# B-K-ütemezők

Nézzük meg, milyen hatással lesz az új B-K-ütemező megjelenése a Linux lemezteljesítményére, illetve milyen előnyöket várhatunk a 2.6-os rendszermagban megjelent B-K-ütemezőtől.

A legtöbb Linux-felhasználó tisztában van a folyamatütemező szerepével (jó példa erre az új O(1) ütemező), azonban viszonylag kevesen ismerik az B-K (bevitel/kivitel) ütemezők működését. A B-K-ütemezők bizonyos szempontból hasonlítanak a folyamatütemezőkre: mind a kettő erőforrásokat ütemez több felhasználó között. A folyamatütemező a rendszeren futó folyamatoknak kiszabott processzoridőért felelős. Mit ütemez akkor a B-K-ütemező?

Egy rendszerben nem is feltétlenül szükséges a jelenléte, vagyis a folyamatütemezővel ellentétben a B-K-ütemező nem elengedhetetlen része az operációs rendszernek.

Az B-K-ütemező egyedüli „raison d'□tre"-je a teljesítmény. Mivel meg szeretnénk érteni a B-K-ütemezők szerepét, át kell rágnunk magunkat néhány háttéradaton, ezt követően megnézzük, miként viselkedne egy B-K-ütemező nélküli rendszer. A merevlemezek az ismerős sáv, fej, szektor felosztású, geometriai alapú címzéssel kezelik az adatokat. A merevlemez több tálcából áll, ezek mindegyike tartalmaz egy-egy lemezt, tengelyt és író–olvasófejet. Minden tálca – a CD-khez hasonlóan – további körkörös, gyűrűszerű pályákra bontható; végezetül minden ilyen sáv egész számú szektorból áll.

Ha a merevlemezen egy bizonyos adategységet keresünk, a meghajtó logikája három adatot vár tőlünk: a sáv, a fej és a szektor számát. A sáv mondja meg, hogy melyik körkörös pályán találjuk az adatot. Ha a tálcákat egymás fölé helyezzük (miként a merevlemezben is elhelyezkednek), a megadott sáv egy hengert alkot a tálcákon keresztül (innen származik az angol elnevezése: cylinder, azaz henger). A kérdéses író–olvasófejet (és ezzel egyúttal a tálcát) a fejadat adja meg. A keresést ezáltal egyetlen tálca egyetlen sávjára szűkítettük le. Végül a szektorérték azonosítja a sávunk adott szektorát. A keresés ezzel befejeződött: a merevlemez már tudja, hogy az adat melyik szektorban, melyik sávon és melyik tálcán helyezkedik el. Beállíthatja az író–olvasófejet a megfelelő tálca megfelelő sávja fölé és beolvashatja a megfelelő szektort.

Szerencsére a korszerű merevlemezek már nem kényszerítik a gépeket arra, hogy sáv-, fej- és szektoralapon tartsák velük a kapcsolatot. A jelenlegi merevlemezek minden egyes sáv–fej–szektor hármashoz egyedi blokkszámot rendelnek. A sáv–fej–szektorértéket ez az egyedi azonosító adja meg. A mostani operációs rendszerek e blokkszám használatával tarthatják a kapcsolatot a merevlemezzel (ezt nevezik logikai blokkcímzésnek), amit aztán a merevlemez a helyes sáv–fej–szektorértékre fordít le magának. Ezzel a blokkos címzéssel kapcsolatban egy dolgot érdemes megemlíteni: bár garancia nincs rá, a fizikai hozzárendelés általában sorban történik, vagyis az n-edik logikai blokk fizikailag többnyire az n+1-dik logikai blokk után következik. Hogy ez miért fontos, arról később még szó lesz.

