

InfiniBand és Linux

Vajon miért is jó, ha a hálózatunk távoli rendszere irkálhat a memóriánkba, hogyan küldhetnek felhasználói programok a rendszermag zaklatása nélkül adatokat, valamint néhány újabb adalék a nagy sebességű kapcsolattartásról.

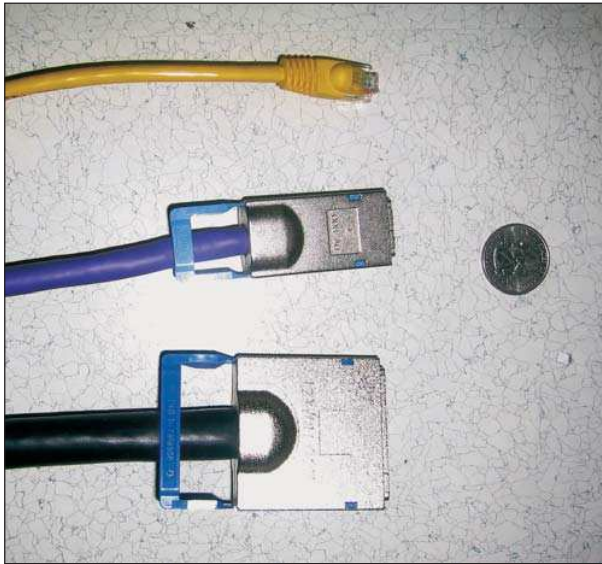
Hosszú vajúrást követően végül megszületett az **InfiniBand (IB)**, sőt a **Linux IB** támogatása is már fejlesztés alatt áll. Fizikai szinten az **IB** nagyon hasonlít a **PCI Express**-re. Az adatokat több nagy sebességű soros csatornán továbbítja. Az **InfiniBand** első változatának leírása valamennyi csatornán azonos jelzésszélességet engedélyezett, mégpedig 2.5Gb/s-ot, akárcsak a **PCI Express**. A leírás legutóbbi változata (1.2), ugyanakkor már támogatja a csatornánkénti 5Gb/s és 10Gb/s sebességeket. Ezen kívül az **IB** az 1X, 4X, 8X és 12X, míg a **PCI Express** a X1, X2, X4, X8, X12, X16 és X32 szélességeket támogatja. A leggyakrabban használt **IB** sebesség manapság a 4X 2.5Gb/s/csatorna sebesség mellett, azaz összesen 10Gb/s. A 12X szélesség a 10Gb/s/csatorna sebességgel kombinálva azonban a jelenlegi **IB** leírás szerint akár a lenyűgöző 120Gb/s átviteli teljesítményt is támogatja. Minthogy az **IB** általában hálózatépítésre használatos, a réz és optikai kábelezést egyaránt támogatja, míg a **PCI Express** kábelmeghatározásai még fejlesztés alatt állnak. A legtöbb **IB** telepítés rézkábelezést alkalmaz (1. ábra) amely körülbelül 10 méterig használható. Az **IB** ezen kívül lehetővé tesz optikai kábelezést is, amely elméletileg akár 10 kilométerig is használható.

Az elmúlt években az **IB**-t tekintették a **PCI** utódjának, de manapság már nem ez a helyzet. Az **IB** adapterek továbbra is inkább külső eszközök maradnak amelyek **PCI**, **PCI Express**, **HyperTransport** vagy hasonló periféria buszokon keresztül csatlakoznak a rendszerhez.

A rendszereket az **IB** hálózathoz csatoló hálózati eszközt **Host Channel Adapternek (HCA)** nevezik. Az eszköz igen magas sebessége mellett az **IB HCA** egy üzenetközvetítő felületet is biztosít, melynek segítségével az **InfiniBand** 10Gb/sec (vagy még nagyobb) sebességét ki tudják aknázni a rendszerek. Az **IB** sebességének kihasználásának kulcsa a zéró másolású hálózatkezelés támogatása; máskülönben az alkalmazások egyfolytában csak az adataikat másolják. A zéró másolást a **HCA** felület három kulcseleme teszi lehetővé: az elvont, magasszintű munkasor, a rendszermag megkerülés és a **távoli közvetlen memória elérés (Remote DMA; RDMA)**. Az elvont munkasor annyit jelent, hogy az alkalmazások nem csomagról csomagra formálják meg és dolgozzák fel a hálózati csomagokat, hanem kérélmeket küldenek a **HCA** által kezelt soroknak. Egy munkakérelmen belül küldött üzenetet legfeljebb 4GB hosszú lehet. A **HCA** felügyeli az üzenet csomagokra tördelését illetve várakozik a jóváhagyásokra és küldi

