Nullmásolás felhasználói szemszögből

Lássuk, mit takar a nullmásolás fogalma, továbbá mire és hol lehet használni.

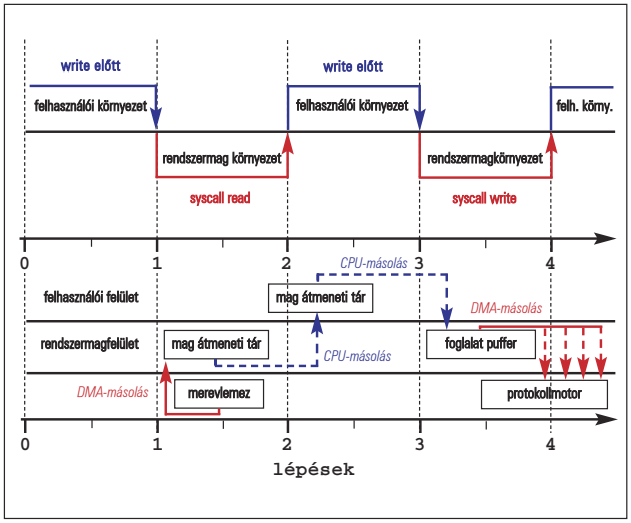
Szinte mindenki hallott már a Linux nullmásolási (zero copy) lehetőségéről, mégis sok emberrel találkozom, akik nem teljesen értik a lényegét. Ezért határoztam úgy, hogy egy cikksorozatban kicsit mélyebbre ások e témakörben, abban a reményben, hogy sikerül eloszlatnom a vele kapcsolatos bizonytalanságot. Jelen írásomban felhasználói szemszögből vizsgálom a nullmásolást, tehát a rendszermag élveboncolását szándékosan mellőztem.

Mi az a nullmásolás?

Ha egy kérdésre meg szeretnénk találni a választ, először magát a kérdést kell pontosan megértenünk. Vizsgáljuk meg, mire van szükség ahhoz, hogy egy egyszerű hálózati démon a fájlokban tárolt adatokat az ügyfeleknek átadhassa. Példakódunk:

```
read(file, tmp_buf, len);
write(socket, tmp_buf, len);
```

Elég egyszerűnek tűnik, nyilván úgy gondolod, hogy két egyszerű rendszerhívás végrehajtása nem igényel túl sok munkát. Sajnos a valóság messze nem így fest. Bár csak két hívást hajtottunk végre, az adatok másolása legalább négyszer megtörténik, és legalább ennyi felhasználói- és rendszermagkörnyezet váltásra is sor kerül. (Valójában a folyamat ennél is összetettebb, de nem akarok elmerülni a részletekben.) Az *1. ábrára* tekintve egy kicsit jobban is átláthatod a történéseket. Felül láthatók a környezetváltások, alul pedig a másolási műveletek. Első lépés: a rendszerhívás hatására a rendszer felhasználóiból rendszermagmódba vált. Az első másolást a DMA-motor végzi el, ami beolvassa a lemezről a fájl tartalmát, és egy a rendszermag címterében található átmeneti tárban tárolja.
A második lépés során a rendszer felhasználói átmeneti tárba másolja az adatokat a rendszermagéból, és visszatér az olvasási hívásból. Most, hogy az adatok bekerültek egy a felhasználói címtérben található átmeneti tárba, megkezdhetik lefelé tartó útjukat.
Harmadik lépésben az írási hívás hatására a rendszer felhasználóiból rendszermagkörnyezetbe vált. A harmadik másolás során az adatok újra egy a rendszermag címterében található átmeneti tárba kerülnek. Természetesen itt egy másik, kifejezetten a foglalatokhoz tartozó átmeneti tárról van szó.
A negyedik lépés az írási hívás visszatérése, ezzel megtörténik a negyedik környezetváltás. Ettől független és aszinkron módon egy negyedik másolás is lezajlik, amikor a rendszermag átmeneti tárában található adatokat a DMA-motor átadja a protokollmotornak. Talán felmerül benned a kérdés: „Mi az, hogy független és aszinkron módon? Hát nem történt meg az adatok átvitele még a hívás visszatérése előtt?" Az, hogy maga a hívás visszatér, önmagában még nem biztosítja a másolás tényleges elvégzését, sőt még annak megkezdését sem. Egyszerűen csak annyit jelez, hogy az ethernet illesztőprogram várólistájában voltak szabad leírók, és amit adtunk neki, azt elfogadta átvitelre váró adatként. Lehet, hogy jó néhány csomag van még a várólistában a miénk előtt. Hacsak maga az eszköz vagy az illesztőprogram nem állít fel fontossági várólistákat, az adatok küldése érkezési sorrend szerint történik. (Az *1. ábra* `fork` hívással párosuló DMA-másolása is jelzi, hogy az utolsó másolás késlekedhet.)