Most nézzünk meg egy átlagos Unix-rendszert: blokkokat olvasó vagy kiíró B-K-kérelmeket a lemezen alkalmazások széles kínálata, adatbázisok, levelezőügyfelek, webkiszolgálók és szövegszerkesztők használnak. A blokkok fizikailag általában összevissza helyezkednek el a lemezen. A postaláda adatai teljesen más helyen lehetnek, mint a webkiszolgáló HTML-adatai vagy a szövegszerkesztő beállításfájljai. Igazság szerint létezhet olyan állomány is, amely fizikailag az egész lemezen található, amennyiben szét van darabolva, azaz nem egymást követő blokkokban helyezkedik el. Mivel a fájlokat egyedi blokkokba tördeltük, a merevlemez pedig blokkokat kezel és nem a jóval elvontabb fájlokat, az állomány adatainak olvasását több, különféle blokkokra hivatkozó egyedi B-K-kérelem sorozatára kell

*1. próba:* Írás miatt koplal az olvasás

Egy folyamatos írást futtassunk a háttérben, például:

```
while true
do
        dd if=/dev/zero of=file bs=1M
done
```

Mindeközben mérjük meg, mennyi időbe telik egy 200 MB méretű fájl beolvasása:

```
time cat 200mb-file > /dev/null
```

Ez jól mutatja „az írás miatt koplal az olvasás" feladat nehézségét.

lebontani. Szerencsés esetben a blokkok sorban helyezkednek el vagy legalább fizikailag közel találhatóak egymáshoz. Amennyiben a blokkok nincsenek közel egymáshoz, a lemez fejének egy másik helyre kell mozdulnia a lemezen. A lemezfej mozgatását keresésnek (seeking) nevezik, és a számítógépen ez az egyik legidőigényesebb művelet. A mai merevlemezek keresési ideje tíz milliszekundumokban mérhető. Ez az egyik oka annak, amiért jó dolog, ha az állományaink töredezettségmentesek (defragmented). Sajnos azonban nem igazán számít, hogy fájljaink töredezetlenek-e vagy sem, mivel a rendszer az egész lemezen egyszerre több állományhoz hoz létre B-K-kérelmeket.
A levelezőügyfél egy kicsit innen szeretne, a webkiszolgáló egy kicsit onnan… De várjunk csak, a szövegszerkesztő épp most szeretne beolvasni egy fájlt. Ez együttesen annyit jelent, hogy a meghajtófej összevissza ugrál a lemezen.
A legrosszabb esetben több állomány együttes B-K-kérelmeinél a fej egyfolytában csak ide-oda ugrál, ez pedig nincs valami jó hatással a rendszerteljesítményre.
Ezen a ponton lép be a képbe a B-K-ütemező. Az B-K-ütemező ütemezi a függőben lévő B-K-kérelmeket, a legkisebbre véve a lemezfej mozgatására fordított időt. Ennek megfelelően csökken a keresési idő és növekszik a merevlemez teljesítménye.
A varázslat két fő feladat: a rendezés és egyesítés segítségével hajtható végre. Először is a B-K-ütemező egy blokkszám szerint rendezett listát tart fenn a függőben lévő B-K-kérelmekről. Ha új B-K-kérelem érkezik, blokkszám szerint be kell szúrni a függőben lévő kérelmek listájára, így a meghajtófejnek nem kell összevissza szaladgálnia a lemezen, hogy a B-K kérelmeket kiszolgálhassa. Amíg a listát rendezetten tartjuk, a lemezfej szépen sorban halad a lemezen. Ha a meghajtó éppen egy kérelmet szolgál ki a lemez valamelyik részén és pontosan erre a területre vonatkozó új kérelem érkezik, akkor ezt a kérelmet ki lehet szolgálni, még mielőtt egy másik helyre mozgatnánk a fejet.
Az egyesítést olyankor használjuk, amikor az azonos vagy sorrendben következő lemezrészre hivatkozó B-K-kérelem érkezik. Új, önálló kérelem kiadása helyett a kérelmet egyszerűen egyesítjük az azonos vagy következő kérelemmel – ezáltal a legkisebbre csökkenthetjük a várakozó kérelmek számát.
Lássunk egy példát! Képzeljük el azt az esetet, amikor két alkalmazás a felsorolt blokkszámokra ad ki kérelmeket, és ezek a következő sorrendben érkeznek a rendszermaghoz: 10, 500, 12, 502, 14, 504 és 12. A B-K-ütemező nélküli megközelítés a blokkokat a megadott sorrendben szolgálná ki, ami hét hosszú keresést jelentene, oda-vissza a lemez két különböző területe közt. Micsoda pazarlás! Ha a rendszermag a kérelmeket sorba rendezte és egyesítette volna, és ilyen módon szolgálta volna ki őket, ez az eredmény egészen más lenne: 10, 12, 14, 500, 502 és 504. Mindössze egyetlen távoli keresés és összességében is eggyel kevesebb kérelem. Ezzel a módszerrel a B-K-ütemező több B-K-kérelem számára teszi elérhetővé a lemezforgalom erőforrását, maximalizálva a teljes átvitelt. Minthogy a B-K-átvitel meglehetősen fontos a rendszer teljesítménye szempontjából és mivel a keresések ilyen iszonyatosan lassúak, a B-K-ütemező munkája nagyon fontos.

