A jó időzítés titkai

Amikor egy felhasználót arra kérünk, hogy soroljon fel minél több olyan eszközt, amellyel bevitel–kivitel hajtható végre, az esetek nagy százalékában biztosan el fog feledkezni a számítógép órájáról.

Első hallásra valóban furán hangzik, hogy az órát is B-K eszköznek kell tekintenünk, pedig látni fogjuk: valóban az, és nem is akármilyen! Nélkülözhetetlen feladatokat bíz rá ugyanis az operációs rendszer.

Az első kérdés, amit érdemes lenne megvizsgálni, hogy tulajdonképpen mit is értünk a számítógép órája alatt. Az elnevezés megtévesztő lehet, ugyanis a számítógépekben található órák egyáltalán nem hasonlítanak az éjjeliszekrényünkön található vekkerünkhöz, de még a Rolexünkkel sincsenek semmiféle rokonságban. Ezek az órák ugyanis olyan eszközök, amelyeknek az a feladatuk, hogy bizonyos időközönként megszakítást hajtsanak végre.

A felhasználó magával az órával általában csak akkor találkozik (és akkor is csak közvetett módon), amikor lepillant a képernyő sarkába, ahol ablakkezelőnk lelkesen mutatja a pontos időt.

Az operációs rendszer számára azonban elengedhetetlenebb szolgáltatásokat nyújt. Vegyük például azt az esetet, amikor egy folyamat semmilyen beviteli–kiviteli műveletet nem hajt végre, csak számolgat magában a végtelenségig. Ilyenkor az ütemezőnek esélye sem lenne megakadályozni, hogy az ilyen folyamatok teljes egészében kisajátítsák maguknak a processzort (ugyanis B-K művelet hiányában nem keletkezik megszakítás, lásd a Linuxvilág 2002. augusztusi számát, 48–51. oldal).

Ehhez hasonló esetekkor válnak hasznossá az óra által létrehozott megszakítások (amelyek előbb vagy utóbb biztosan bekövetkeznek, függetlenül attól, hogy van-e valamilyen B-K művelet folyamatban), így az ütemezőnek lehetősége nyílik felfüggeszteni az adott folyamat futását. Az órát még sok minden másra is felhasználja az operációs rendszer, hogy mire, arra még később visszatérünk.

Érdekes, hogy az órát is a beviteli-kiviteli eszközök közé soroljuk, miközben semmiben sem hasonlít például a lemezekhez vagy a terminálokhoz. Sőt még arról sem beszélhetünk, hogy az óra karakteres vagy blokkeszköz-e. Ám egy másik nézőpontból tekintve egyértelműen B-K eszköz. A felhasználó szemszögéből nézve az óra nem a hétköznapi értelemben vett B-K eszköz, mivel nem látja, nem foghatja kézbe, hiszen egybe van építve az alaplappal. Most gondoljuk át az egészet még egyszer! Mit csinál az óra? Megszakításokat hoz létre. Akkor szükség lesz egy olyan eljárásra, amelyik válaszol ezekre a megszakításokra, azaz egy órakezelőre. Mivel ez is megszakításokat kezel, mint például a lemezkezelők, neki az összes többi B-K eszközkezelővel egy szinten kell futnia. Akkor tehát az operációs rendszer szempontjából az óra is egyszerű B-K eszköz. Igaz, se nem blokkos, se nem karakteres, mégis ugyanúgy kell vele foglalkozni, mint a többi egységgel.

A számítógépekben általában kétféle órát alkalmaznak. Az első közvetlenül a táphoz kapcsolódik, és minden egyes fázisváltáskor megszakítást okoz (ez viszonylag gyakori esemény, egy másodpercben körülbelül ötvenszer-hatvanszor következik be). A másik órában egy kristály oszcillátor van, ami nem más, mint egy megfelelő formára hozott és nyomás alá helyezett kvarckristály. Ennek köszönhetően kihasználhatjuk a kvarckristály jellétrehozó tulajdonságait, ezek ugyanis nagyon megbízhatóak. Az órához tartozik egy számláló is, és minden, a kvarckristály által keltett jelre csökkenti ennek a számlálónak az értékét. Amikor a számláló eléri a nullát, létrejön a megszakítás. A két óra között az érdembeli különbség az, hogy az elsőnél nem határozhatjuk meg a megszakítás rezgésszámát (frekvenciáját, azaz az órát nem programozhatjuk), míg ennél megmondhatjuk, hogy a megszakítások milyen időközönként történjenek. Az ilyen órák másik jellemző tulajdonsága, hogy általában több egymástól független programozható órát tartalmaznak.

