezek mellett, mivel az eszközök felől a DMA alapú átvitelek túlnyomó része egy külső sínen zajlik, újabb három kérdés merül fel. Az első az, hogy a I/O eszköz címszélessége eltérhet a fizikai memóriacímek szélességétől. Az ISA-s eszközök például 24 bites, a 64 bites rendszerekben pedig egyes PCI sínre illeszkedő eszközök csak 32 bites címzésre képesek. A második, hogy a I/O sín vezérlője is gyorstárazhatja a kéréseket. Ilyesmire elsősorban a PCI-sínen kerülhet sor, itt az írási kéréseket a PCI-vezérlő visszatarthatja abban a reményben,



1. ábra Címtartományok DMA alatt

hogy több kérést is összegyűjtve gyorsabb átvitelt valósíthat meg. Ezt a műveletet, jelenséget nevezzük PCI-kötegelésnek. A harmadik kérdéses dolog azzal függ össze, hogy az operációs rendszer olyan területre kérhet továbbítást, amely saját képzetes memóriaterében összefüggő ugyan, ám a fizikai memóriában töredezett; ennek oka általában az, hogy az átvitel több lapot is érint. Az ilyen átvitelek lebonyolításához szétszóró/gyűjtögető (***Scatter/Gather – SG***) listákra van szükség.

Írásomban kizárólag az eszközök kezelését segítő DMA API-val foglalkozom. A 2.6-os Linux új általános eszközmodellje rendkívül elegáns lehetőséget biztosít az eszközök jellegzetességeinek leírására és – egy hierarchikus fa révén – sínjellemzőik megállapítására. Az említett csatolófelületek a 2.4 - 2.6 áttérés során fontos ellenőrzéseken és módosításokon estek át. Az itt szereplő általános elvek ugyan a 2.4-es sorozatra is érvényesek, az API és a rendszermag itt említett képességei csak a 2.6-os sorozattal jelentek meg.

SG listák

A DMA alapú átviteleknél figyelembe kell venni, hogy a felhasználó nagy mennyiségű, akár több MB-nyi adat továbbítását kérheti az adott pufferbe. A képzetes memória kezelési módja miatt előfordulhat, hogy ez a képzetes térben összefüggő terület számos, a fizikai memória különféle részeire szétszórt lapból áll. A Linux elvárja, hogy minden lapnyi méretűnél (x86-os rendszereken ez 4 KB) nagyobb

© Kiskapu Kft. Minden jog fenntartva

átvitel SG lista segítségével történjen. A listát normál esetben a blokkos I/O réteg (BIO layer) állítja elő. Az illesztőprogram egyik alapvető feladata a blokkos I/O réteg azon működési beállításainak megadása, amelyek alapján a I/O műveletek SG listaelemekre való felosztása történik.
Szinte minden olyan eszközt, amely nagy mennyiségű adat mozgatását végzi, úgy terveznek, hogy az ilyen átviteli kéréseket SG listák formájában tudja fogadni. Ugyan a listák pontos formátuma a legtöbb esetben eltér a rendszermag által előállítottól, az átalakítás csak kivételes esetben jár komolyabb nehézséggel.

I/O memóriakezelő egységek (IOMMU)

Az IOMMU olyan memóriakezelő egység, amely a I/O sín (vagy több szintre rendeződő sínek) és a központi memória között helyezkedik el. Működése teljesen független a processzorba épített memóriakezelő egységétől. Ha egy eszköz és a központi memória között átvitelt szeretnénk indítani, az IOMMU-t fel kell programozni az átvitel során szükséges címfordításokkal, nagyjából úgy, ahogy a processzor memóriakezelő egységével tennénk ezt. Ha ezt megtesszük, akkor a blokkos I/O réteg által létrehozott SG listák úgy adhatók meg az IOMMU-nak, hogy a sínen lógó eszköz számára a memóriaterület újfent összefüggőnek tűnjön.

