Operációs rendszerek (2. rész)

Folytatjuk az operációs rendszerek belső „titkait” feltáró sorozatunkat. Az első részt néhány elengedhetetlen alapfogalom tisztázására szántuk. A dolog most kezd igazán érdekessé – és talán kissé meredekké – válni. A továbbiakban részletesebben sorra vesszük az operációs rendszer részeit és a közérthetőségre törekszünk.

Először arra keressük a választ, miként válik lehetségessé, hogy több alkalmazást egyszerre futtassunk. A mai világban az operációs rendszerekkel szemben alapkövetelmény több program egy időben történő futtatása. Manapság már egyáltalán „nem nagy szám”, hogy a gépen végzett lázas munka közben vadul bömböltethetjük MP3-lejátszónkat, hogy féltucatnyi letöltést indíthatunk el a háttérben, vagy a főnök mihamarabbi távozásáig kedvenc játékunkat a háttérbe bújtathatjuk. A mai operációs rendszereknek azonban komoly kihívásokkal kell szembenézniük, hogy mindez valósággá váljon. A továbbiakban megkíséreljük bemutatni azt az alapelvet, amely mindezt lehetővé teszi, és a gyakorlati megvalósításról is ejtünk pár szót.

Természetesen ez az „egyszerre való futás” csak látszat, ugyanis a processzor egy időben csak egy utasítást tud végrehajtani, azaz mindig csak egy programon dolgozhat. Az operációs rendszer azonban szabályos időközönként felfüggeszti az éppen futó alkalmazást, és egy másiknak adja át a lehetőséget. Másodpercenként több ilyen váltás történik, ezért a felhasználónak olyan érzete támad, mintha alkalmazásai a szó legszorosabb értelmében egyszerre futnának, de valójában csak ***látszólagos párhuzamosságról*** van szó. Az egynél több processzorral felszerelt masináknál más a helyzet, az ilyen gépek esetén a valós időben való egyszerre futtatás is megvalósítható.

Jogosan merülhet fel az a kérdés, hogy miért kell ezzel a témakörrel kezdeni egy operációs rendszerek működésével foglalkozó sorozatot. Ahelyett, hogy ezt most példával szemléltetnénk, egyszerűen csak elégedjünk meg annyival, hogy azért, mert a rendszer összes többi része eköré épül fel. Megígérjük, hogy a cikk utolsó szakaszában ezt a kérdést is megválaszoljuk.

A folyamat

Még mielőtt bármibe belekezdenénk, nagyon fontos, hogy tisztázzuk a folyamatok fogalmát. Az előző részben ugyan már meghatároztuk, de mivel ez az operációs rendszerek világának talán legfontosabb kulcsfogalma, nem árt, ha egy kicsit visszatérünk rá, illetve pontosítjuk az ott leírtakat.

Minden általunk elindított alkalmazást, amely a többi már futó programmal párhuzamosan dolgozik, folyamatnak nevezünk. Nem ártana azonban, ha egy kicsit másmilyen megvilágításból is szemügyre vennénk a dolgot.

A folyamat valójában csak egy tevékenységet jelöl, amelynek vannak be- és kimenő adatai, programja (algoritmusa, amely az adott feladat megoldásának menetét írja le), továbbá állapota. A processzor idejét több folyamat között megoszthatjuk. Ennek a megosztásnak a vezérléséről az úgynevezett ***ütemező*** gondoskodik (lásd később). Nagyon fontos, hogy a folyamatok által végzett tevékenység bármikor felfüggesztődhet, például azért, mert a rendszernek hirtelen egy fontosabb (nagyobb fontosságú) folyamatot kell kiszolgálnia. Ez pedig azzal jár, hogy nem jósolhatjuk meg előre, hogy a folyamat által előírt műveletek mennyi idő alatt kerülnek elvégzésre, tehát akárhányszor elindítunk egy folyamatot, elég nagy valószínűséggel mindig más-más futási időt fog eredményezni. Ezért általános alapelv, hogy a folyamatokon belül semmiféle belső időzítési feltételt nem használhatunk.

Lehetséges, hogy ez elsőre kissé nehezen emészthető, de semmi pánik!

Egyet jegyezzünk meg: minden általunk elindított program egy-egy folyamatnak fog megfelelni, amelyek látszólag egymással párhuzamosan futnak, valójában egymást felváltva.