Ezenkívül találhatunk még egy órát, amire akkor van szükség, amikor a gépet kikapcsoljuk (ezt gyakran háttérórának is nevezzük). Ez gondoskodik ugyanis arról, hogy a pillanatnyi dátum és idő ne felejtődjék el. Ezek a karórákhoz hasonlóan elemmel működnek és nagyon keveset fogyasztanak. (Ma már azonban egyre divatosabbá válik, hogy a pontos időt is az Internetről „töltsük le”. Egy különleges protokoll segítségével megoldható két számítógép órájának az összehangolása.)

A legtöbb operációs rendszer tehát az órát is beviteli–kiviteli eszközként tartja számon, így az órakezelő is a többi eszközmeghajtóval együtt fut. Maga az eszközkezelés rendkívül egyszerű, csupán a létrejövő megszakításokat kell kezelni.

Az órakezelőnek azonban rengeteg más feladata is van, nézzük, melyek ezek!

A pontos idő

Kezdjük a legegyszerűbbel, a pontos idő (vagy valós idő) karbantartásával, ami tulajdonképpen nem jelent mást, mint azt, hogy számoljuk az óramegszakításokat. A Unix-rendszerek a pontos időt úgy tárolják, hogy erre a célra egy számlálót tartanak fenn, és abban mindig egy meghatározott alapidőpont (általában 1970. január 1-je, déli 12 óra) óta eltelt órajelek vagy másodpercek számát tárolják.

Ez azonban felvet egy bosszantó kérdést, ha ugyanis az eltelt órajeleket számoljuk, könnyen bajba kerülhetünk, ha a számláló méretének a megválasztásakor nem vagyunk eléggé körültekintőek. Vegyünk példának egy 32-bites számlálót, amely első hallásra nagyon sok órajel ábrázolására képes (több mint négymilliárdra), de ha utánaszámolunk, rájövünk, hogy kevesebb mint két év alatt túlcsordul.

A legkézenfekvőbb megoldásnak az tűnik, ha az órajelek számlálására nem 32-bites, hanem 64-bites számlálót használunk. Ezzel csak az a gond, hogy egy másodperc alatt sokszor meg kell változtatnunk a számláló értékét, ami rendkívül időigényes. Ezért kedvezőbb az a megközelítés, ha a számláló 32 bit marad, és az órajelek helyett az eltelt másodpercek számát raktározzuk. Ebben az esetben szükség van egy másik számlálóra, ahová a beérkező órajeleket tesszük. Egy 60 Hz-es óra esetében egy másodperc alatt pontosan 60 megszakítás kelet-

kezik, így amikor az órajelszámláló eléri a hatvanat, lenullázzuk, és a másodpercszámláló értékét eggyel növeljük. Sajnos ezzel a módszerrel is gond van: csak 136 évig alkalmazhatjuk, utána a másodperceket számláló 32-bites változó túlcsordul. Ennek következtében egy harmadik megoldásra is szükség van, ami kerüli a nagy méretű számlálók használatát, de nincs benne ez a 136 éves korlát. A megoldás nagyon egyszerű: nem egy meghatározott alapidőponttól (például 1970-től) kezdve számolunk, hanem a rendszer indulása óta eltelt órajeleket figyeljük. Amikor a rendszer betöltődik, kiolvassa a háttéróra állását, átváltja az alapidőponttól eltelt másodpercekre, majd raktározza a memóriában. Ha valaki a pontos időre kíváncsi, csupán egy egyszerű összeadás műveletre lesz szükség: a rendszer indulásának időpontját össze kell adni a számlálóval.

Korábban már említettük, hogy ha a gépünkben nem lenne óra, az operációs rendszernek esélye sem lenne felügyelni az olyan folyamatokat, amelyek futás közben egyetlen megszakítást sem okoznak (azaz egyetlen B-K műveletet sem hajtanak végre, csak számolgatnak magukban csendesen). Ez azonban még csak a jéghegy csúcsa, ezek a folyamatok ugyanis egyszer várhatóan abbahagyják a futást, esetleg mégis elvégeznek egy B-K műveletet. De ha egy folyamat például végtelen ciklusba kerül, az idők végezetéig várhatunk.

Szerencsére ilyen esetek nem fordulhatnak elő, mivel minden számítógép rendelkezik órával, így bármi történjék is, az ütemező előbb vagy utóbb biztosan „szóhoz fog jutni”, azaz lehetősége lesz megakadályozni, hogy akár egy B-K műveleteket kerülő folyamat is sokáig fusson. Ezt nyilván úgy éri el, hogy az éppen futó folyamathoz egy számlálót rendel, amelybe beírja, hogy a folyamat hány órajelen keresztül futhat. A számláló értékét minden órajel-megszakításkor csökkenteni kell, és ha eléri a 0-t, akkor az éppen futó folyamatot fel kell függeszteni, és a futási lehetőséget egy másiknak át kell adni.