1. táblázat **Összehasonlító tábla**

Technológia	Adatátviteli sebesség	Kábelezés
USB	12Mb/s	5m
Hi-Speed USB (USB 2.0)	480Mb/s	5m
IEEE 1394 (FireWire)	400Mb/s	4m
Gigabit Ethernet	1,000Mb/s	100m (cat5 kábel)
10 Gigabit Ethernet	10,000Mb/s	10m (réz IB kábel), 1+ km (optikai)
Myrinet	2,000Mb/s	10m (réz), 200m (optikai)
1X InfiniBand	2,000Mb/s	10m (réz), 1+ km (optikai)
4X InfiniBand	8,000Mb/s	10m (réz), 1+ km (optikai)
12X InfiniBand	24,000Mb/s	10m (réz), 1+ km (optikai)



1. ábra Felülről lefelé: cat5 Ethernet kábel, 4X InfiniBand kábel és 12X InfiniBand kábel (méretarány kedvéért egy negyeddolláros érme látható)

1. lista IPoIB Meghajtó alaphelyzetbe állítása

```
struct ib_qp_init_attr init_attr = {
    .cap = {
        .max_send_wr = IPOIB_TX_RING_SIZE,
        .max_recv_wr = IPOIB_RX_RING_SIZE,
        .max_send_sge = 1,
        .max_recv_sge = 1
    },
    .sq_sig_type = IB_SIGNAL_ALL_WR,
    .rq_sig_type = IB_SIGNAL_ALL_WR,
    .qp_type = IB_QPT_UD
};
priv->pd = ib_alloc_pd(priv->ca);
priv->cq = ib_create_cq(priv->ca,
    ipoib_ib_completion,
    NULL, dev,
    IPOIB_TX_RING_SIZE +
    IPOIB_RX_RING_SIZE + 1);
if (ib_req_notify_cq(priv->cq, IB_CQ_NEXT_COMP))
    goto out_free_cq;
priv->mr = ib_get_dma_mr(priv->pd,
    IB_ACCESS_LOCAL_WRITE);
init_attr.send_cq = priv->cq;
init_attr.recv_cq = priv->cq;
priv->qp = ib_create_qp(priv->pd, &init_attr);
```

újra az elveszett csomagokat. Mivel a *HCA* alkatrész felügyeli a nagyméretű üzenetek kézbesítését bármiféle *CPU* felhasználás nélkül, az alkalmazások több *CPU* időt használhatnak fel a küldött és fogadott adatok elkészítéséhez és feldolgozásához.

A rendszermag megkerülése lehetővé teszi, hogy az alkalmazások közvetlenül küldjenek munkakérelmeket és fogadjanak befejezési eseményeket a *HCA* sorairól, így a rendszerhívásokkal járó felesleges környezetváltás elkerülhető. A rendszermag meghajtó beállítja a sorokat, ahol a szokásos memória védelem gondoskodik arról, hogy minden folyamat csak a saját erőforrásait érje el. Valamennyi nagysebességű irányítási (path) művelet ugyanakkor a felhasználói térben megy végbe.

Az utolsó elem, az *RDMA*, lehetővé teszi, hogy az üzenet magával vigye a célcímet ahová a memóriában majd kerülnie kell. Az adat helyének megadása például az *IB* alatti tárolók kialakításakor lehet hasznos, ahol a kiszolgáló lemezelvasásai teljesen rendszertelenek lehetnek. *RDMA* nélkül vagy a kiszolgálónak kell időt pazarolnia mikor áll elküldésre készen az adat vagy az ügyfélnek kell *CPU* erőt pazarolnia az adatok végső helyre másolásához.

Bár a távoli rendszerek memóriájába firkálással szemben vannak bizonyos fenntartások az *IB* használó alkalmazások szigorú határait és jogosultságokat állíthatnak be az *RDMA* szolgáltatásnak. Az *IB RDMA* legalábbis biztonságosabb dolog mint a lemezevezérlő *DMA*-ját a memóriába engedni.

