Ha bemegyek...

A folyamatkezelés mellett a beviteli-kiviteli eszközök (például billentyűzet, monitor, merevlemez, nyomtató) kezelése is fontos része az operációs rendszernek. E feladat megvalósítása az úgynevezett beviteli-kiviteli programra hárul.

Minden programnak szüksége van arra, hogy adatokat kérhessen be, illetve futásának végeredményét valahol „kiadhassa”. Erre szolgálnak számítógépünk úgynevezett beviteli-kiviteli eszközei, amelyek masinánk nélkülözhetetlen részei.

Beviteli eszköznek számít például a billentyűzet vagy az egér, amellyel alkalmazásaink közvetlenül tőlünk, azaz a felhasználótól kérhetik be az adatokat. Ugyanakkor adatokhoz hozzá lehet jutni például egy állományból vagy egy adatbázisból is, tehát a különböző lemezes egységek is beviteli eszköznek számítanak. Kiviteli eszköz a monitor, a nyomtató, de természetesen egy lemezre is kiírhatunk adatokat, tehát a lemezek egyszerre beviteli és kiviteli eszközök.

Egy többfeladatos operációs rendszerben azonban nem engedhető meg, hogy bármelyik program akkor használhassa a beviteli-kiviteli (a továbbiakban B-K) eszközöket, amikor csak jólesik neki. Ha például két program egyszerre kezd nyomtatni, ennek eredményeként minden bizonnyal valamiféle összevisszaságot kapnánk. Ezért be kell osztani, hogy ki és mikor veheti igénybe az adott eszköz szolgáltatásait. Ez az operációs rendszer feladata.

A másik nehézség, hogy ahányféle B-K eszköz létezik, annyiféleképpen kell programozni. A felhasználói alkalmazások fejlesztői nyilván nem azzal akarnak foglalkozni, hogy alkotásuk milyen színű és formájú gépen lesz futtatva, csupán azt szeretnék, hogy mindenhol fusson, ahol az adott operációs rendszert használják. Ezért szükséges egy olyan eszközfüggetlen környezet megteremtése, amely a felhasználók szeme elől az eszközök zord világát teljes egészében elfedi, és egy egyszerűen kezelhető felületet biztosít. Erről is az operációs rendszer képes gondoskodni.

Azt a részt, amely e két feladat megvalósításáért felelős, az operációs rendszer B-K programjának nevezzük. Sorozatunk következő részeiben ezzel foglalkozunk. Most még csak elméleti síkon, általánosan ismerkedünk meg a B-K eszközökkel, illetve a kezelésükért felelős programmal, a továbbiakban pedig az összes jelentős beviteli-kiviteli eszközön sorban végigmegyünk, a termináloktól kezdve egészen a lemezes meghajtókig.

Egy kis alkatrész...

Ha valamilyen módon csoportosítani szeretnénk a B-K eszközöket, a következő három csoportot állíthatnánk fel: blokkos eszközök, karakteres eszközök és minden egyéb, ami a két másik csoportba nem tartozik bele.

A legjellemzőbb blokkos eszköz a lemez (például merevlemez, hajlékonylemez stb.). Ezek az adatokat előre meghatározott méretű blokkokban tárolják. Minden blokk saját címmel rendelkezik és egymástól teljesen függetlenül olvasható-írható.

A karakteres eszközöktől azonban az adatokat karakterenként egymás után ömlesztve küldjük, illetve fogadjuk, anélkül, hogy bármiféle szerkezeti felépítéssel foglalkoznánk. Klasszikus karakteres eszköz a billentyűzet, a nyomtató, a hálózati csatoló, a modem, az egér és a sort még folytathatnánk.

Ez a csoportosítás nem tökéletes, ugyanis olyan eszközök is vannak, amelyek egyik csoportba sem tartoznak igazán. Nézzük például a szalagmeghajtókat, amelyek nem igazi blokkos eszközök. Esetleg vegyük számítógépünk valós idejű óráját, amelynek az a feladata, hogy meghatározott időközönként megszakítást hozzon létre. Ez is B-K eszköznek számít, mégsem sorolható egyik csoportba sem. Mindezek ellenére a B-K eszközök ilyen módon való csoportosítása jó alapot szolgáltat az operációs rendszer B-K programjának tervezéséhez.