Annak eldöntése, hogy melyik folyamat mikor és meddig futhat, a rendszerbe beépített ütemező feladata.

A folyamatfa

Az operációs rendszerek életében kulcsfontosságú kérdés, hogy folyamatainak nyilvántartásáról miként gondoskodik. A Unix-rendszerek a folyamatok logikai tárolására a könyvtárszerkezethez hasonló úgynevezett hiearchikus (fa) szerkezetet használják.

A Unix világában a folyamatokat a már meglévő folyamatok a `FORK` rendszerhívás segítségével hozhatják létre.

A folyamat által létrehozott gyermekfolyamat szülőfolyamatának pontos másolata lesz. Fontos megjegyeznünk, hogy a hiearchikus szerkezet következtében minden folyamatnak csak egyetlen szülője lehet, de elméletileg korlátlan számban „gyárthat” gyermekfolyamatokat.

A Unix-típusú rendszereknél a fa gyökerét például az `init` nevű folyamat képezi. Ez az egyetlen olyan folyamat, amelynek nincs szülőfolyamata, létrehozásáról a rendszer indulásának pillanatában maga a rendszermag gondoskodik.

Az `init` folyamat a következőt végzi: beolvassa a */etc/inittab* állományt, amelyben meg van adva, hány ***terminálfolyamatot*** kell létrehoznia. A terminálfolyamatok szülője az `init` lesz, és addig várakoznak, amíg csak az adott terminálon egy felhasználó be nem jelentkezik. Ha ez megtörtént, a terminálfolyamat elindít egy héjat, amely az ő gyermekfolyamata lesz. Pillanatnyi folyamatfánkat a `pstree` parancs segítségével bármikor megtekinthetjük.

A folyamatok fizikai tárolása teljesen rendszerfüggő. A Linux ezt a következőképpen oldja meg: valahol a rendszermemória mélyén létezik egy táblázat, amelyet feladatmutató táblának nevezünk. A táblában minden folyamathoz egy bejegyzés tartozik.

A feladatmutató tábla mérete állandó, általában legfeljebb 512 bejegyzést tartalmazhat, tehát egyszerre nem lehet több folyamatunk. Ha egy folyamat időközben munkájának végére ér, vagy

mi magunk szüntetjük meg, a helye természetesen feladatmutató táblából felszabadul.
A feladatmutató tábla semmiféle adattal nem rendelkezik a folyamatokról, csupán mindegyikhez egy memóriacímet rendel, amely megmondja, memóriánk mely részében találjuk meg azt az adatszerkezetet, amely az adott folyamat tulajdonságait tartalmazza. A Linux-rendszermag elég sok tulajdonságot tárol a folyamatokról. Milyen tulajdonságai lehetnek egy folyamatnak? Például állapota, ütemezési beállításai, a folyamatfában elfoglalt helye, olyan adatok, amelyek ahhoz szükségesek, hogy ismét elindíthatóak legyenek (például a regiszterek állapota) stb. A Linux-mag rengeteg mindent tárol egy folyamatról, mi most csak a legfontosabbakra térünk ki részletesebben.