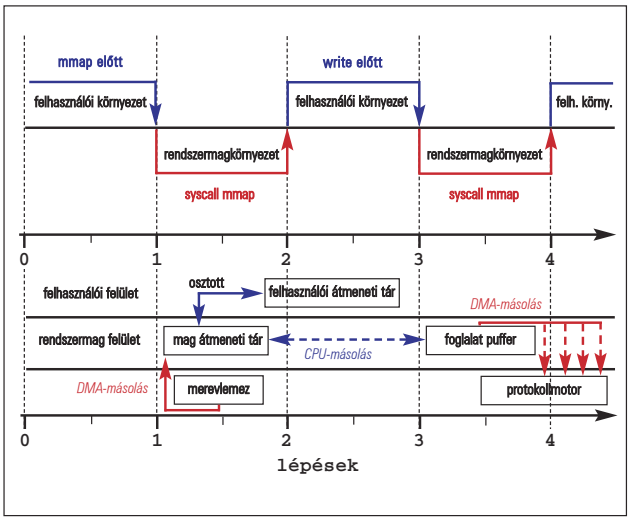


1. ábra Másolás két rendszerhívással

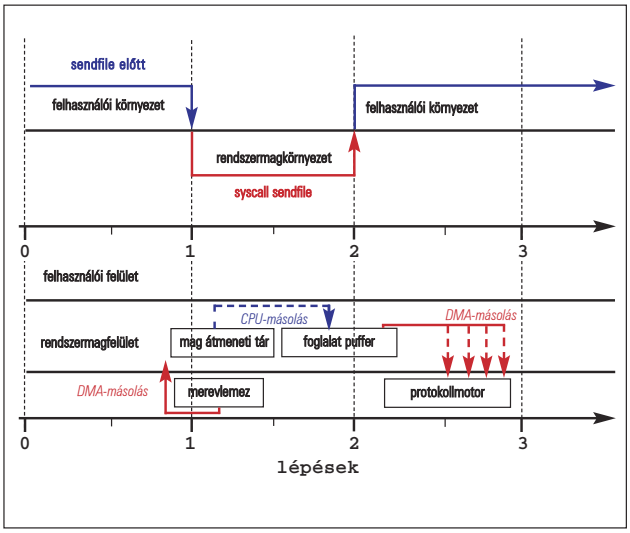
Mint láthatod, a cél eléréséhez egyáltalán nem lenne szükség ennyi adatmásolásra. Ha a másolgatások számát sikerülne csökkenteni, egyben a rendszer terhelése is kisebb lenne, így növekedne a teljesítménye. Illesztőprogram-fejlesztőként közvetlenül a vassal dolgozom, és bizony találok érdekes lehetőségeket. Egyes eszközök képesek megkerülni a központi memóriát, és közvetlenül a másik eszköznek adni át az adatokat. Így teljesen szükségtelen másolatot készíteni a központi memóriába, meg úgy általában véve nekem tetszik a dolog, de sajnos nem minden eszköz képes erre. Arra is gondolni kell, hogy a lemezről beolvasott adatokat a hálózat igényeinek megfelelő formátumba át kell csomagolni – és máris kezdünk „bonyolódni". A többletterhelés csökkentése érdekében először próbáljuk meg kiküszöbölni a rendszermag átmeneti tár és a felhasználói átmeneti tár közötti másolásokat.
Az egyik lehetőség a másolás elhagyására a `read` hívás mellőzése és az `mmap` használata. Például:

```
tmp_buf = mmap(file, len);
write(socket, tmp_buf, len);
```