Ezek után nézzük meg, hogyan is épül fel általában egy hagyományos B-K egység. Először is adott egy szerkezet, amely tulajdonképpen nem más, mint maga az eszköz. Ehhez egy elektronikus rész van kapcsolva , amely általában egy nyomtatott áramköri lap. Ez az úgynevezett eszközvezérlő, amelynek feladata a szerkezeti rész irányítása. A vezérlő a számítógép alaplapjához kapcsolódik és a processzorral az úgynevezett sínrendszer segítségével tartja a kapcsolatot.

A hagyományos személyi számítógépek általában az egysínű, más néven buszmodellt alkalmazzák. Ez azt jelenti, hogy a processzor, a memória és a B-K-vezérlők egy sínre vannak felfűzve. Ejtsünk pár szót a vezérlő és a szerkezeti rész kapcsolatáról is. Ez általában nagyon alacsony szintű, ugyanis az eszköz szinte kivétel nélkül mindig bitszervezésű. Ez azt jelenti, hogy a vezérlő a szerkezeti résztől egy bitsorozatot kap, amelyet majd át kell alakítania bájtokká (azaz 8 bites blokkokká). A vezérlő egy belső memóriával is rendelkezik, amelyet átmeneti tárnak (buffer) nevezünk. A fogadott biteket először itt tárolja, majd ha egy bájt összeállt, ellenőrzi annak épségét, és ha minden egybevág, akkor kerülhet tovább a főmemóriába.

A vezérlővel a kapcsolatot az úgynevezett regiszterek segítségével tarthatjuk. Egyes géptípusok esetében ezek a regiszterek a főmemória részét képezik, ezért például úgy adhatunk nekik értéket, hogy a memória egy meghatározott címtartományába írunk (az ilyeneket memóriában leképezett beviteli-kivitelnek nevezzük).

Ha a vezérlő végzett az előírt művelettel és az eredményt elhelyezte a regisztereiben, megszakítást hoz létre. A megszakításokról a folyamatok ütemezésénél már beszéltünk, csakhogy most ki kell egészítenünk néhány dologgal.

PC esetében az alaplapon általában két megszakításvezérlő található. Mindegyik 8-8 bemenettel rendelkezik, de az egyik közülük egymás összekapcsolására szolgál, így egy hagyományos PC-ben 15 megszakítás áll a B-K eszközök rendelkezésére. Ezeket a bemeneteket IRQ-nak (Interrupt ReQuest) nevezzük, mivel ha egy eszköz megszakítást szeretne előidézni, csak elektromos jelet kell küldenie az adott bemenetre. Hogy melyik eszköz melyik bemenethez (azaz IRQ-hoz) kapcsolódjon, azt általában saját maga is eldöntheti, viszont bizonyos eszközök (például a billentyűzet) esetében már előre