GART-ok és az IOMMU kihagyása

A GART alapvetően egy egyszerű IOMMU. Egy, a fizikai memóriában található ablakból és egy laplistából áll. Feladata az ablakba eső fizikai címek leképezése a listában szereplő fizikai lapokra. Az ablak általában elég keskeny, például 128 MB méretű, így a fizikai memóriának az ablakon kívülre eső elérései nem képeződnek le.
Ez az apró hiányosság azonnal rá is világít a Linux rendszermag jelenlegi DMA-kezelésének egyik hibájára: a DMA API-k egyike sem képes hibával visszatérni, ha a memórialeképezés sikertelen. A GART-ok csak korlátozott nagyságú leképezési címtérrel rendelkeznek, és ha ez kimerül, akkor nincs több leképezésre lehetőség, legalábbis amíg néhány I/O művelet be nem fejeződik, és kellő nagyságú terület fel nem szabadul.
Egyes esetekben a GART-okhoz hasonlóan az IOMMU-kat is fel lehet programozni úgy, hogy a I/O sín és a meghatározott ablakokba eső memória között ne végezzenek címleképezést.
Ezt kihagyásos módnak nevezzük, és érdemes tudni róla, hogy nem minden IOMMU-típus támogatja. A kihagyásos módot akkor érdemes használni, ha a leképezés miatt romlik a teljesítmény, és az IOMMU-t eltávolítva az adatok útjából nagyobb átviteli sebességet lehet elérni.
A blokkos I/O réteg alapesetben feltételezi, hogy ha található IOMMU a rendszerben, akkor használjuk is azt, és ennek megfelelően számítja ki az eszközök SG listája által igényelt hely méretét. Jelenleg nincs lehetőség arra, hogy a blokkos I/O rétegnek jelezzük, az eszköz ki szeretné hagyni a folyamatokból az IOMMU-t.
Baj akkor van, ha a blokkos I/O réteg IOMMU jelenlétét feltételezi, valamint azt, hogy az SG bejegyzéseket az IOMMU állítja össze. Ilyenkor, ha az illesztőprogram az IOMMU kihagyása mellett dönt, akkor több SG bejegyzés keletkezik, mint amennyit az eszköz megenged.
Már megkezdődtek azok a munkák, amelyek e két hibaforrásnak a 2.6-os rendszermagból való eltüntetését célozzák. Az IOMMU kihagyásos hibára készült javítást már vizsgálják. A megoldás az illesztőprogramok készítői számára láthatatlan lesz, a kihagyásokról ugyanis az alapkód fog határozni. A leképezési hiányosságok elhárításához valószínűleg a leképező API-k hibajelzési képességeinek megvalósítására lesz szükség. Mivel a munka a rendszer minden DMA-illesztőprogramjára kihat, elvégzése várhatóan eléggé el fog húzódni.

Sínszélességek és DMA maszkok

Annak érdekében, hogy a lehető legnagyobb címzési szélességgel végezhesse az adatcseréket, minden általános eszköz rendelkezik egy DMA maszk nevű értékkel, amely az elérhető címvonalakat kiválasztó maszkbiteket tartalmazza, és amelynek beállítása az illesztőprogram feladata. A DMA szélességmaszk kétféle jelentést hordozhat, attól függően, hogy használunk-e IOMMU-t. Ha igen, a DMA maszk egyszerűen a leképezhető síncímek tartományát korlátozza, ám az IOMMU segítségével az eszköz a fizikai memória minden részét el tudja érni. Ha nincs IOMMU, a DMA maszk abszolút korlátot jelent az eszközre nézve. Ekkor az eszköz a fizikai memóriának a maszkon kívülre eső részeire nem tud adatokat továbbítani.
A szétszóró/gyűjtögető listák készítésekor a blokkos réteg a DMA maszk alapján dönti el, hogy az adott lapot át kell-e tükrözni. Tükrözés alatt most azt értjük, hogy a blokkos réteg vesz egy, a DMA maszkon belülre eső lapot, és minden adatot átmásol rá egy a tartományon kívülre eső lapról. Amikor a DMA használata befejeződött, a blokkos réteg visszamásolja az adatokat a tartományon kívüli lapra, majd felszabadítja a tükrözött lapot. Természetesen az oda-vissza másolgatás rontja a hatékonyságot, ezért a gyártók általában arra törekednek, hogy kiszolgáló kategóriájú gépeiknek ne legyenek DMA maszk-vonatkozású korlátaik.