A gyakorlatban nagyon fontos kérdés, hogy az operációs rendszer melyik része végezze például a valós idő, illetve az éppen futó folyamathoz rendelt számláló karbantartását. Mert például az utóbbi ugyan az ütemező feladata lenne, de bizonyos szempontból logikusabb, ha az órakezelő végzi, és ha letelt az egy folyamatra szánt idő, majd ő hívja meg az ütemezőt. Ezek olyan tervezési kérdések, amelyek felvetésekor sokféle szempontot figyelembe kell venni.

E szempontok közül talán az egyik legmeghatározóbb a sebesség kérdése. Az órajel-megszakítás ugyanis nagyon gyakori jelenség, másodpercenként sok ilyen történik. Ezért mindig arra kell törekedni, hogy az órakezelő kódja minél gyorsabb legyen. Az órakezelő feladatait nagyjából két részre oszthatjuk. Az első csoportot azok a műveletek alkotják, amelyeket kivétel nélkül minden órajel-megszakításkor el kell végezni, a másikat pedig azok, amelyeket csak bizonyos esetben. Így nem rossz ötlet ezeket a feladatokat különválasztani, és más-más szinteken végrehajtani.

Hogy egy kicsit jobban értsük, miről van szó, nézzünk egy példát ennek a megvalósítására. Példánk alanya most a Minix nevű operációs rendszer lesz, amelyből maga a Linux is sokat tanult. Mivel a Minix jellemzően folyamatokra épülő rendszer, minden folyamat (és eszközkezelő) üzenetküldéses módszer segítségével tartja a kapcsolatot egymással. Ha egy bizonyos eszköztől beérkezik egy megszakítás, a megszakításkezelő üzenetet küld a megfelelő (éppen blokkolt állapotban lévő) eszközkezelőnek, ami az üzenet hatására felébred, és nekilát az esemény kezelésének. Ám ez az óra esetében egyáltalán nem hatékony, mivel az üzenetküldés viszonylag lassú módszer. Megoldást jelenthetett volna, ha ebben az esetben felhagynak a folyamatstrukturáltság elveivel, és egyszerűen egy közvetlen rutinhívás segítségével a vezérlést egyből az órakezelőnek adják át. A Minixet azonban nem „éles” rendszernek tervezték, hanem oktatási célokra, így működését mindenki szabadon tanulmányozhatja. (Ennek köszönhetően a Minix egyike a legjobban átgondolt felépítésű operációs rendszereknek.) Tehát a Minix tervezői nem akartak ilyen „durva” megoldásokat alkalmazni, ezért a következőt találták ki: bizonyos – az óramegszakítással kapcsolatos – feladatokat maga az operációs rendszer megszakításkezelője intéz el. Ilyen feladat például a pillanatnyi futó folyamathoz rendelt számláló értékének csökkentése. Az órakezelőnek csak abban az esetben küld üzenetet (tehát csak akkor fog meghívódni), amikor valamilyen összetettebb feladat elvégzésére van szükség (lásd később).

Láthatjuk tehát, hogy bizonyos esetekben a feladatokat érdemes úgymond szétosztani az operációs rendszer különböző részei között. Jelen esetben ez a megosztottság több mint 15 százalékos sebességnövekedést jelent, ami az operációs rendszerek körében rendkívülinek számít.

A processzoridő könyvelése és a felügyeleti időzítők

A továbbiakban két nagyon hasznos, szintén az órakezelőre háruló feladatot mutatunk be. Az első az úgynevezett processzoridő-könyvelés, ami azt jelenti, hogy nyilvántartjuk, melyik folyamat mennyi ideig futott (azaz mennyit használta a processzort). Erre kétféle módszer kínálkozik: egy pontos és egy kevésbé pontos. Az első esetben minden folyamathoz rendelünk egy időzítőt, és amikor a folyamat megáll (vagy felfüggesztődik), a tartalmát mentjük, és amikor ismét futni kezd, visszatöltjük.