***Az eredmények***

| B-K-ütemező és rendszermag | 1. teszt | 2. teszt |
|---|---|---|
| Linus lift 2.4 alatt | 45 s | 30 perc, 28 s |
| Határidős B-K-ütemező 2.6 alatt | 40 s | 3 perc, 30 s |
| Jósló B-K-ütemező 2.6 alatt | 4,6 s | 15 s |

***2. próba:*** A nagy olvasási lappangás hatásai

Indítsunk folyamatos olvasást a háttérben:

```
while true
do
        cat big-file > /dev/null
done
```

Mindeközben mérjük meg, hogy mennyi ideig tart beolvasni a rendszermagfa valamennyi állományát:
`time find . -type f -exec cat ´{}´ ´;´ > /dev/null`
Így kipróbálhatjuk, hogyan viselkedik egyetlen nagy folyamatos olvasás alatt végrehajtott sok apró függő olvasás.

## A Linus-lift

A 2.4-es Linux-rendszermagban található B-K-ütemezőt Linus liftnek (Linus Elevator) nevezik. A B-K-ütemezőket gyakran liftalgoritmusoknak nevezik, mivel hasonló feladatokkal birkóznak meg, mintha egy nagy épületben folyamatosan működtetni szeretnénk a liftet. A Linus-lift megoldás csaknem pontosan a fent leírt klasszikus B-K-ütemezőmodellt követi. A legtöbb szempontból ez kiválóan megfelelt, hiszen az egyszerűség jó dolog és a 2.4-es rendszermag B-K-ütemezője valóban jól működött. Sajnos a B-K-ütemező teljes B-K-átvitel maximalizálását célzó küldetése során engedményt kellett tenni, az ár pedig a méltányosság sérülése lett (különösen kérelemlappangás – latency – esetén). Lássunk egy példát!
Vegyük a logikai lemez blokk-kérelmeinek a következő sorozatát: 20, 30, 700 és 25. A B-K-ütemező rendező algoritmusa a következő sorrendbe rendezné és adná ki a kérelmeket (feltételezve, hogy a fej jelenleg a lemez logikai kezdőpontjában áll): 20, 25, 30 és 700. Ez az, amire számítottunk, eddig nincs is gond. Tegyük fel azonban, hogy a 25-ös blokk kiszolgálása közben egy újabb kérés érkezik a lemez azonos részére. Azután egy újabb. Majd egy még újabb. Igencsak valószínű, hogy a várólista 700-as blokkra hivatkozó kérelme egy jó darabig nem lesz kiszolgálva.
De ez még nem minden. Mi történik, ha a kérelem egy lemezblokk beolvasása volt? Az olvasási kérelmek általában összehangoltak. Amikor az alkalmazás lemezírási utasítást ad ki, általában megáll és várakozik, amíg a rendszermag a kért adatot vissza nem adja. Az alkalmazásnak üldögélnie kell és várakozás közben malmozhat, míg a 700-as blokk kérelmére végre válasz érkezik. Az írás ezzel szemben általában nem összehangolt, hanem aszinkron módon működ-