A DMA és az egységesség viszonya

A DMA alapú átvitelek a processzor közreműködése nélkül folynak, tehát a rendszermagnak biztosítania kell egy olyan API-t, amely képes a processzor gyorstárainak tartalmát összhangba hozni a DMA alapú átvitel által érintett memóriaterületek tartalmával. Fontos, hogy a DMA API a rendszermag képzetes címeinek figyelembe vételével biztosítsa a processzor gyorstárainak frissítését. A gyorstáraknak a felhasználói térbeli memóriatartalom változásait követő frissítéseire egy másik API-t kell használni.

Sínkötegelés

A felső kategóriás sínvezérlő lapkák néha gyorstárral is rendelkeznek. Az alapötlet az, hogy a processzor felől a lapkakészlet felé végzett írások végrehajtása gyors, a sínen keresztül végrehajtottaké viszont lassú, vagyis ha a sínvezérlő gyorstárazza az írásokat, a processzornak nem kell megvárnia befejezésüket. A sínkötegeléssel ahogy az ilyen jellegű gyorstárazást nevezzük az a baj, hogy a processzornak nincs utasítása a sín gyorstárának kiürítésére. A gyorstár ürítése – a helyes sorrend biztosítása érdekében – szigorú szabályrendszer szerint történik. Az első szabály az, hogy csak memória alapú írásokat szabad gyorstárazni, a I/O té-

© Kiskapu Kft. Minden jog fenntartva

ren keresztül történőket nem. A második szabály szerint a memória alapú olvasások és írások sorrendjét szigorúan meg kell őrizni, még az írások gyorstárazásakor is. Az utolsó szabály lehetővé teszi az illesztőprogram írója számára a gyorstár kiürítését. Ha memória alapú olvasást indítunk az eszköz memóriaterületének bármely szakaszára, az összes gyorstárazott írás elvégzésére garantáltan sor kerül az olvasás megkezdése előtt.
A kötegelés kezelésében semmilyen API segítségére nem számíthatunk, az illesztőprogramok íróinak tehát maguknak kell ügyelniük a kötegelési szabályok betartására, amikor olvassák vagy írják az eszköz memóriaterületét. Egy ötlet: ha végképp nem tudunk semmilyen értelmes és szükséges olvasási műveletet kitalálni, amivel elérhetnénk a függőben lévő írások végrehajtását, egyszerűen olvassunk ki egy szakaszt az eszköz sínbeállításai közül.

A DMA API használata illesztőprogramban

Az API-ról kimerítő leírást találhatunk a rendszermag leírásait tartalmazó könyvtárban (***Documentation/DMA-API.txt***). Az általános DMA API-nak van egy párja is, amely csak PCI eszközökkel működik, ennek leírása a ***Documentation/DMA-mapping.txt*** fájlban található. A továbbiakban szeretném nagy vonalakban áttekinteni azokat a lépéseket, amelyek a DMA helyes működésének eléréséhez szükségesek. Részletesebb tájékoztatást az imént említett leírásokban lehet találni.
Az illesztőprogram elindításakor az első lépés a DMA beállítása:

```
int dma_set_mask(struct device *dev, u64 mask);
```

Itt a `dev` az általános eszköz, a `mask` pedig a beállítani kívánt maszk. A függvény igaz értékkel tér vissza, ha a maszkot a rendszer elfogadta, és hamissal, ha nem. A maszk visszautasítására akkor kerülhet sor, ha a tényleges rendszerszélesség kisebb. Egy 32 bites rendszer például elutasítja a 64 bites maszkot. Ha tehát tudjuk, hogy az eszköz 64 bites címzésre képes, először 64 bites maszkkal kell próbálkoznunk, majd ha ennek beállítása sikertelen, akkor 32 bitessel.
A következő művelet a várakozási sor lefoglalása és kezdeti értékek megadása. Ennek ismertetése meghaladja írásom kereteit, de a ***Documentation/block/*** alatt pontos leírást lehet találni róla. Ha megvan a várakozási sor, két alapvető beállítást kell megadnunk. Elsőként állítsuk be az SG tábla legnagyobb méretét (vagy engedélyezzük tetszőleges nagyságú elfogadását):

```
void blk_queue_max_hw_segments(request_queue_t *q,
                               ➥unsigned short
                               ➥max_segments);
```

Másodikként – szükség szerint – adjuk meg a legnagyobb méretet:

```
void blk_queue_max_sectors(request_queue_t *q,
                     ➥unsigned short max_sectors);
```

Az utolsó lépés a DMA maszk beprogramozása a várakozási sorba:

```
void blk_queue_bounce_limit(request_queue_t *q,
                            ➥u64 max_address);
```

Általában a `max_address` a DMA maszkkal egyenlő, ám ha IOMMU-t használunk, akkor a `max_address`-t `BLK_BOUNCE_ANY` értékre kell állítani, ezzel tilthatjuk meg a blokkos réteg számára a tükrözést.