A folyamatot a *2. ábra* szemlélteti. A környezetváltások változatlanok.
Elsőként az `mmap` rendszerhívás hatására a DMA-motor a fájl tartalmát egy rendszermag átmeneti tárba másolja. Ezután a rendszer az átmeneti tárat megosztja a felhasználói folyamattal,



2. ábra Az mmap használata



3. ábra A read és a write hívások kiváltása sendfile hívással

a felhasználó és a rendszermag memóriaterülete között nem végez másolást.
Másodikként a `write` rendszerhívás hatására a rendszermag az adatokat a foglalatokhoz rendelt átmeneti tárba másolja.
A harmadik másolás akkor történik, amikor a DMA-motor az adatokat átadja a rendszermagfoglalat átmeneti tárból a protokollmotornak.
Azzal, hogy `read` helyett `mmap` hívást használunk, felére csökkentettük a rendszermag által másolandó adatok mennyiségét. Már ezzel is számottevő eredményt érhetünk el, ha nagy mennyiségű adatról van szó. Csakhogy megfizetjük az árát is, az `mmap` és `write` használatának bizonyos buktatói is vannak. Az egyik lehetséges hiba az, amikor memória-hozzárendelést hozol létre egy fájlhoz, majd `write` hívást hajtasz végre, de közben egy másik folyamat ugyanezt az állományt csonkolja. Az írási műveletet egy `SIGBUS` sínhibajelzés fogja megszakítani, ugyanis hibás memóriaeléréssel próbálkoztál. Alapesetben a hibajelzés a vétkes folyamat kilövését okozza és memóriakiíratással jár, ezt viszont hálózati kiszolgálóknál általában igyekszünk elkerülni. A gondot kétféleképpen oldhatjuk meg.

Az első lehetőség egy jelkezelő (signal handler) telepítése a `SIGBUS` jelzéshez, majd egyszerű visszatérést alkalmazni a jelkezelőben. Így a `write` rendszerhívás a megszakítás előtt sikeresen kiírt bájtok számával tér vissza, az `errno` pedig sikert jelez. Hadd mutassak rá, miért rossz ez a megoldás: csak a tüneteket kezeli, de az okot nem szünteti meg. A `SIGBUS` azt tudatja velünk, hogy az adott folyamat kapcsán valami nagyon rosszul sült el, így ettől a módszertől inkább tartózkodnék.
A másik megoldás a fájl kibérlése (file leasing) a rendszermagtól, amit a Windows világában alkalmi zárolásnak is neveznek. A bérlést végrehajtva a fájlleírón a fájlt ideiglenesen bérletbe veheted. Ezt követően olvasási, illetve írási bérlést kérsz a rendszermagtól, így amikor a másik folyamat az éppen küldött fájlt megpróbálja csonkolni, a rendszermag egy valós idejű `RT_SIGNAL_LEASE` jelzést küld neked. Így tudomásodra juthat, hogy a rendszermag megszünteti a fájlra szóló ideiglenes bérletedet. Írási hívásod még azelőtt megszakad, hogy a programod érvénytelen címhez férne hozzá, és a `SIGBUS` hibajelzés miatt idő előtt elhalálozna. A `write` hívás a megszakítás előtt sikeresen kiírt bájtok számát adja vissza, az `errno` pedig sikert jelez. Az alábbi példa szemlélteti, hogyan lehet fájlt bérelni a rendszermagtól:

```
if(fcntl(fd, F_SETSIG, RT_SIGNAL_LEASE) == -1 {
        perror("rendszermag bØrlØs
                ➥beÆll tÆsa");
        return -1;
}

/* l_type ØrtØke F_RDLCK vagy F_WRLCK
   lehet */
if(fcntl(fd, F_SETLEASE, l_type)){
         ➥bØrlØst pus beÆll tÆsa");
        return -1;
}
```