A másik módszer az, hogy a folyamattáblában egy bejegyzést készítünk, ami a futási időt fogja tárolni. Amikor óramegszakítás érkezik, megkeressük az éppen futó folyamatot a folyamattáblában, majd a futásidőt tartalmazó bejegyzését eggyel növeljük. Ez az elsőnél jóval egyszerűbb módszer, de sajnos akad egy hátránya: amikor a processzor a beérkező megszakításokkal foglalkozik, az is a folyamat „számlájára íródik”. Tehát nem a tiszta futási időt kapjuk vissza, hanem azt is, amikor például a lemezkezelő egy beérkezett megszakítás kezelésével volt elfoglalva. Ez azzal jár, hogy ha sok ilyen megszakítás történik, akkor nem kapunk pontos eredményt.

Némelyik operációs rendszer rendelkezik egy ***Profiling*** nevű szolgáltatással (amit futásidő-elemzésként fordíthatnánk). Ez azt jelenti, hogy nemcsak az egész programnak, hanem az egyes részeinek a futásidejét is mérhetjük.

Az órakezelő másik hasznos szolgáltatása, hogy minden folyamat kérhet tőle figyelmeztetést (más néven riasztást vagy időzítést), ha egy meghatározott idő eltelik. Ez a gyakorlatban úgy működik, hogy a folyamat beállít az órakezelőnek egy időpontot, és amikor az elmúlt, egy figyelmeztetést kap – ez lehet egy üzenet, egy megszakítás vagy akár egy jel (ez rendszerenként különböző).

Lássunk néhány példát

Mire lehet ezt használni? Vegyük a legegyszerűbb példát, mondjuk egy vizsgáztató alkalmazást. Ez a felhasználónak (jelen esetben a vizsgázónak) feltesz egy kérdést, amelyre az adott időn belül válaszolni kell, ezért a program kér egy figyelmeztetést a kérdés megválaszolásához rendelkezésre álló időre. Ha a felhasználó közben választ ad, akkor törli a figyelmeztetést, majd megnézi, hogy helyes-e a felelete. Ha a figyelmeztetésig nem érkezik válasz, elárulja a megoldást, majd a következő kérdésre ugrik.

Egy másik jellemző példa riasztás használatára a hálózati alkalmazások működése. Vegyük azt az esetet, amikor egy másik gépnek elküldünk egy csomagot, és megerősítést (nyugtázást) várunk arról, hogy az általunk küldött csomag megérkezett-e. Ilyenkor beállíthatunk egy figyelmeztetést, és ha a nyugta a riasztás idejéig nem érkezett meg (mert például az általunk küldött csomag elveszett, vagy maga a nyugta tűnt el valahol idefelé jövet), akkor a csomagot újra kell küldenünk.

Érdekes megnézni, hogy az órakezelő miképpen valósítja meg a figyelmeztetéseket. Mint már említettük, a számítógépben lévő programozható órameghajtó általában több, egymástól független órát tartalmaz, amelyeket külön-külön használhatunk. Ha tehát van elég óránk, nincs más dolgunk, mint minden egyes figyelmeztetéshez beállítani egyet, hogy amikor a megadott idő letelik, megszakítást hozzon létre.

Sajnos ennyi óra csak a legritkább esetekben áll rendelkezésünkre, ezért az órakezelőnek több órát kell utánoznia. Erre sokféle módszer kínálkozik, de általában a következőt használják: a kért riasztásokat egy táblázatba rendezik. A táblázatból kikeresik azt az időpontot, amelyik a leghamarabb fog bekövetkezni, és ezt egy változóban tárolják. (Ezt a műveletet mindig meg kell ismételni, valahányszor csak új riasztási kérelem érkezik be). Amikor a pontos idő módosítására kerül a sor (tehát nem minden óramegszakításkor), az órakezelő megnézi, hogy ez az időpont elérkezett-e már. Ha nem, akkor nem történik semmi, ha igen, akkor elküldi a figyelmeztetést a megfelelő folyamatnak, majd kikeresi a következő riasztás időpontját, és kezdődik minden elölről.

A felügyeleti időzítés

A figyelmeztetés különleges formája a felügyeleti időzítés (watchdog timing). A különbség annyi, hogyha a megadott idő letelik, akkor nem egy jel (signal) vagy egy üzenet (esetleg megszakítás) történik, hanem egy előre meghatározott eljárás hívódik meg. A felügyeleti időzítőket az eszközmeghajtók használják.

