Az operációs rendszerek belügyei

Hogyan is működik egy mai korszerű operációs rendszer? Miként osztja meg gépünk erőforrásait a párhuzamosan futó alkalmazások között? Miként gazdálkodik a rendelkezésre álló memóriával, és hogyan tárolja adatainkat a merevlemezen?

Most induló sorozatunk az ehhez hasonló kérdésekre keresi a választ, leginkább csak elméleti síkon, de a gyakorlatban is bemutatjuk: vizsgálódásunk alanyául az egyik legelterjedtebb nyílt forráskódú rendszert, a Linux-rendszermagot választottuk.

Még mielőtt buzgón a mély vízbe vetnénk magunkat, tisztáznunk kell néhány alapvető fogalmat, mivel pontos ismeretük rendkívül fontos az operációs rendszerek lelkivilágának mélyebb megismeréséhez. Elképzelhető, hogy sorozatunk első részének mondanivalójával már sok olvasó tisztában van, de mi azt szeretnénk, hogy minél szélesebb olvasói réteg vehesse hasznát írásunknak. Ezért a hangsúlyt nem a tömörségre és teljességre, hanem a könnyebb megértésre próbáljuk helyezni. Teljességre egyébként is hiú ábránd lenne törekedni, sorozatunk témája ugyanis rendkívül bonyolult és összetett, vaskos könyvek és tanulmányok láttak róla napvilágot. Mi elsősorban e szövevényes téma azon szeleteivel foglalkozunk, amelyek egy átlagos felhasználót érdekelhetnek.

Az itt bemutatott elvek gyakorlatban történő megvalósítása rettenetesen nehéz és összetett feladat, részletes taglalására újságunk keretein belül nem is lenne mód, ezért megelégszünk annyival, ha elméleti síkon be tudjuk mutatni egy mai korszerű operációs rendszer működését. Akit a téma ennél is mélyebben érdekel, javasoljuk, lapozgassa az ajánlott olvasmányként feltüntetett műveket.

Akkor kezdjük az elején! Mi a feladata tulajdonképpen az operációs rendszernek? E kérdés megválaszolása nem is olyan egyszerű. Az operációs rendszernek alapvetően két különböző, ám egymással szoros kapcsolatban lévő feladatnak kell eleget tennie: az ***erőforrás-kezelésnek***, és egy olyan ***egységes programozási felület megteremtésének***, amelyen keresztül a felhasználói alkalmazások erőforrásainkat könnyebben használhatják. Nézzük meg, mit is takarnak ezek a kifejezések!

Az operációs rendszer egyrészt erőforrás-kezelő. Erőforrás alatt azokat az eszközöket (például háttértárolókat, nyomtatókat, egereket), illetve adategységeket (például egy állományt vagy egy adatbázis meghatározott rekordját) értjük, amelyet ***egy időben csak egy program érhet el***. Hogy egy időben valóban csak egy alkalmazás férhessen az adott erőforráshoz, azt az operációs rendszer biztosítja.

Ennek könnyebb megemésztése végett nézzünk egy hétköznapi esetet: ki szeretnénk nyomtani egy állomány tartalmát. Többfeladatos rendszerekben (például Linux, Windows, OS/2) bőségesen előfordulhatnak olyan helyzetek, hogy két párhuzamosan futó alkalmazás egyszerre akar egy erőforrással dolgozni, jelen esetben nyomtatni. Ha az erőforrásokhoz bárki bármikor korlátlanul hozzáférhetne, könnyen adódhatnak nagy galibák. Például míg mi buzgón állományunk tartalmát nyomtatjuk, addig valaki egy másik terminálról szintén nyomtatási parancsot adhat ki. Eredményül nagy összevisszaságot kapunk, amelyen mindkét egyszerre kinyomtatott állomány tartalmát megtalálhatjuk, egymással összeolvadva.