Nagy sebességén felül, az *IB* egyúttal a *fürtök (clusters)* készítését és kezelését is leegyszerűsíti, hiszen egyetlen eszköz szállítja a hálózati és tároló forgalmat valamint a fürt kommunikációt. Több csoport is hozott létre különféle *IB* felett futtatható felső szintű protokollokat, néhány ezek közül:

- **IP-over-InfiniBand (IPoIB):** az *Internet Engineering Task Force (IETF)* szabványfejlesztő munkacsoportja az *IB* alatti *IP* forgalomküldés ösvényét tapossa ki. Ezek az ösvények idővel a *IpoIB RFC* szabványává válhatnak. Az *IPoIB* ugyanakkor nem használja maximálisan ki az *IB* képességeit, hiszen a forgalom továbbra is az *IP* vermen keresztül utazik és csomagról csomagra kerül elküldésre. Az *IPoIB* egyszerű lehetőséget biztosít régebbi alkalmazásaink futtatására vagy a vezérlőforgalom átküldésére *IB*-n keresztül.
- **Sockets Direct Protocol (SDP):** az *InfiniBand Trade Association* maga adott meg egy olyan protokollt, amely a szabványos socket műveleteket helyi *IB RDMA* műveletekké alakítja. Ezáltal a socket alkalmazások változtatás nélkül futtathatók, ugyanakkor az *IB* teljesítményének szinte minden előnyét élvezhetik.
- **SCSI RDMA protokoll (SRP):** a *SCSI* szabványokért felelős *InterNational Committee for Information Technology Standards (INCITS) T10* bizottság, kiadott egy szabványt, amely leírja hogyan kell a *SCSI* protokollt az *IB*-hoz rendelni. A második generációs *SRP-2* protokoll fejlesztése jelenleg folyik.

Sok más csoport is tanulmányozza az *IB* kihasználási lehetőségeit, például a *DAT Collaborative* és az *Open Group Interconnect Software Konzorciuma* készített *API*-kat, *RDMA* csatlósokat az *NFS*-hez valamint *IB* támogatást különféle *MPI* csomagokhoz.

2. lista IPoIB meghajtó küldési kérelem feladása

```

priv->tx_sge.lkey          = priv->mr->lkey;
priv->tx_sge.addr         = addr;
priv->tx_sge.length       = len;
priv->tx_wr.opcode        = IB_WR_SEND;
priv->tx_wr.sg_list       = &priv->tx_sge;
priv->tx_wr.num_sge       = 1;
priv->tx_wr.send_flags    = IB_SEND_SIGNALED;
priv->tx_wr.wr_id         = wr_id;
priv->tx_wr.wr.ud.remote_qp = qpn;
priv->tx_wr.wr.ud.ah      = address;

ib_post_send(priv->qp, &priv->tx_wr, &bad_wr);

```

Természetesen nyílt forráskódú támogatás nélkül az egész csinos eszköz nem lenne különösebben érdekes a *Linux* világban. Szerencsére az *OpenIB Alliance* ipari szövetség feladata pontosan az, hogy egy teljesen nyílt forrású *IB* vermet hozzon létre. Az *OpenIB* jelenleg 15 tag-céggel rendelkezik, amelyek között egyaránt találunk *IB* alkatrészt terjesztőket, kiszolgálócégeket, programcégeket és kutatási szervezeteket.

Az *OpenIB* programon 2004 februárjában kezdődtek meg a munkálatok, az első rendszermag meghajtó pedig 2004 decemberében került a rendszermag fába, közvetlen azután, hogy a 2.6.10-es kiadás után megszületett a 2.6.11-es fa. A rendszermagba illesztett első *IB* meghajtó kód csomag éppen csak annyit tud, hogy már azért működőképes. Tartalmaz egy középhéteget, amely elfedi az alacsony szintű alkatrészmeghajtókat a felső szintű protokollok elől, egyetlen alacsony szintű meghajtót a *Mellanox HCA*-hoz, valamint egy *IPoIB* felső szintű protokoll meghajtót amellyel egy alhálózat kezelőt futtatunk a felhasználói térben.