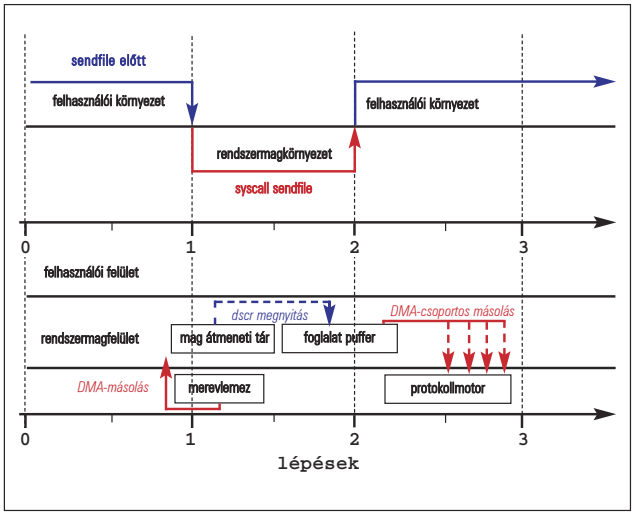
A bérlést az `mmap` használata előtt kell végrehajtanod, és csak miután végeztél, szabad megszüntetned. Utóbbit az `fcntl F_SETLEASE` hívásával érheted el, a bérlés típusát `F_UNLCK` értékre állítva.

Sendfile

A 2.1-es változatú rendszermagban jelent meg a hálózati vagy két helyi fájl közti adatátviteleket leegyszerűsítő `sendfile` rendszerhívás. A `sendfile` révén nemcsak a másolások, hanem a környezetváltások száma is csökkenthető. Használata a következő:

```
sendfile(socket, file, len);
```

A folyamatot a *3. ábra* szemlélteti.
Először a `sendfile` rendszerhívás hatására a DMA-motor a fájl tartalmát egy rendszermag átmeneti tárba másolja. Ezután az adatokat a rendszermag a foglalatokhoz rendelt átmeneti tárba másolja.
Másodszor: a harmadik másolás akkor zajlik le, amikor a DMA-motor a rendszermagfoglalatokhoz rendelt átmeneti tárában található adatokat átadja a protokollmotornak.
Nyilván eszedbe jut, hogy vajon mi történik, ha egy másik folyamat csonkolja az éppen `sendfile` rendszerhívással továbbított állományt? Ha nem jegyzünk be jelkezelőt, a `sendfile` egyszerűen a megszakítás előtt sikeresen átvitt bájtok számával visszatér, és az `errno` is sikert jelez.



4. ábra A gyűjtögetést támogató eszköz képes arra, hogy több memóriarészből olvassa ki az adatokat, így eggyel kevesebb másolásra van szükség

Ha még a `sendfile` meghívása előtt kibéreled a fájlt a rendszermagtól, változatlan viselkedést és visszatérési értéket tapasztalsz. Megkapod a már ismert `RT_SIGNAL_LEASE` jelzést is, mielőtt még a `sendfile` visszatérne.
Az eddigiek alapján sikerült elkerülnünk, hogy a rendszermag nagyszámú másolást végezzen, de egy másolás még hátravan. Vajon ezt is el lehet hagyni? Természetesen igen, ha a vas is nyújt hozzá némi segítséget. Ha el szeretnénk felejteni a rendszermag által végzett adatkettőzéseket, akkor az adatgyűjtést támogató hálózati csatolóra van szükségünk. Ez mindössze annyit jelent, hogy a továbbításra várakozó adatoknak nem muszáj összefüggő memóriaterületen lenniük, hanem több helyre is szétszórhatók. A 2.4-es rendszermagban a foglalat átmenetitár-leíróját úgy módosították, hogy teljesíti ezeket a követelményeket – Linux alatt ezt nevezzük nullmásolásnak. Az elgondolás révén nemcsak a többszörös környezetváltás, de a processzor által végzett adatkettőzések is elhagyhatók. A felhasználói alkalmazások szemszögéből nézve semmi sem változott, a kód továbbra is így alakul:

```
sendfile(socket, file, len);
```