Két kérdés merülhet fel. Az első (erre könnyebb válaszolni): mikor lehet szüksége egy eszközkezelőnek időzítésre? Ehhez sorozatunk előző részét érdemes megnézni, amelyben a hajlékonylemez-meghajtókról írtunk: megemlítettük, hogy a hajlékonylemez-kezelő bizonyos biztonsági intézkedéseket kap (például a kért blokk a megadott idő alatt a lemez hibája miatt nem érkezik meg, és a meghajtó többszöri próbálkozásra sem tudja beolvasni). A felügyeleti időzítés például ilyen biztonsági intézkedés. A lemezkezelőhöz tartozik egy eljárás, ennek címe adódik át az órakezelőnek. Ha a beállított idő lejár, az órakezelő önműködően meghívja ezt az eljárást, ami már „tudja”, hogy a kért művelet nem járt sikerrel, és megszakítja az olvasási folyamatot.

De felügyeleti időzítést használunk akkor is, amikor például a hajlékonylemez motorját kapcsoljuk be. Mint már említettük, amikor nincs lemezművelet, a motor ki van kapcsolva, hogy ne kopjon a lemez. Amikor kérés érkezik, először be kell kapcsolnunk, ám ez időbe telik, csaknem fél másodpercbe. Addig a lemezkezelőnek unatkoznia kell, tehát kér egy időzítést fél másodpercre, és blokkol. Láthatjuk tehát, hogy a felügyeleti időzítés a B-K eszközök programozásánál szinte nélkülözhetetlen.

A másik kérdés az, hogy az eszközkezelők miért felügyeleti időzítést alkalmaznak, és nem „hagyományosat”. Erre az a válasz, hogy ezek a rendszermagon belül futnak. A Minix például jelek formájában küldi a folyamatoknak a figyelmeztetést. Rendszermagon belül azonban nincsenek jelek. A másik indok az, hogyha mégis üzenetet vagy valami hasonlót küldünk, nem biztos, hogy azt a eszközkezelő megkapja vagy kezelni tudja. Elképzelhető, hogy az eszközben lefagyás történt, és addig meg se moccan, amíg az általa kezelt eszköztől választ nem kap. Ezért sokkal biztonságosabb, ha közvetlen eljáráshívást hajtunk végre, így a meghívott eljárásnak lehetősége lesz életre kelteni a kezelőt, és kiírni a hibaüzenetet. (Ezzel egy kicsit megcáfoltuk az előzőekben tett állításunkat, miszerint a Minix folyamatokra épülő rendszer, így tartózkodni kell a közvetlen eljáráshívásoktól. Sajnos vannak helyzetek, amikor ki kell egyeznünk, és bizonyos engedményeket kell tennünk. Ez is egy ilyen helyzet. Ha ebben az esetben sem engednénk meg a közvetlen eljáráshívást, más módszerre lenne szükségünk, például egy megszakításra, amellyel ezt helyettesíteni tudnánk. De a rendszermagon belüli megszakítások elég veszélyesek lehetnek. Kereshetünk valami kevésbé durva megoldást is, viszont annak megvalósítása annyi munkával járna, hogy nem érné meg.)

Ezzel végére is értünk a beviteli–kiviteli eszközkezelés leírásának. Ezek után elhagyjuk az eszközszintet, és a memóriával kezdünk el foglalkozni, illetve az operációs rendszer úgynevezett memóriakezelőjével. Ebbe a témakörbe rengeteg minden beletartozik, például a virtuális memóriakezelés, az osztott könyvtárak (libraries) működése; de azt is követni fogjuk, hogy mi történik akkor, ha egy alkalmazást Linux alatt elindítunk. Ezzel párhuzamosan a fájlrendszerrel is megismerkedünk, amely a Unixok esetében igencsak központi szerepet játszik. Ugyanis a fájlrendszer nemcsak az adataink tárolásáért felelős, hanem azért is, hogy a folyamatok a szükséges erőforrásokat el tudják érni, illetve ellenőrizze, hogy minden felhasználó csak azt tegye, ami számára engedélyezett (tehát a jogosultságok ellenőrzése is ide tartozik).

Ezek a területek az operációs rendszernek azok a részei, amelyet a felhasználó már közvetlenül érzékel. Míg az eszközkezelés a legtöbb rendszerben többé-kevésbé ugyanazon az elven működik, addig az előbb említett szolgáltatások megvalósítása rendszerfüggő lehet. Ezért most igyekszünk már nemcsak általánosságokban beszélni az operációs rendszerekről, hanem gyakorlati példákat is megemlítünk. Továbbra is elsősorban a Unix alapú rendszereknél (elsősorban a Linuxnál) maradunk, de összehasonlításképpen kitekintünk más „létsíkokra” is, például a Windows NT/2000/XP-k felé is.

Témánk tehát akad bőven, így a következő részben neki is kezdünk. Az első, amivel foglalkozni fogunk, a memóriagazdálkodás lesz.

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.