dik. Amikor az alkalmazás kiad egy írási műveletet, a rendszermag az írandó adatot és a metaadatot bemásolja a rendszermagba, felkészít egy vermet az adat számára és visszalép az alkalmazáshoz. Az alkalmazást nem érdekli és igazából nem is kell tudnia, hogy az adata ténylegesen mikor kerül a lemezre.

Ettől viszont olvasási kérelem barátunk helyzete csak tovább romlik. Mivel az írás aszinkron, általában folyamként viselkedik, azaz gyakori a nagy mennyiségű adat egy idejű visszaírása. Ennek következtében rengeteg egyedi írási kérelem kerül a merevlemez közeli területeire. Példaképpen képzeljük el egy nagy állomány mentését: az alkalmazás írási műveleteket küldözget a rendszernek, a merevlemez pedig olyan gyors, amilyen gyorsra ütemezzük.

Az olvasási kérelmek ezzel szemben általában nem folyamként viselkednek. Az alkalmazások az olvasási kérelmüket apró egyenkénti darabokban adják ki, ahol is minden darabka az utolsótól függ. Gondoljunk csak egy könyvtár bejegyzéseire: az alkalmazás megnyitja az első állományt, kiadja az olvasási kérelmet az állomány megfelelő szakaszára, megvárja a visszaadott adatot, kiadja a következő darab olvasási igényét, vár, majd így folytatja mindaddig, amíg az egész állományt be nem olvassa. Ezután az állományt bezárja, megnyitja a következőt – és a folyamat kezdődik elölről. Minden soron következő kérelemnek meg kell várnia az előzőt, ami igen komoly késedelmet okozhat alkalmazásunk számára, ha a kérelmek túlságosan messzi lemezblokkokban találhatók.

Az a jelenség, amikor a folyamot alkotó írási kérelmek koplalásra ítélik a függő olvasási kérelmeket, „írás miatt koplal az olvasás” (writes-starving-reads) néven vált ismertté (lásd a széljegyzetet: „1. próba: Írás miatt koplal az olvasás”). Annak a lehetőségét, ha egy kérelmet elfogadható időn belül nem szolgálnak ki, éhezésnek nevezzük. A kérelmek éheztetése igazságtalansághoz vezet. A B-K-ütemezők esetében a rendszer a nagyobb átlagos átviteli teljesítmény érdekében egyértelműen feladta az igazságosságot. Azaz a rendszer megpróbálja növelni a rendszer összesített teljesítményét, bármely egyedi B-K-kérelem lehetséges feláldozása árán is. Ez elfogadott és tulajdonképpen kívánatos is, eltekintve attól, hogy a hosszúra nyúló éhezés nem valami jó dolog. Az olvasási kérelmek közepes ideig tartó éheztetése is nagy lappangási időhöz vezet, ha az alkalmazások más lemezműveletek közben próbálnak olvasási kérelmeket kiadni. Ez a nagy lappangás hátrányosan érinti a rendszer teljesítményét és kényelemérzetét (lásd a széljegyzetet: „2. próba: a magas olvasási lappangás hatásai”).

## A határidős B-K-ütemező

Az új, 2.6-os B-K-ütemezők fő célja pontosan az ilyen éhezés elkerülése volt általában a kérelmek, különös tekintettel az olvasási kérelmek esetében.