A lezajló folyamatot a *4. ábra* szemlélteti.
Első lépésként a `sendfile` rendszerhívás hatására a DMA-motor a fájl tartalmát egy rendszermag átmeneti tárba másolja. Második lépésben az adatok másolása helyett a rendszer csak a helyüket és a méretüket megadó leírókat fűzi hozzá a foglalat átmeneti tárhoz. A DMA-motor a rendszermag átmeneti tárból közvetlenül a protokollmotornak adja át az adatokat, így az utolsó másolás is elmarad.
Mivel az adatokat a lemezről be kell olvasni a memóriába, majd ki kell küldeni a hálózatra, néhányan talán nehezményezik, hogy nem igazi nullmásolást végzünk. Ez részben így is van, de az operációs rendszer szempontjából elértük a célunkat, ugyanis az adatokat már nem többszörözzük rendszermag átmeneti tárak között. Nullmásolás használatakor nemcsak a kevesebb másolás által javítunk a teljesítményen, hanem kevesebb környezetváltásra van szükség, nem kavarjuk össze a processzor gyorsítótárát, és ellenőrző összegeket sem kell számítani.
Most, hogy tisztában vagyunk a nullmásolás lényegével, a gyakorlatban is alkalmazzuk elméleti tudásunkat. A teljes forráskódot a ➲ http://www.xalien.org/articles/source/sfl-src.tgz címen vagy a 45. CD Magazin/Zero_Copy könytárában érheted el. Kicsomagolni úgy tudod, hogy kiadod a `tar --zxvf sfl-src.tgz` parancsot. A `make` paranccsal egyrészt lefordíthatod a kódot, másrészt egy véletlenszerű tartalommal feltöltött, ***data.bin*** nevű adatállományt is létrehozhatsz.
Lássuk a kódot, a fejlécfájlokkal kezdve:

```
/* sfl.c sendfile pØldaprogram
Dragan Stancevic visitor@xalien.org 09-09-2002
fejlØc neve                    f ggvØny/vÆltoz
```

További érdekességek

Forráskód és használat

A teljes forráskódot a 45. CD Magazin/Zero_Copy könyvtárában találod. A `tar --zxvf sfl-src.tgz` paranccsal bonthatod ki, majd a `make` paranccsal hozhatod létre a futtatható állományt. A könyvtárban egy `sfl` nevű futtatható fájlnak kell létrejönnie, átadott értékek nélkül indítva a program rövid használati útmutatót jelenít meg. Mind a kiszolgáló-, mind az ügyféloldalon meg kell adnod a kiszolgáló IP-címét.

Példa:
Küldő oldal:
visitor@xalien-saucer:~/sfl-src> ./sfl s 10.0.0.2
Server binding to [10.0.0.2]
Server sent 10240 bytes.
visitor@xalien-saucer:~/sfl-src>

Fogadó oldal:
visitor@earth:~/sfl-src> ./sfl r 10.0.0.2
Client connecting to [10.0.0.2]
Client received 10240 bytes.
visitor@earth:~/sfl-src>

Megjegyzések a kódhoz

1. Én az 1033-mas kaput használtam, ha te a saját rendszereden ezt más célra használod, válassz másikat.
2. A példaprogram egyszerűsítése és rövidítése érdekében a 10 KB-os átmeneti tárat a verembe raktam. Saját kódodban a `malloc` függvénnyel foglalhatsz átmeneti tárakat.
3. A cikkben szereplő program lerövidítése érdekében elhagytam a `close` rendszerhívás visszatérési értékének vizsgálatát. Egy valódi programban természetesen el kell végezned az ellenőrzést.

Példák

Ha mélyebben is meg szeretnél ismerkedni a `sendfile` rendszerhívás használatával, pillants bele néhány olyan alkalmazásba, amelyik használja:
Apache ➲ http://www.apache.org
Samba ➲ http://www.samba.org
Mozilla ➲ http://www.mozilla.org
Pure-FTPd ➲ http://pureftpd.sf.net
Ha érdekel a rendszermag fájlbérlési szolgáltatása, nézz bele a Samba forráskódjába, az *smbd/oplock_linux.c* fájlba.

```
#include <stdio.h>           /* printf,
                                perror */
#include <fcntl.h>           /* open */
#include <unistd.h>          /* close */
#include <errno.h>           /* errno */
#include <string.h>          /* memset */
#include <sys/socket.h>      /* socket */
#include <netinet/in.h>      /* sockaddr_in */
#include <sys/sendfile.h>    /* sendfile */
#include <arpa/inet.h>       /* inet_addr */
#define BUFF_SIZE (10*1024)  /* a tmp Ætmeneti
                                tÆr mØrete */
```