Az eszköz működése

Egy eszköz működéséhez szükség van egy `request` függvényre (lásd a blokkos I/O leírását), az alábbi paranccsal ez fogja kivenni a várakozási sorból a kéréseket:

```
struct request *elv_next_request(request_queue_t *q);
```

A kérésben igényelt leképezési bejegyzések számát a `req-nr_phys_segments` tartalmazza. Létre kell hoznunk egy ilyen méretű ideiglenes táblát, `sizeof(struct scatterlist)` méretű egységekkel, majd ideiglenes leképezést kell megadnunk:

```
int blk_rq_map_sg(request_queue_t  *q,
                  ➥struct  request  *req,
                  ➥struct scatterlist *sglist);
```

Ezzel megkapjuk a felhasznált SG listabejegyzések számát. Az alábbi paranccsal a blokkos réteg által biztosított ideiglenes táblát kapjuk meg, ennek leképezése végül így történik meg:

```
int dma_map_sg(struct  device  *dev,
        ➥struct scatterlist *sglist, int count,
        ➥enum dma_data_direction dir);
```

Itt a `count` a visszatérési érték, az `sglist` pedig a `blk_rq_map_sg` függvénynek átadott listával egyezik meg. A visszatérési érték a kérésben igényelt SG listabejegyzések tényleges száma.
Az SG listát újra felhasználjuk és a tényleges bejegyzésekkel töltjük fel, amelyeket az eszköz SG bejegyzéseibe kell beprogramozni. A `dir` némi segítséget nyújt a gyorstárak tartalmának egységesen tartásához. Háromféle értéket vehet fel:

1. `DMA_TO_DEVICE`: Az adatátvitel a memória felől az eszköz felé történik.
2. `DMA_FROM_DEVICE`: Az eszköz a központi memóriába továbbít adatokat.
3. `DMA_BIDIRECTIONAL`: Nem tudunk semmit az adatátvitel irányáról.

Az SG lista végigjárásakor és az eszköz SG listájának feltöltésekor két makrót kell használni:

```
dma_addr_tsg_dma_address(struct scatterlist
                        ➥*sglist_entry);
```

és:

```
int sg_dma_len(struct scatterlist *sglist_entry);
```

© Kiskapu Kft. Minden jog fenntartva

Ezek rendre az egyes bejegyzésekhez tartozó fizikai síncímet és szegmenshosszt adják vissza. A kérés leképezését azért két lépésben végezzük, mert a blokkos I/O réteget általános kódként tervezték, és így nincs kapcsolatban a géptípustól függő, az IOMMU programozására képes alapréteggel. Az egyetlen dolog tehát, amelyet a blokkos I/O réteg ki tud számítani, az az IOMMU által a kérés miatt létrehozott SG szegmensek száma. A blokkos I/O réteg nem ismeri az IOMMU által ezekhez a szegmensekhez rendelt síncímeket, ezért egy az összes leképezendő lap fizikai címét tartalmazó listát kell átadnia. A géptípustól függő réteggel a `dma_map_sg` függvény tartja a kapcsolatot, továbbá ez végzi az IOMMU programozását és a fizikai síncímek listájának lekérését is. Ez a két tényező az, ami miatt a blokkos I/O réteg listája több elemet tartalmaz, mint a DMA API által visszaadott.
A DMA alapú átvitel befejezése után a DMA tranzakciót le kell zárni:

```
int dma_unmap_sg(struct  device *dev,
                  ➥struct scatterlist *sglist,
                  ➥int hwcount,
                  ➥enum dma_data_direction dir);
```

Itt az összes átadott érték a `dma_map_sg` hívásakor alkalmazottal egyezik, kivéve a `hwcount`ot, amely a függvény visszatérési értékét tartalmazza. Végső lépésként a korábban lefoglalt SG listát fel kell szabadítani, a kérés ezzel teljesítettnek tekinthető.