Könnyű belátni, mennyire kulcsfontosságú, hogy az erőforrások használata az operációs rendszer által szabályozva legyen. Ez a szabályozás a gyakorlatban úgy valósul meg, hogy a különböző erőforrások közvetlen kezeléséről maga az operációs rendszer gondoskodik. Ha ki szeretnénk nyomtatni egy állományt, meg kell kérnünk a rendszert, hogy tegye meg nekünk (egy átlagos alkalmazás a legtöbb rendszerben a párhuzamos kapuhoz nem is férhet hozzá közvetlenül). Például Unix-alapú rendszerekben ezt úgy tehetjük meg, hogy a kinyomtatni kívánt állományt elhelyezzük egy erre a célra kijelölt könyvtárban. Egy másik, vele együtt futó különleges alkalmazás – a nyomtatódémon – folyamatosan ellenőrzi e könyvtár tartalmát, és ha talál benne nyomtatásra szánt anyagot, annak tartalmát elküldi a párhuzamos kapura, majd munkája befejeztével az állományt kitörli. Ha időközben újabb kinyomtatásra váró anyag érkezik, egészen addig nem foglalkozik vele, míg előző feladatát nem teljesítette.

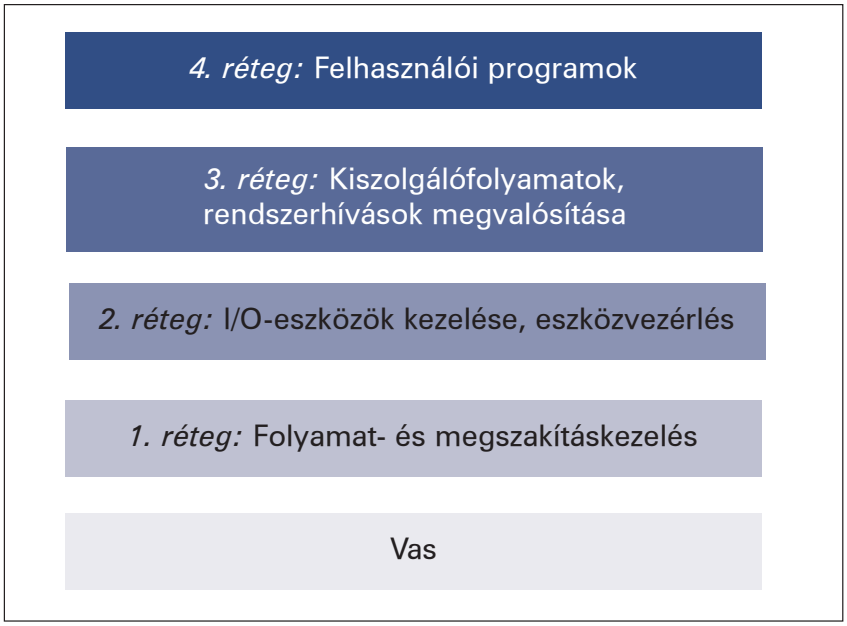
A többfeladatos operációs rendszerek lehetővé teszik, hogy több alkalmazást is futtathassunk párhuzamosan. Azt a legkisebb egységet, amely párhuzamos feldolgozásra kerülhet, ***folyamatnak*** (process) nevezzük. Tulajdonképpen minden elindított alkalmazás egy-egy folyamatnak felel meg a rendszerben.

Mint tudjuk, a processzorok soros feldolgozásúak (az utasításokat egymás után egyesével hajtják végre), azaz egy időben mindig csak egy folyamat futhat. Az operációs rendszernek tehát nemcsak az erőforrásokat kell beosztania a programok között, hanem azt is, hogy melyik folyamat mikor és mennyi ideig futhat. Amikor ez az idő letelik, a futási lehetőséget egy másik folyamat kapja meg. Szakszerűen mondva az operációs rendszernek a folyamatok ***ütemezéséről*** is gondoskodnia kell.

Jogosan merül fel a kérdés, hogy mi a helyzet a manapság egyre jobban terjedő többprocesszoros rendszerek, illetve a géptelepek (több gépből összeállított szupergéptek) esetében? Itt az operációs rendszernek egy sokkal bonyolultabb ütemező algoritmussal kell rendelkeznie, de több processzor hatékonyabb kihasználásához a felhasználói programok felőli támogatásra is szükség lesz.