Az alapvető foglalatműveletekhez szükséges `<sys/socket.h>` és `<netinet/in.h>` mellett a `sendfile` rendszerhívás törzstípusát is meg kell adnunk. Ezt a `<sys/sendfile.h>` `server` jelzőjében találjuk:

```
/* k ld nk vagy fogadunk */
if(argv[1][0] == 's') is_server++;

/* le r k megnyitÆsa */
sd = socket(PF_INET, SOCK_STREAM, 0);
if(is_server) fd = open("data.bin", O_RDONLY);
```

Ugyanaz a program kiszolgálóként (küldőként) és ügyfélként (fogadóként) is használható. Ellenőrizni kell a megfelelő parancssori átadott értéket, majd – szükség szerint – az `is_server` jelzőt beállítva küldő módba válthatunk. Az `INET` protokollcsalád egy folyamfoglalatát is megnyitjuk. Mivel a program most kiszolgálóként üzemel, adatokat kell küldenie az ügyfeleknek, ezért az adatállományt is megnyitjuk. Az adatok küldésére a `sendfile` rendszerhívást használjuk, így nincs szükség a fájl tartalmának beolvasására és a saját programunk átmeneti tárában történő tárolására. A kiszolgáló címe:

```
/* a mem ria t rlØse */
memset(&sa, 0, sizeof(struct sockaddr_in));

/* az adatszerkezet kezdı ØrtØkadÆsa */
sa.sin_family = PF_INET;
sa.sin_port = htons(1033);
sa.sin_addr.s_addr = inet_addr(argv[2]);
```

Töröljük a kiszolgáló címét tároló adatszerkezet tartalmát, majd adjuk meg a protokollcsaládot, a kaput és a kiszolgáló IP-címét; az utóbbit a program átadott értékként kapja meg. A kapu bedrótozott módon az egyébként használaton kívüli 1033-mas lesz. Azért ez, mert e kaputartomány használatához az adott rendszeren már nincs szükség rendszergazdai jogosultságra.
A kiszolgálói ág:

```
if(is_server){
      int client;  /* œj  gyfØlfoglalat */
      printf("A kiszolgÆl  a [%s] kaput
            ➥hasznÆlja.\n", argv[2]);

      if(bind(sd, (struct sockaddr *)&sa,
            ➥sizeof(sa)) < 0){
            perror("bind");
            exit(errno);
      }
```

Kiszolgálóként címet kell rendelnünk a foglalat leírójához. Ezt a `bind` rendszerhívással tehetjük meg, amely a foglalat leírójához (sd) egy kiszolgálócímet rendel (sa):

```
if(listen(sd,1) < 0){
      perror("listen");
      exit(errno);
}
```

Mivel folyamfoglalatot használunk, valahogy ki kell nyilvánítanunk kapcsolatok fogadására irányuló szándékunkat, és meg kell adnunk a kapcsolati várólista méretét is. A kapcsolati várólista (backlog queue) hosszát 1-re állítottam, de a már felépült kapcsolatok fogadása érdekében ennél nagyobb értéket is meg szoktak adni. A rendszermag régebbi változatainál a kapcsolati várólista segítségével előzték meg a syn-elárasztásos támadásokat. Mivel a `listen` rendszerhívás időközben módosult, és csak a felépült kapcsolatok beállításait adja meg, a kapcsolati várólistára többé nincs szükség; pontosabban a rendszermag `tcp_max_syn_backlog` beállítása gondoskodik arról, hogy megelőzze a rendszer ellen irányuló syn-elárasztásos támadásokat:

```
if((client = accept(sd, NULL, NULL)) < 0{
      perror("accept h vÆs");
      exit(errno);
}
```

Az `accept` rendszerhívás a függőben lévő kapcsolatok várólistájának első kérését feldolgozva új csatlakoztatott foglalatot hoz létre. A hívás visszatérési értéke az új kapcsolat leírója. Ezzel a foglalat alkalmassá vált a `read`, `write`, `poll` és `select` rendszerhívások használatára:

```
if((cnt = sendfile(client, fd, &off,
BUFF_SIZE)) < 0) {
      perror("sendfile h vÆs");
      exit(errno);
}
print("A kiszolgÆl  %d bÆjtot k ld tt el.\n",
      ➥cnt);
close(client);
```