be vannak állítva, ezért nincsen lehetőség változtatásra.
Egy megszakításkérés esetében a CPU felfüggeszti azt, amit addig csinált, és az úgynevezett megszakítás-vektortáblázatból kikeresi az adott megszakításhoz tartozó memóriacímet. Ezen a memóriacímen található a megszakítást kiszolgáló eljárás, ami az operációs rendszer része. Ez az eljárás rögvest futni is kezd, és amíg a dolgát be nem fejezi, a processzor nem térhet vissza az eredeti tevékenységéhez.
Bonyolultan hangzik? Sebaj, nézzünk egy példát arra, miként zajlik egy blokk beolvasása a merevlemezről! Először beírjuk a vezérlő regisztereibe, hogy melyik blokkot szeretnénk kiolvasni a lemezről. Miután ez megtörtént, a processzornak nincs több feladata, ettől kezdve a vezérlőn múlik minden. Amíg az dolgozik, az adott folyamat blokkol, és a CPU egy másikat kezd el futtatni. Amikor a vezérlő végzett, megszakítást idéz elő, és az éppen végrehajtás alatt álló folyamat futása felfüggesztődik: ismét a rendszer veszi át az irányítást. Elindul a merevlemez kezeléséért felelős eljárás, amely a vezérlő regisztereiből kiolvassa a művelet végeredményét. Ha a beolvasás sikerült, az eljárás a blokk tartalmát a vezérlő átmeneti tárából bájtonként átmásolja a memóriába.
Mindez eddig rendben is lenne, azonban akad egy gond: a rendszer az adott blokkot kénytelen bájtonként átmásolni, ezzel pedig rengeteg processzoridő megy kárba. Ezért találták ki az úgynevezett közvetlen memóriaelérést, azaz a DMA-t (Direct Memory Access). Ha lehetőség nyílik a DMA használatára, a művelet megkezdése elején a vezérlőnek egy memóriacímet is átadunk, ahová az adott blokk tartalma kerülhet. A másolást majd a vezérlő végzi el, miután a blokk beolvasása megtörtént, és csak ezután hoz létre megszakítást, de a blokk tartalmának bájtonkénti átmásolásával már nem kell foglalkoznunk, mivel az már a memóriában van.
Meg kell jegyeznünk, hogy nem minden számítógép használ DMA-t, sőt, sok esetben egyáltalán nem hatékony, ha a „hagyományos" módszer helyett ezt használjuk. A DMA-vezérlő ugyanis gyakran sokkal lassabb, mint a processzor, és ha a CPU-nak épp nincs semmi más teendője, várakoznia kell, amíg a DMA-vezérlő befejezi a memóriába való másolást; gyorsabban végeznénk, ha ezt programból megvalósított módon tennénk.