A 2.4-es B-K-ütemező és általában véve a hagyományos liftalgoritmusok körüli nehézségek kezelésére jelent meg a határidős B-K-ütemező. Mint azt bemutattuk, a Linus-lift a függőben lévő B-K-kérelmek listáját egyetlen sorban tárolja. A soron következő kiszolgált kérelem a sor élén álló B-K-kérelem lesz. A határidős B-K-ütemező megtartja ezt a sort, de egy kicsit megfűszerezi a dolgokat két további sor – az olvasási FIFO és az írási FIFO sor – bevezetésével. A határidős B-K-ütemező mindkét sorban küldési idő szerint rendezve tárolja az elemeket (azaz lényegében az elsőként belépő távozik elsőként, ezt jelenti egyébként a FIFO rövidítés: first in, first out – a ford.). Az olvasási FIFO sor, mint a neve is sugallja, kizárólag olvasási kérelmeket tárol. Az írási FIFO sor hasonlóképpen kizárólag az írási műveletekkel foglalkozik. Minden FIFO sorhoz tartozik egy lejárati érték: az olvasási FIFO sor lejárati ideje 500 milliszekundum, az írási FIFO soré pedig öt másodperc.

Amikor új B-K-kérelmet kapunk, beszúrva a sorba rendezzük, valamint a megfelelő (olvasási vagy írási) FIFO sor végére helyezzük. Normál esetben a merevlemeznek a rendezett, hagyományos sor tetejéről küldjük a kérelmeket, ezáltal maximalizáljuk az átvitelt és minimalizáljuk a kereséseket, hiszen akárcsak a Linus-lift esetében, a normál lista blokkszám szerint rendezett.

Amikor az egyik FIFO sor elején álló elem idősebb lesz, mint a sorhoz rendelt lejárati idő, a B-K-ütemező felfüggeszti a normál sor B-K-kérelmeinek a továbbítását. Ehelyett a FIFO sor elején álló kérelmeket fogja elküldeni, az egyenletesség kedvéért még néhány további elemmel is megtoldva. A B-K-ütemezőnek csak a FIFO sorok elején álló bejegyzéseket kell ellenőriznie, hiszen mindig ezek lesznek a legöregebb elemek a sorban.

Emlékszünk még öreg barátunkra, a 700-as kért blokkra? A távoli blokkokat célzó írási kérelmek áradata ellenére 500 milliszekundum elteltével a határidős B-K-ütemező felfüggesztené az ottani kérelmek kiszolgálását és elküldené a 700-as blokk olvasási kérelmét. A lemez megkeresné a 700-as blokkot, kiszolgálná az olvasási kérelmet, majd folytatná a további várakozó kérelmek feldolgozását.

Ilyen módon a határidős B-K-ütemező lágy határidőt vezetett be a B-K-kérelmekre. Igaz, semmi sem biztosítja, hogy a B-K-kérelmek a lejárati idő előtt ki lesznek szolgálva, de a B-K-ütemező általában a lejárati időhöz közeli tartományon belül szolgálja ki a kérelmeket. Ennek megfelelően a B-K-ütemező továbbra is jó, teljes körű átvitelt szolgáltat, de mindeközben egyetlen kérelmet sem éheztet elfogadhatatlanul hosszú ideig. Mivel az olvasási kérelmekhez gyors lejárati időt adtunk meg, az „írás miatt koplal az olvasás” nehézségeit a lehető legkisebbre csökkentettük.

## Jósló B-K-ütemező

Ez mind szép és jó, de még mindig nem a tökéletes megoldás. Gondoljuk végig, mi történik a képzeletbeli 700-as blokkunkkal, amely – tegyük fel – a lemezterületen számos egymástól függő olvasásnak az első tagja. Az olvasási kérelem kiszolgálása után a határidős B-K-ütemező a korábbi blokkokban folytatja az előzőleg elkezdett írási kérelmek kiszolgálását. Ez rendben is van egészen addig, amíg az olvasást végző alkalmazás a következő olvasási kérelmét el nem küldi (mondjuk a 710-es blokkra). 500 milliszekundumon belül ez a kérelem is lejár és a lemez a 710-es blokkhoz szalad, kiszolgálja a kérelmet, majd visszamegy oda, ahol az előbb állt és folytatja az írási kérelmek folyamának a kiszolgálását. Azután érkezik egy újabb olvasási kérés.