A foglalatleíró használatával létrejött egy kapcsolat, tehát megkezdhetjük az adatok továbbítását a távoli rendszerre. Erre a `sendfile` rendszerhívást használjuk, amelynek törzstípusa Linux alatt a következő:

```
extern ssize_t
sendfile (int __out_fd, int __in_fd, off_t
          ➥*offset, size_t __count) __THROW;
```

Az első két átadott érték fájlleíró. A harmadik az eltolás, ez adja meg, hogy a `sendfile` honnan kezdje meg az adatok küldését. A negyedik a küldeni kívánt bájtok száma. Ahhoz, hogy a `sendfile` nullmásolással továbbítsa az adatokat, a hálózati csatolónak támogatnia kell az adatgyűjtést. Azoknál a protokolloknál, amelyek ellenőrző összegeket számítanak – ilyen többek közt a TCP és az UDP –, az ellenőrző összegeket is ki kell tudnod számítani. Szerencsére akkor sem kell lemondanod a `sendfile` használatáról, ha a hálózati csatolód régi, és nem támogatja ezeket a szolgál-

tatásokat. A különbség mindössze annyi, hogy a rendszermag ekkor küldés előtt összerendezi az átmeneti tárakat.

Hordozhatósági kérdések

A `sendfile` rendszerhívással általános esetben az a baj, hogy – az `open` hívással ellentétben – nincs szabványos megvalósítása. A Linux, Solaris és HP-UX alatti megvalósítások kismértékben eltérnek egymástól, és ez gondot jelenthet a hálózati alkalmazásaikban a nullmásolási lehetőséget kihasználni kívánó fejlesztőknek.

Az egyik eltérés a megvalósítások között az, hogy a linuxos két fájlleíró között teszi lehetővé az adatátvitelt, azaz fájl–fájl és fájl–foglalat párosításokat is kezel, míg a HP-UX és a Solaris megvalósítás csak fájl–foglalat átvitelekhez használható.

A másik különbség az, hogy Linux alatt nem lehet vektoros átviteleket végezni. A Solaris és a HP-UX `sendfile` megvalósítása külön átadott értékek használatával szükségtelenné teszi az átvivendő adatokhoz csatolt fejlécek miatt elvégzendő többletmunkát.

Előre tekintve

A Linux alatti nullmásolás megvalósítása még nem fejeződött be, és a közeljövőben valószínűleg módosulni fog a szolgáltatás működése – várhatóan újabb lehetőségekkel bővül. Például a `sendfile` nem támogatja a vektoros átviteleket, és a kiszolgálók – mint az Apache vagy a Samba – több, `TCP_CORK` jelzővel jelölt `sendfile` hívást kénytelenek végrehajtani. Ez a jelző arról tudósítja a rendszert, hogy újabb `sendfile` hívásokkal további adatokat akarunk küldeni. A `TCP_CORK` azonban nem fér össze a `TCP_NODELAY` jelzővel, illetve akkor is használjuk, ha az adatokhoz fejlécet akarunk csatolni.

Nagyszerű példája ez annak, amikor a vektoros átvitellel több `sendfile` hívást ki lehetne váltani, illetve a jelenlegi megvalósítással járó késleltetéseket el lehetne hagyni.

Ugyancsak kellemetlen megkötés, hogy a jelenlegi `sendfile` csak 2 GB-nál kisebb fájlok továbbítására használható. Manapság egyáltalán nem szokatlanok az ekkora állományok, viszont ekkora adatmennyiséget megkettőzni küldés közben több mint kellemetlen. Mivel ilyen esetben az `mmap` és a `sendfile` egyaránt használhatatlanok, egy `sendfile64` megvalósítás igencsak jól jönne az újabb rendszermagváltozatokban.

Összegzés

Korlátai ellenére a nullmásolás hasznos szolgáltatás, és remélem, írásomban elegendő adatot találtál ahhoz, hogy te is elkezdd használni a saját programjaidban. Ha érdekel a téma, olvasd el a hamarosan megjelenő második részt, amelyben a rendszermag szempontjából járom körül a nullmásolás kérdéskörét.

Linux Journal 2003. január, 103. szám

Dragan Stancevic

Húszas éveinek végén járó rendszermag- és eszközfejlesztő. Szakmája szerint programmérnök, de az alkalmazott fizika is érdekli, szabadidejében különlegesen magas elektromos feszültségekkel szeret kísérletezni.