A B-K program

A B-K program feladatait a cikk bevezetőjében már ismertettük, most nézzük, hogyan is épül fel. A folyamatkezelés mellett ez is nagyon fontos részét képezi az operációs rendszernek, a rendszermag kódjának nagy részét is ez teszi ki. A legfontosabb elv, amelyhez egy B-K program fejlesztése során ragaszkodnunk kell, az eszközfüggetlenség. Erről már sorozatunk első részében is meséltünk (Linuxvilág 18. szám, 53. oldal), de nem árt, ha szánunk némi helyet arra, hogy egy példa segítségével szemléltethessük, mit is értünk valójában eszközfüggetlenség alatt.
Alapgondolata a következő: az operációs rendszerünk alá fejlesztők szemszögéből teljesen mellékesnek kell lennie, hogy például a bemeneti adatokat tartalmazó állományt merevlemezről, CD-ről, esetleg hajlékonylemezről kívánjuk-e beolvasni. Esetleg az általunk írt MP3-lejátszó kódját ne kelljen csak azért módosítanunk, mert egy olyan gépen szeretnénk futtatni, amely a miénktől eltérő típusú hangkártyával bír. A lényeg az, hogy az operációs rendszer támogassa a használni kívánt eszközöket.
Egy kicsit általánosabban megfogalmazva: például egy `cat <bemenet >kimenet` formájú utasítást is kiadhassunk a parancsértelmezőnek. A `cat` voltaképpen egy program, ami nem tesz mást, mint a bemenetről érkező adatokat a kimenetre folyatja. A lényeg az, hogy a `cat` program szempontjából teljesen mindegy, hogy a bemenet állomány-e vagy valamilyen más fizikai eszköz, például maga a billentyűzet. Az is teljesen mellékes számára, hogy kimenetnek a monitort, a nyomtatót adjuk-e meg vagy egy másik állományt. A `cat` ugyanúgy képes lesz ellátni feladatát, függetlenül attól, mit adtunk meg bemeneti, illetve kimeneti eszköznek.
Az ilyen szintű eszközfüggetlenséget csak úgy tudjuk megvalósítani, ha a B-K programot rétegekbe szervezzük. A legalacsonyabb szinten fekvő réteg van a legközelebb a géphez, és mindegyik réteg egy kicsit többet rejt el a felsőbb szintek elől az eszközből, míg a legutolsó (felhasználói) réteg már csupán egy könnyen kezelhető kényelmes eszközfüggetlen környezetet lát. Ám még mielőtt részletesebben is bemutatnánk a rétegeket, meg kell oldanunk néhány fontos feladatot, például a hibakezelést. Ha egy B-K-művelet közben hibát észlelünk, nem mindegy, hogy melyik rétegben kezeljük. Előfordulhat például, hogy a hiba csak valamilyen ideiglenes állapot miatt következett be (például egy porszem került az olvasófejre), és a művelet megismétlésekor már megkaphatjuk a helyes eredményt. Ezért arra kell törekedni, hogy a hibát az eszközhöz minél közelebbi szinten kezeljük, és a felsőbb rétegek csak akkor szerezzenek tudomást róla, ha a kívánt feladat végrehajtása valóban lehetetlen.
Ha a Linuxot vesszük alapul, a B-K programot négy különálló rétegre oszthatjuk fel: a megszakításkezelőre, az eszközmeghajtókra, az eszközfüggetlen B-K-kódra és a felhasználói rétegre. Legalul a megszakításkezelő foglal helyet. Erről már volt szó a folyamatok ütemezésének tárgyalásakor, de most egy kicsit részletesebben is nézzük meg. Az operációs rendszerek általában arra törekednek, hogy a megszakításokkal foglalkozó eljárások a lehető legalacsonyabb szinten helyezkedjenek el. Amikor egy folyamat B-K-műveletet kezd, az önműködően blokkol. A B-K-művelet elvégzése után az eszköz megszakítást idéz elő, amelyet a megszakításkezelő fogad. Ennek feladata általában csak annyi, hogy felébressze a műveletet indító folyamatot, ami a Linux világában egy üzenet küldésével történik. A megszakításkezelő üzenetet küld a blokkolt folyamatnak, aminek hatására az ismét futásra kész lesz.
Az eszközmeghajtó az operációs rendszernek az a része, ami mindent tud a hozzárendelt eszközről, illetve annak programozásáról. Egyedül ez foglalkozik olyan dolgokkal, mint például a vezérlő regiszterei. Az eszközmeghajtó feladata a következő: az eszközfüggetlen felső rétegekből kapott utasításokat úgymond a vezérlő által emészthető formájúra alakítja át. Ezek az utasítások meglehetősen elvontak, merevlemez esetében például csak annyit kap, hogy olvassa be a 12. blokkot. Ám a merevlemez buta eszköz, csupán nagyon alacsony szintű műveletek elvégzésére képes. Ezért az eszközmeghajtónak az lesz a dolga, hogy megállapítsa, milyen merevlemez-műveleteket kell kiadni, hogy eleget tehessen ennek az elvont utasításnak. Például először ki kell számolnia a 12. blokk fizikai helyét a lemezen, be kell kapcsolnia a fej mozgatómotorját, az olvasófejet a megfelelő helyre kell pozícionálnia és így tovább.
A felsőbb rétegek erről természetesen mit sem tudnak, ők csak a kért blokk tartalmát fogják visszakapni, az összes többi az eszközmeghajtóra van bízva.
Az eszközmeghajtók valójában különálló folyamatok, amelyek a felhasználói szinten futó programokkal együtt ütemeződnek. Igaz, ezek a folyamatok jóval alacsonyabb szinten futnak, és sokkal többet tehetnek, például közvetlenül hozzáférnek az eszközvezérlők regisztereihez, ugyanis az eszközmeghajtók magában a rendszermagban helyezkednek el.