Mit értünk ez alatt? Logikusnak tűnhet, hogy ahány processzort tömünk számítógépünk belsejébe, a feldolgozás annyiszor gyorsabb lesz. Ez az elképzelés azonban téves, ugyanis sebességnövekedést csak több alkalmazás párhuzamos futtatásakor észlelhetünk. Ám a hálózati kiszolgálóktól eltekintve a tapasztalat azt mutatja, hogy a felhasználók általában csak egy folyamat feldolgozásán szeretnének gyorsítani. A megoldás kézenfekvő: az adott folyamatot egyszerre több processzoron futtatjuk. Ám ezt csak akkor tehetjük meg, ha az alkalmazást eleve erre felkészülve írták meg, azaz párhuzamosították. A párhuzamos

feldolgozásra szánt programok írása egy kissé más szemléletet igényel, mint a hagyományos sor programoké. Nem ritka, hogy egy alkalmazást a párhuzamosítás érdekében szinte teljesen át kell írni. Sőt, bizonyos esetekben csupán kis százalékban érünk el sebességnövekedést, tehát programunkat minden olyan alkatrészre külön meg kell írnunk, amelyen futtatni szeretnénk.
Érthető, hogy aki felhasználói alkalmazások fejlesztésére adta a fejét, nem kíván ilyen mélységben megismerkedni az alkatrész programozás rejtelmeivel.