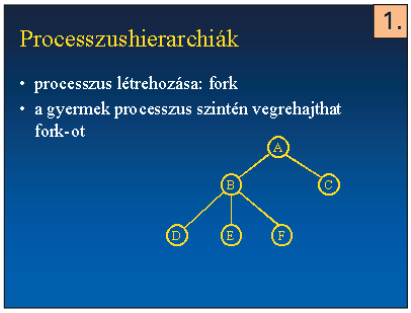
Folyamatállapotok

A folyamatok hagyományosan háromféle állapotban létezhetnek: futó, futásra kész és blokkolt állapotban. Az első állapot azt jelenti, hogy az adott folyamat jelen pillanatban fut, azaz őt illeti meg a processzor használatának joga. A futásra kész állapotba tartozó folyamatok futni szeretnének, habár erre nem ők kapták meg a lehetőséget. Amikor az üzemező úgy dönt, hogy a futó folyamat eleget használta a CPU-t, állapotát futásra készre változtatja, és – egy bizonyos algoritmus szerint – a többi futásra kész folyamat közül kiválaszt egyet. A szerencsés „nyertes" helyzete ezután futóra változik. Elmondhatjuk tehát, hogy az első két állapot logikailag ugyanaz.
Ám előfordulhatnak olyan esetek is, amikor bizonyos külső esemény bekövetkeztéig a folyamat nem képes folytatni a munkát. Vegyük például a parancsértelmezőt, amelynek az a feladata, hogy fogadja és végrehajtsa a felhasználó utasításait. A parancsértelmező addig nem képes dolgozni, amíg a felhasználó le nem üt egy billentyűt, tehát amíg ez az esemény be nem következik, a folyamat blokkolódik. Felébredni csak akkor fog, miután a felhasználó megtalálta és megnyomta a keresett billentyűt. Ekkor ismét futásra kész helyzetbe kerül, sőt, ha nincs nála fontosabb (nagyobb fontosságú) folyamat, akár meg is kaphatja a processzor használatának lehetőségét. Miután a parancsértelmező feldolgozta a beérkezett adatot, a következő billentyű lenyomásáig ismét blokkolásra kerül.
A folyamatok blokkolásának másik jellemző esete, amikor bizonyos bemenő adatokat egy másik folyamattól várunk, azok azonban valamilyen okból kifolyólag még nem érkeztek meg. Nézzük meg például a `cat szoveg | grep micimack` utasítást! A `cat` folyamat nem tesz mást, mint kimenetként visszaadja a ***szoveg*** nevű állomány tartalmát, a `grep` pedig a bemenő adatokból kiválasztja azokat a sorokat, amelyekben a „micimackó" szó szerepel. A csővezeték (|) segítségével a `cat` folyamat kimenetét a `grep` folyamat bemenetébe „folyathatjuk". Előfordulhat olyan eset, hogy a grep folyamat már futásra készen áll, viszont még nem kapott adatot a `cat`-től, vagy amit már kapott, réges-régen feldolgozta, és várja az újakat. Ilyenkor a `grep` blokkolásra kerül, amíg a `cat` folyamattól újabb adat nem érkezik. (A folyamatok közti adatcserével – IPC – következő számunkban részletesen foglalkozunk majd.)
Most nézzük meg, milyen folyamatállapotokról beszélhetünk a Linux esetében. A Linux nem tesz különbséget a futó és a futásra kész állapot között, mindkét helyzetet „running"-nak (futónak) hívja. A blokkolt folyamatok „waiting" (várakozó) helyzetbe kerülnek. Lehetőségünk nyílik – szintén jelek (signals) segítségével –, hogy egy folyamatot megállítsunk. Ekkor az úgynevezett „stopped" (leállított) állapotba kerül. Érdemes az eszünkbe vésni, hogy ez nem egyenlő a folyamat megszüntetésével, ugyanis a folyamat bármikor újra elindítható arról a pontról, ahol futását megszakítottuk.
Említést érdemelnek még az úgynevezett zombifolyamatok. Ezek olyan halott folyamatok, amelyek megszüntetésre kerültek, azonban valamilyen okból még mindig szerepelnek a feladatmutató táblában.