Az *IPoIB* meghajtó kódjának néhány apró részlete sokat elárulhat a rendszermag *IB* támogatásának felhasználási lehetőségeiről. Amennyiben egészében szeretnénk látni a kódot, a teljes *IPoIB* meghajtót kell megnéznünk, amelyet egyébként a *Linux* forrásfájának *drivers/infiniband/ulp/ipoib* könyvtárban találjuk meg.

Az 1. lista bemutatja, mit is csinál az *IPoIB* meghajtó az *IB* erőforrások lefoglalásához. Először meghívja az `ib_alloc_pd()` függvényt, amely lefoglalja a *védelmi tartományt (protection domain vagy PD)*, ez egy átlátszó tároló amelyre valamennyi *IB* felhasználónak szüksége van az erőforrások tárolásához.

Itt most kihagytuk a listából a helyes hibakezelést, annak ellenére, hogy minden valódi rendszermag kód valamennyi függvény visszatérési értékét ellenőrizné. Minden erőforrást foglaló és mutatót visszaadó *IB* függvény a szokásos *Linux* módszert alkalmazza a hibák visszaadására az `ERR_PTR()` makrón keresztül, következésképpen az állapotot az `IS_ERR()` segítségével le lehet kérdezni. Például a `ib_alloc_pd()` hívás a valódi rendszermag kódban a következőképpen nézne ki:

```

priv->pd = ib_alloc_pd(priv->ca);
if (IS_ERR(priv->pd)) {
    printk(KERN_WARNING "%s: failed "
           "to allocate PD\n", ca->name);
    return -ENODEV;
}

```

Ezek után a rendszermag meghívja az `ib_create_cq()` függvényt, amely létrehozza az *befejeződési sort (completion queue, azaz CQ)*. A meghajtó igényli, hogy az befejeződési esemény bekövetkeztekor meghívódjon az `ipoib_ib_completion()` függvény, valamint, hogy a *CQ* legalább `IPOIB_TX_RING_SIZE + IPOIB_RX_RING_SIZE + 1` darab munkabefejezési struktúrát tárolhasson. Erre a méretre abban a különleges esetben van szükség, amikor a meghajtó a lehető legnagyobb számú üzenetet küldi és és fogadja majd nem tud lefutni, míg valamennyi el nem készül. Elég zavaró módon, a *CQ*-k olyan *IB* erőforrások, amelyek nincsenek kapcsolatban a *PD*-kel, ezért a függvénynek nem is kell átadnunk *PD* adatot.

Amikor a *CQ* elkészült a meghajtó az `ib_req_notify_cq()` függvényhez fordul és kérelmezi, hogy amikor a következő munkabefejezés a *CQ* sorba kerülésére hívódjon meg a befejezési esemény függvény. Az `ipoib_ib_completion()` eseményfüggvény feldolgozza a befejezéseket amíg a *CQ* ki nem ürül, majd meghívja az `ib_req_notify_cq()`-t. Így újra elindul, amint újabb befejezések válnak elérhetővé. A meghajtó ekkor meghívja az `ib_get_dma_mr()`-t amely beállítja a rendszermag *DMA* osztó *API*-tól kapott *DMA* címmel használható memóriaterületet (*MR*). Ennek kezeléséhez az *IB HCA*-ban to elkészülnek az átalakító táblák, és egy helyi kulcsot (`L_Key`) adunk vissza, amelyet aztán tovább lehet adni a *HCA*-nak, és hivatkozhat az adott *MR*-re.

Végül a meghajtó az `ib_create_qp()`-t hívja meg, amely létrehoz egy sorpárt (*QP*). Ezt az objektumot azért nevezzük sorpárnak mert két darab munkasort tartalmaz. Egyet a küldési kérelmeknek és egyet a fogadási kérelmeknek. A *QP* létrehozásához először ki kell töltenünk a meglehetősen méretes `ib_qp_init_attr` szerkezetet. A fedőstruktúra a készítenő küldő és fogadó sorok méretét adja meg. A `sq_sig_type` és az `rq_sig_type` mezőket `IB_SIGNAL_ALL_WR` értékre állítjuk, így minden munkakérelem befejezési eseményt vált ki.