A Linux-rendszer felépítése

ugyanis amennyi időtöbbletet nyerünk azáltal, hogy több számítást egyszerre tudunk végezni, ugyanannyit kell áldoznunk a több processzoron való futtatás összehangolására.
De térjünk vissza eredeti témánkhoz! Az operációs rendszerek másik fő feladatköre következik: az erőforrások elérésének megkönnyítése. A számítógép alkatrészeinek programozása sosem volt kényelmes feladat. A felhasználói alkalmazások által gyakran használt legalapvetőbb műveletek (például egy fájlból való olvasás) megvalósítása is komoly kihívást jelent. Például nem mondhatjuk meg a merevlemeznek, hogy az adott fájlból olvassa ki az első száz bájtot. Mi csak szektorokat olvastathatunk be vele, de még ehhez a legegyszerűbb művelethez is rengeteg adat ismerete szükséges, mint például a pályánkénti szektorok száma, vagy hogy hány darab cilinder található a merevlemezen. Sőt, olyasmire is figyelni kell, hogy mennyi időt vesz igénybe az olvasófej állítása, illetve a motor felpörgetése (ezt az időt ugyanis programunknak várakozással kell töltenie). Nem is beszélve arról, hogy a különböző cégek által gyártott eszközöket az esetek többségében nem azonos módon kell programoznunk, Csak annyit szeretne, hogy a fájlból való olvasást egy egyszerű eljárás meghívásával elvégezhesse, és értékként csak a fájl nevét, illetve a beolvasandó bájtok számát kelljen átadnia. Az, hogy az állomány milyen módon, illetve milyen típusú háttértárolón van tárolva, a programozót a legkevésbé sem érdekli.
Ezért az operációs rendszer számítógépünk szolgáltatásait a felhasználói programok számára kibővíti. Eme bonyodalmas műveletek – mint például a fájlkezelés – megvalósítását az operációs rendszer magára vállalja, és olyan felületet hoz létre, amely a felhasználó szeme elől „eltakarja” az alkatrész programozás sötét világát. Ha úgy tetszik, egy olyan virtuális számítógépet kapunk, amelynek programozása lényegesen egyszerűbb.
Azok a szolgáltatások, amelyek a programozók számára leegyszerűsítik az erőforrások használatát, ***rendszerhívások*** formájában valósulnak meg. A rendszerhívások tulajdonképpen olyan eljárások, amelyeket a felhasználói alkalmazások bármikor szabadon meghívhatnak. Hogy miképpen az, gép- és rendszerfüggő. Például MS-DOS-ban a hexadeximális 21-es megszakítást kellett meghívnunk (a megszakításokról később bővebben beszélni fogunk), de az értékeket előtte meghatározott regiszterekben kellett tárolnunk. Unix-alapú rendszerek alatt a rendszerhívások azonban C könyvtárak formájában is elérhetőek, így a C nyelvben írt programokból közvetlenül meghívhatók.
Meg kell jegyeznünk, hogy az operációs rendszerek többsége csak rendszerhívásokon keresztül engedélyezi az erőforrások elérését, közvetlenül nem. Ugyanis csak így tarthatja meg állandó befolyását számítógépünk felett, tehát kizárólag így szavatolhatja, hogy egy erőforrást egyszerre mindig csak egy program használ, illetve ellenőrizheti, hogy egy művelet végrehajtásához a futó alkalmazásnak megvan-e a kellő jogosultsága. Ezek együttesen biztosítják, hogy semmilyen felhasználó által indított program nem lesz képes a rendszerünket összeomlasztani.
A POSIX-szabvány (amely azt írja le, hogy egy Unix-alapú rendszernek milyen feltételeknek kell megfelelnie) 53 rendszerhívást határoz meg. Ezek olyan alapvető műveleteket valósítanak meg, mint folyamatok és jelek kezelése, fájl-, könyvtár-, illetve fájlrendszerkezelés, valamint a dátum- és az idő kezelése. A Linux ezeknél természetesen sokkal több rendszerhívást tartalmaz, amelyek például a további perifériák kezelését vagy a hálózati szolgáltatásokat valósítják meg.
Mielőtt továbbmennénk, tisztáznunk kell, hogy a gépünkön található programrengetegből melyek alkotják közvetlenül operációs rendszerünk részeit, és melyek nem. Azok a programok, amelyek úgynevezett ***felhasználói módban*** futnak, semmiképp sem tartozékai az operációs rendszernek. Nemcsak a különböző felhasználói alkalmazások (például szövegszerkesztők) sorolhatók ebbe a csoportba, hanem például az úgynevezett héjprogramok is. Ezek olyan alkalmazások, amelyek a felhasználó és az operációs rendszer közötti kapcsolattartást biztosítják, segítségükkel adhatunk utasításokat a rendszernek. Ide tartoznak a parancsértelmezők (például a bash, a csh), de tulajdonképpen az ablakkezelők is. Hiába fontos tehát egy rendszer életében a héjprogram, mégsem közvetlen tartozéka az operációs rendszernek.
A rendszerprogram azon részeit hívjuk operációs rendszernek, amelyek úgynevezett rendszermag-, más néven felügyelt módban futnak. Ezek sokkal több mindent megtehetnek, mint a felhasználói szinten futtatott alkalmazások, például közvetlen beviteli/kiviteli műveleteket hajthatnak végre, illetve további