Egy eszközmeghajtó folyamat általában több, egymással szoros rokonságban álló eszközt vezérel. Például létezik egy olyan terminálmeghajtó, amely az összes terminál kezeléséért felelős, de akad olyan ethernetfolyamat is, amely gépünk összes ethernet típusú hálózati csatolójával foglalkozik.
Egy eszközmeghajtó két részből épül fel: egy erősen alkatrészfüggő és egy alkatrészfüggetlen részből. Az utóbbiba azok az eljárások tartoznak, amelyek az eszközmeghajtóhoz tartozó összes eszköznél azonosak. Az alkatrészfüggő részbe pedig csak az kerül, ami kifejezetten ahhoz a típusú eszközhöz használható. Nem kell tehát minden egyes eszközhöz külön eszközmeghajtót futtatni. Az eszközmeghajtók a többi folyamattal a sorozatunk előző részében (Linuxvilág 19. szám, 48. oldal) már ismertetett üzenetküldéses rendszer segítségével tartják a kapcsolatot.
A Linux tehát folyamatokra (és a folyamatok közötti üzenetküldésre) alapuló rendszer rendszer. Ezt a fajta felépítést még a Minixtől örökölte, amely kifejlesztésének az volt a célja, hogy egy jól áttekinthető Unix-alapú operációs rendszert hozzanak létre. Az áttekinthetőbb szerkezet érekében a Unix klasszikus tömbszerű (egy programban megvalósított) felépítését mellőzték, és egy talán jobban átlátható és érthetőbb modellre cserélték.
A folyamat alapú és a tömbszerű rendszer közti különbség a felhasználói és a rendszermagszint közötti kapcsolatban rejlik. Nézzünk például egy fájlból történő olvasást. Az első esetben a felhasználói program üzenetet küld a fájlrendszert kezelő folyamatnak, hogy ő ezt és ezt a fájlt szeretné beolvasni.
A fájlrendszer megnézi, hogy ehhez mely blokkokat kell beolvasni a merevlemezről, majd ő is küld egy üzenetet, amelyben az eszközmeghajtót megkéri az adott blokkok beolvasására. Míg az üzenetet küldő folyamat választ nem kap, addig blokkolt állapotba kerül.
A tömbszerű rendszerekben ilyen nem létezik. Az eszközmeghajtók itt tulajdonképpen eljárások, amelyek a rendszermag által lefoglalt memóriába vannak beágyazva. Ha egy felhasználói rétegben lévő folyamat akar valamit a rendszertől (például egy állomány beolvasását), akkor például egy megszakítás segítségével meghívja a megfelelő magszintű eljárást. Na de itt nem blokkol a folyamat! Ugyanaz a folyamat fut tovább, csak nem felhasználói, hanem rendszermagszinten. Ha a hívott eljárás elvégezte a feladatát, a folyamat visszatér a felhasználói rétegbe. Ne búsuljunk, ha ez egy kicsit zavaros. Egyet jegyezzünk meg: a Linux abban különbözik a legtöbb Unix-alapú rendszertől, hogy folyamatokra alapuló renszer. Ez azzal jár, hogy az egész könnyebben kezelhető kisebb részekből áll, ami az osztott (több gépből álló) rendszerek esetében nagyon jól jön. Akad azonban egy hátránya is: a tömbszerű rendszerek megoldása gyorsabb, mivel egy eljárást kevesebb ideig tart meghívni, mint folyamatok között üzeneteket küldözgetni.
E kitérő után kanyarodjunk vissza eredeti témánkhoz, és vegyük szemügyre a következő réteget, amelyben a B-K program alkatrészfüggetlen része található. Itt olyan tevékenységek zajlanak, amelyeknél lényegtelen, hogy az adott eszköz milyen típusú. Erre a legjellemzőbb példa a fájlrendszert kezelő folyamat. A fájlrendszer felépítése nem változik meg attól, hogy adatainkat valójában milyen lemezen tároljuk, ezzel csak az eszközmeghajtónak kell foglalkoznia.