DMA területen található adatok elérése

Az illesztőprogramok általában nem nyúlnak hozzá az általuk továbbított adatokhoz. Előfordulhat azonban, hogy az illesztőprogramnak meg kell vizsgálnia vagy módosítania kell az adatokat, mielőtt továbbadná őket a blokkos rétegnek. Ehhez a processzor gyorstárait összhangba kell hozni az adatokkal:

```
int dma_sync_sg(struct device *dev,
               ➥struct scatterlist *sglist,
               ➥int hwcount,
               ➥enum dma_data_direction dir);
```

Az átadott értékek a `dma_unmap_sg` hívásnál látottakkal egyeznek.
Az adatok elérésével kapcsolatosan a legfontosabb tényező az elérés időzítése. Az elérési szabályok a `dir` értékétől függenek:

- `DMA_TO_DEVICE`: Az adatok módosítása után és az eszköznek való elküldése előtt meg kell hívni az API-t.
- `DMA_FROM_DEVICE`: Az API-t az után kell meghívni, hogy az eszköz átadta az adatokat, de még az előtt, hogy az illesztőprogram megpróbálná olvasni őket.
- `DMA_BIDIRECTIONAL`: Az API-t kétszer kell meghívni, először az adatok módosításának elvégzése és az eszköznek való elküldése között, másodszor az eszköz munkájának befejezése és az adatok illesztőprogram által történő elérése között.

Egységes memóriaterület foglalása

A legtöbb eszköz postafiók jellegű memóriaterületek segítségével tartja a kapcsolatot az illesztőprogrammal.
A postafiók-memóriatartományt az illesztőprogramon kívül más általában nem használhatja. A postafiók egységességének fenntartása az előbbi API segítségével eléggé nehézkes volna, ezért a rendszermag lehetőséget biztosít olyan memóriaterület foglalására, amelyre minden pillanatban garantált az eszköz és a processzor közötti adategyezés:

```
void *dma_alloc_coherent(struct device *dev,
                         ➥size_tsize,
                         ➥dma_addr_t *physaddr,
                         ➥int flag);
```

Ezzel egy `size` méretű, egységes tartomány képzetes címét kapjuk vissza, amely egyben egy fizikai síncímmel (`physaddr`) is rendelkezik az eszköz felé.
A `flag` segítségével a foglalás típusát adhatjuk meg, ez `GFP_KERNEL` (a memóriafoglalás alvó állapotba is kerülhet teljesítése előtt) vagy `GFP_ATOMIC` (a foglalás nem kerülhet alvó állapotba, sikertelenség esetén `NULL` értékkel kell visszatérni) lehet. Az API segítségével foglalt terület garantáltan folytonos a képzetes és a fizikai sínmemóriában egyaránt. Megkötés, hogy a méretnek 128 KB alatt kell lennie.
Az illesztőprogram eltávolításának részeként az egységes memóriaterületet a következő módon kell felszabadítani:

```
void dma_free_coherent(struct device *dev,
                       ➥size_tsize,
                       ➥void *virtaddr,
                       ➥dma_addr_t *physaddr);
```

Ebben az esetben `size` az egységes terület mérete, a `virtaddr` és a `physaddr` visszatérési érték pedig rendre képzetes processzor oldali és fizikai síncíme.

Összefoglalás

Írásomban villámgyors, felületes áttekintést adtam a blokkos réteg és az illesztőprogramok közötti, az eszközök programozását szolgáló SG listák előállítását célzó párbeszédről.
A DMA API számos egyéb hasznos összetevőt tartalmaz, köztük olyan API-kat, amelyek töredezéstől mentes fizikai memóriaterületeket kezelnek. Ha sikerült felkeltenem az érdeklődésedet, akkor következő lépésként a rendszermag ***Documents*** könyvtárának és forráskódjának tanulmányozását javaslom.

Linux Journal 2004. május, 121. szám

James Bottomley a SteelEye programtervezője és a nyílt forrású közösség aktív tagja. Ő felel a SCSI alrendszer, a Linux Voyager átültetés és az 53c700 illesztőprogram karbantartásáért, továbbá DMA/eszköz modellezési-elvonatkoztatási munkájával a PA-RISC Linux fejlesztésébe is bekapcsolódott.