kedvezményeket is élvezhetnek (például nagyobb elsőbbséget).
Fontos megjegyezni, hogy a magmódban elhelyezkedő programokhoz és az alkatrészekhez felhasználók közvetlenül nem férhetnek hozzá. Míg tehát a felhasználó saját maga választhatja meg például azt, hogy melyik héj alatt szeretne dolgozni, vagy hogy melyik levelezőprogram alatt óhajtja a leveleit olvasgatni, addig például a billentyűzet kezeléséért felelős eljárásokat nem cserélheti csak úgy le.
Most nézzük meg egy mai korszerű operációs rendszernek a felépítését, a Linuxét. A Linux belső felépítését vizsgálva négy eltérő, ám jól meghatározott feladatot ellátó réteget különböztethetünk meg. A legalsó rétegnek két feladata van: a folyamatok ütemezése és a közöttük zajló kapcsolattartás lehetővé tétele. Az utóbbit IPC-nek (InterProcess Communication – folyamatközti kapcsolattartás) hívjuk, és a későbbiekben nagyon fontos szerepe lesz. A legalsó réteg felelős az eszközmegszakítások fogadásáért. Sorozatunk következő részében e réteg működésével bővebben is foglalkozunk.
A második réteg legfontosabb feladata az erőforrás-kezelés megvalósítása. Ebben a rétegben helyezkednek el az eszközmeghajtók (device drivers). Ezek már folyamatok, azaz a felhasználói programokkal párhuzamosan futó eljárások, de a hardveres egységekhez közvetlenül hozzáférhetnek, és az ütemező is „méltányosabban” bánik velük. Az első és a második réteg együtt alkotja a kernelt, közismertebb nevén a rendszermagot.
A harmadik rétegben a rendszerhívások végrehajtásáért felelős úgynevezett ***kiszolgálófolyamatok*** helyezkednek el, amelyek a felhasználói programoknak nyújtanak hasznos szolgáltatásokat. Külön kiszolgálófolyamat fut például a memóriakezeléssel vagy a fájlrendszerrel kapcsolatos rendszerhívások kezelésére. Fontos megjegyezni, hogy a kiszolgálók már teljesen önálló folyamatok, és a hatáskörük is erősen korlátozott (például nem hajthatnak végre közvetlen I/O-műveleteket), mégis megkülönböztetett helyzetben vannak a felhasználói folyamatokkal szemben. Továbbá a kiszolgálófolyamatok előbb indulnak el, mint bármelyik felhasználói folyamat, és egészen addig futni fognak, amíg rendszerünket le nem állítjuk. Ezért a kiszolgálófolyamatok nem felhasználói módban futnak, de már nem is közvetlenül a rendszermag részei.
A felhasználó szemszögéből nézve ez érdektelen, mivel a kiszolgálófolyamatok forrását a Linux-mag forrása tartalmazza, és fordításkor az is a rendszermaggal együtt fordul.
Az utolsó szinten a felhasználói alkalmazások foglalnak helyet, azaz itt fut a szövegszerkesztő, a héj és a különböző egyedi felhasználói folyamatok, a démonok.
A démonok az itt bemutatott kiszolgálófolyamatokkal sok hasonlóságot mutatnak, mivel a rendszer indításától általában ők is egészen a leállításig futnak, de érdemi tevékenységet csak egy meghatározott esemény bekövetkeztekor végeznek (például egy időpont eljövetelekor vagy egy hálózati kapura való kapcsolódási kérelem esetében). Ha ez megtörténik, azonnal munkába állnak, és elvégzik a kijelölt feladatot, majd ismét „álomba szenderülnek”. Ugyanakkor a démon csak egy egyszerű felhasználói folyamat, ugyanolyan megkötések vonatkoznak rá, mint például szövegszerkesztőnkre, ebben az értelemben nem tekinthető az operációs rendszer részének. Ám a Unix világában a démonoknak mégis fontos szerep jutott, ugyanis velük hajtatnak végre minden olyan műveletet, amelyet egy felhasználói szintű program is elvégezhet.
A különböző hálózati kiszolgálók (például a web-, az FTP-, a levélkiszolgálók) is démon formájában futnak.
Ne bánkódjunk, ha a GNU/Linux felépítése még mindig homályos számunkra. A következő résztől kezdve sorban végigmegyünk az összes rétegen, és részletesen, példákkal szemléltetve mutatjuk be feladatukat és működésük lényegét.
Ennyi volt, amit feltétlenül tudnunk kell ahhoz, hogy mélyebben belevessük magunkat az operációs rendszerek működésének tanulmányozásába. A legközelebbi alkalommal már érdekesebb témákkal foglalkozunk: először megismerkedünk a Linux-rendszermag forrásának felépítésével, majd megnézzük, hogy miként kezeli a folyamatokat, illetve hogyan teszi lehetővé a közöttük való kapcsolattartást, az IPC-t.

Ajánlott irodalom

Andrew S. Tannenbaum – Albert S. Woodhull: Operációs rendszerek (Panem, 1999.)
➲ http://mirrors.kernel.org/LDP/LDP/tlk/tlk.html

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.