Milyen feladatokat is kell az eszközfüggetlen B-K-rétegnek ellátnia? Az egyik legfontosabb, hogy minden eszközhöz hozzárendeljen valamilyen nevet, amellyel a felhasználók hivatkozhatnak rájuk. Mint tudjuk, ez Unixban eszközfájlok segítségével történik, amelyek a */dev* könyvtár alatt találhatóak (például a */dev/ttyS0* az első soros kapunkat, vagy a */dev/hda*, amely az elsődleges IDE-merevlemezünket azonosítja). Az eszközfájlok egyébként két számot tartalmaznak, a fő, illetve a másodlagos eszközszámot, amelyek az adott eszközt pontosan azonosítják. Rendszerüknek nemcsak elneveznie, de védenie is kell az eszközöket. Egyrészt meg kell akadályoznia, hogy két folyamat egyszerre ugyanazt az eszközt használja, másrészt azt is figyelembe kell vennie, hogy az adott felhasználó egyáltalán jogosult-e arra, hogy ilyesmit tegyen.
A különböző blokkos eszközök sajnos nem egységes blokkméretekkel dolgoznak. Például nem minden lemeznek ugyanakkora a szektormérete. Ahhoz azonban, hogy fájlokat tároljunk egy fájlrendszerben, elengedhetetlen egy egységes blokkméret alkalmazása. Ezt is itt kell megoldanunk.
A másik feladat az átmeneti tárazás. Előfordulhat, hogy például a felhasználó gyorsabban gépel, mint ahogy a bemenetet fogadó folyamat azt fogadni tudja. Ezért fent kell tartani egy belső tárat, ahová a begépelt karakterek kerülnek, és majd innen kerülnek feldolgozásra a folyamathoz. Aki használt MS-DOS-t, biztos emlékszik rá, hogy miközben a gép dolgozott, és ő nagy lendülettel püfölte a billentyűket, a 16. billentyű lenyomása után a gép vadul sípolni kezdett. Ez azt jelentette, hogy megtelt a billentyűzet átmeneti tára, és addig tele is marad, amíg a begépelteket valaki fel nem dolgozta.
A blokkos eszközök is igényelnek átmenetitár-használatot. Például mi történne akkor, ha egy állomány tartalmát bájtonként szeretnénk beolvasni? Lemezről csak blokkonként olvashatunk. Célszerű tehát az egész blokkot beolvasni az átmeneti tárba, majd kérésre onnan egyenként adagolni a bájtokat.
Vannak még mások is, amik szintén az eszközfüggetlen részben foglalnak helyet, de róluk inkább később, a fájlrendszer tárgyalásakor beszélnénk.
Most már csak az utolsó, a felhasználói szint maradt hátra. Ide a B-K programnak csak nagyon kis része tartozik, az is leginkább könyvtári eljárások formájában. A felhasználói programok ugyanis ezeket az eljárásokat hívogatva végezhetnek B-K-műveleteket, illetve hívhatnak rendszerhívásokat. A könyvtári eljárások tulajdonképpen semmi mást nem tesznek, mint hogy a megfelelő értékeket elhelyezik a rendszerhívásnak megfelelő helyre.
Vannak azonban olyan B-K-eszközök, amelyeket egy másik felhasználói szintű program segítségével használhatunk. Az egyik legjellemzőbb példa erre a nyomtatás. A rendszerben fut egy közönséges felhasználói program, a nyomtatódémon, amelynek az a feladata, hogy a nyomtatást vezérelje. Ha egy folyamat ki szeretne nyomtatni egy állományt, ahelyett, hogy közvetlenül a nyomtatóra küldené, egy meghatározott könyvtárba teszi. Ezután a nyomtatódémon majd eldönti, hogy mikor kerülhet sor az adott állomány kinyomtatására. Ezt a módszert egyébként háttértárolásnak (sorbaállításnak) nevezzük.
Ez a megoldás azért is jó, mert meggátolhatjuk, hogy egy folyamat feleslegesen tartsa fel a nyomtatót (például a nyomtatás késleltetésével), aminek következtében esetleg gátolna más, a nyomtatóra váró folyamatokat.
A háttértárolás másik gyakori felhasználási területe egyébként a hálózaton való állomány-, illetve levéltovábbítás. Ha levelet szeretnénk küldeni, csak elhelyezzük egy megadott könyvtárban, és a levélküldésért felelős démon majd gondoskodik a célba juttatásáról.

Garzó András
(garzoand@interware.hu) körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség. Kedvenc étele a palacsinta, és van egy Richard nevű macskája. Minden észrevételt, megjegyzést, levelet szívesen fogad.