A `qp_type` mezőt `IB_QPT_UD` értékre állítjuk, létrehozva ezzel a *megbízhatatlan adatsomag (unreliable datagram avagy UD) QP*-t. Az *IB QP* esetében négyféle szállításról beszélhetünk: *megbízhatóan összekapcsolt (RC)*, *megbízható adatsomag (RD)*, *megbízhatatlanul kapcsolt (UC)* és *megbízhatatlan adatsomag (UD)*.

A megbízható szállítások esetében az *IB* alkatrészt garantálni tudja, hogy minden egyes üzenet vagy sikeresen kézbesítődik, vagy hibát jelez amennyiben a hibát egy javíthatatlan hiba okozza (például kihúzzuk a kábellet). A kapcsolt adatszállítás esetében minden üzenet egyetlen célállomásra irányul, amelyet a *QP* beállítása során határozunk meg, az adatsomag szállítás esetében minden egyes csomag különböző célokra irányulhat. Amikor az *IPoIB* meghajtó elkészítette a *QP*-t, a kapott csomagokat a *QP*-n keresztül küldi a hálózati verembe. A 2. lista mutatja be, hogy mit kell tennünk akkor,

ha egy kérelmet akarunk helyezni a *QP* küldési sorába. Először is a meghajtó a küldési kérelemhez felállít egy gyűjtőlistát. Az `lkey` mező annak az *MR*-nek az `l_key` értékét veszi fel, amelyet az `ib_get_dma_mr()`-ből kaptunk. Minthogy az *IPoIB* olyan csomagokat küld, amelyek egyetlen összefüggő részben találhatóak, a gyűjtőlistában mindössze egyetlen bejegyzés található. A meghajtónak csak a címet és csomag hosszát kell megadnia. A gyűjtőlistában található cím nem a virtuális cím hanem a `dma_map_single()` függvénnyel megkapott *DMA* cím lesz. Általánosságban a programok hosszabb gyűjtőlistákat is használhatnak, ezáltal rákényszerítve a *HCA*-t, hogy egyetlen üzenetbe gyűjtsön össze több puffert és így nem kell egyetlen pufferbe másolnia az adatokat.

A meghajtó ezután kitölti a munkakérelem maradék mezőit. Az `opcode`-ot `send-re` állítja, az `sg_list` és `num_sge` értékeket az éppen kitöltött gyűjtőlista címe kerül a `send_flags` értéke pedig `signaled` lesz, így a munkakérelem befejezést vált ki amikor végez. A távoli *QP* számot és címkezelőt beállítjuk, majd a `wr_id` mezőbe a meghajtó munkakérelem azonosítója kerül.

A munkakérelem kitöltése után a meghajtó meghívja az `ib_post_send()`-et, amely végül a kérelmet felveszi a sorba. Amikor a kérelmet teljesítette az *IB* alkatrész, a munka teljesítése a meghajtó *CQ* sorába kerül, amelyet végül az `ipoib_ib_completion()` dolgoz fel.

Az *InfiniBand* rengeteg mindenre képes és az *OpenIB Alliance* éppen csak elkezdte megírni a képességeit kihasználó programokat. Most, hogy már a Linux rendelkezik az

alapvető *IB* támogatással, nekiláthatunk a felsőszintű protokollok elkészítésének, ideértve például az *SDP*-t és a tároló protokollokat. A másik komolyabb terület amely jelenleg foglalkoztat bennünket, a közvetlen felhasználói térbeli *IB* elérés, azaz a korábban említett rendszermag megkerülő képesség. Rengeteg érdekes dolgot lehet még csinálni az *IB*-hez és az *OpenIB Projekt* mindenki számára nyitott, kalandra fel tehát, csatlakozz a mókához.

Linux Journal 2005. május, 133. szám

Roland Dreier Linux InfiniBand meghajtók karbantartója és vezető fejlesztője az OpenIB.org projekt keretében. Roland a Kaliforniai Berkeley egyetemen szerezte matematika doktorátusát és több különféle titulusa volt már akadémiai körökben valamint a műszaki világban. 2001 óta a Topspin Communications alkalmazásában áll.

KAPCSOLÓDÓ CÍMEK

- www.openib.org
- www.infinibandta.org
- www.ietf.org
- www.t10.org
- www.datcollaborative.org