A feladat ismét azok miatt az átkozott függő írások miatt keletkezik! Mivel az olvasást függőben lévő darabok alapján végezzük, az alkalmazás csak akkor adja ki a következő olvasást, miután a korábbit már visszakapta. Csakhogy miközben az alkalmazás megkapja az adatot, a saját futási

üteméhez ér és elküldi a következő olvasást, a B-K-ütemező viszont már továbblépett és egy másik kérelmet szolgál ki. Az eredmény minden olvasáskor néhány felesleges keresés: kikeressük az olvasás helyét, kiszolgáljuk és visszamegyünk. Mennyivel jobb lenne, ha valami módja akadnak annak, hogy a B-K-ütemező megtudja – azaz inkább megjósolja –, hogy érkezik-e újabb olvasás a lemez azonos szakaszára. Előre-hátrakeresgélés helyett egyszerűen csak előrelátóan várna a következő olvasásra. A rút keresések kihagyása biztosan megér néhány milliszekundum várakozást és két keresést is kihagyhatunk.

Természetesen pontosan ezt teszi a jósló B-K-ütemező. Éppen úgy indul, mint a határidős B-K-ütemező: ugyanazt a határidő alapú rendszert használja, de ráadásképpen jósképességekkel is rendelkezik. Amikor az olvasási kérelem megérkezett, a jósló B-K-ütemező, akárcsak a korábbi, határidőn belül kiszolgálja. A határidős B-K-ütemezővel ellentétben azonban a jósló B-K-ütemező ezután csak üldögél és várakozik, semmit sem csinál egészen hat milliszekundumig. Igen jó esély van rá, hogy ez alatt a hat milliszekundum alatt az alkalmazás a fájlrendszer ugyanezen részére ad ki újabb olvasási kérelmet. Ha ez bekövetkezik, a kérelmet azonnal kiszolgáljuk, és a jósló B-K-ütemező még vár egy kicsit. Amennyiben a hat milliszekundum olvasási kérelem nélkül telik el, a jósló B-K-ütemező nem jól választott, és visszatér ahhoz, amit eddig csinált.

Mégha csak mérsékelt számú kérelmet sikerül is helyesen megjósolni, jelentős időt – darabonként két drága keresést – tudunk megtakarítani (lásd *táblázatunkat*). Minthogy a legtöbb olvasás függő, a jóslás a legtöbb esetben kifizetődő. Hogy a helyes jóslás esélyeit tovább növelje, a jósló B-K-ütemező heurisztikus módszereket alkalmaz, annak megállapítására melyik folyamatnak kell várnia. Ennek kivitelezéséhez az ütemező minden folyamathoz B-K-statisztikát vezet, nyomon követve annak viselkedését.

A szükségtelen keresések kiiktatásával és az olvasások gyorsabb kiszolgálásával a jósló B-K-ütemező egyszerre ad kisebb kérelemlappangási időt és nagyobb globális átvitelt a határidős B-K-ütemezőhöz vagy a Linus-lifthez képest. Nem meglepő, hogy a 2.6-os rendszermagban a jósló B-K-ütemező lett az alapértelmezett. Nagyon helyesen.

### Köszönetnyilvánítás

A Linus-lift fő szerzője ***Andrea Arcangeli*** és ***Jens Axboe***. A határidős B-K-ütemező elsősorban Jens Axboe munkája, a jósló B-K-ütemező megszületése ***Nick Piggin*** érdeme.

*Linux Journal 2004. február, 118. 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.