Ütemező, megszakítások és egyéb nyalánkságok

Annak eldöntése tehát, hogy a folyamatok közül ki mikor és mennyi ideig használhatja a processzort (azaz futhat), az ütemező feladata. Nézzük, miképp birkózhat meg ezzel a feladattal!
Az ütemező alapvetően két módszere létezik. Az ősibb módszer az úgynevezett ***készre futtatás*** elve, amelynek megvalósítása egyáltalán nem bonyolult, viszont egy általános célú többfelhasználós operációs rendszer „meghajtására" alkalmatlan. Nem véletlenül ***nem megszakítható ütemezésnek*** is nevezik, mivel a rendszer nem szól bele, hogy egy folyamat meddig futhat (azaz futását időközben nem szakítja meg). Ha egy folyamat megkapta a processzor használatának lehetőségét, akkor addig marad nála, amíg vagy önszántából át nem adja, vagy nem blokkolódik. Az ilyen rendszerekben könnyű önzőnek lenni: az olyan folyamatok, amelyek kizárólag számítási műveleteket hajtanak végre, akár az idők végezetéig is kisajátíthatják maguknak a processzort, megfosztva ezzel a többi folyamatot a futási lehetőségtől. Nem is beszélve arról, hogy komoly kellemetlenség érhet minket, ha egy hibásan megírt folyamat időközben lefagy, és képtelenné válik a processzor átadására.
A jelenlegi korszerű operációs rendszerek szinte kivétel nélkül a ***megszakítható ütemezés*** elvét alkalmazzák. Itt az ütemezőnek rendszeres időközönként lehetősége nyílik átadni a futási lehetőséget egy másik folyamat számára. Ez már valóban hatékonyabb megoldás, ám ennek is akadnak árnyoldalai. Például gondoskodnunk kell arról, hogy úgynevezett ***versenyhelyzetek*** ne forduljanak elő, mert ezek számos kellemetlenség forrásai lehetnek (a versenyhelyzetek fogalmával következő számunkban foglalkozunk, most legyen elég annyi, hogy e jelenségek megakadályozása komoly kihívás az operációs rendszerek fejlesztői számára).
Az utóbbi módszer megvalósítását a ***gépszintű megszakítások*** (hardware interrupts) teszik lehetővé. Akik elmélyültek már a gépközeli programozásban, biztos tisztában vannak a megszakítások fogalmával, ám a többiek kedvéért tegyünk egy kis kitérőt. A megszakításoknak két fajtája van: a programszintű és a gépszintű. A különbség annyi, hogy a programszintű megszakítást egy program, a gépszintűt pedig valamely a számítógépünkhöz csatlakozó eszköz válthatja ki. Minket most csak az utóbbi típus érdekel.
Az Intel (és a hasonló típusú) processzorokkal felszerelt PC-k általában 15 különböző sorszámú gépszintű megszakítással (IRQ) rendelkeznek, és egy IRQ-n egyszerre csak egy eszközt működtethetünk.
A gépszintű megszakítást vagy valamilyen külső esemény (például billentyűlenyomás, egy adatcsomag megérkezése a hálózatról stb.), vagy valamilyen kért szolgáltatás teljesítése, például egy lemezművelet befejezése válthatja ki.
Mi történik megszakításkor? A pro-

cesszor azonnal abbahagyja, amit éppen tesz (például egy folyamat végrehajtását). Megjegyzi, hogy „hol tartott”, majd a memória elején lévő ***megszakításvektor-táblából*** kiolvassa az adott megszakításhoz tartozó ***kiszolgálóeljárás*** címét, amely megszakítás kezeléséért felelős. Ezután megkezdődik az eljárás végrehajtása. Az itt leírt megoldás teljes egészében gépi szinten zajlik, az operációs rendszer fejlesztőinek csak a kiszolgálóeljárás megírásával kell bajlódniuk, illetve a megszakításvektor



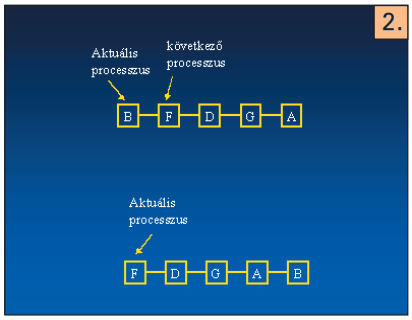
tábla helyes beállításával. Az eljárás feladata végeztével „visszaküldheti” a processzort, hogy folytassa eredeti munkáját.

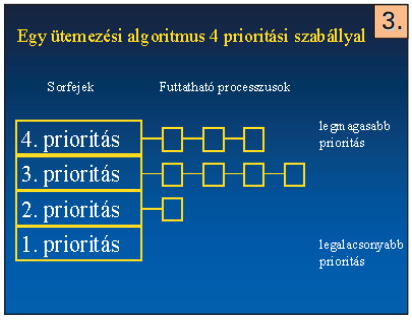
Talán nem kell ecsetelnünk, hogy egy gépszintű megszakítás mennyire kedvező alkalom arra, hogy a futási lehetőséget elvehessük a futó folyamattól, és átadhassuk egy másiknak. A dolgunk csak annyi, hogy a kiszolgálóeljárás ne az eredeti helyre térjen vissza, hanem az ütemezőt hívja meg, ami majd eldönti, mi legyen a továbbiakban.

Felmerülhet a kérdés, hogy mi történik akkor, ha csak vad számításokat végző folyamatok futnak, amelyek semmiféle eszközt nem használnak, így megszakítások sem igazán történnek. Szerencsére ekkor se lenne semmiféle gond, mivel a 8-as megszakításra egy ***valós idejű órát*** „kötöttek”, amely szabályos időközönként megszakításokat vált ki. Hogy mekkora időközönként, az számítógéptípusonként változik, sőt, bizonyos géptípusnál ezt az időtartományt akár programból is beállíthatjuk. Általában másodpercenként ötven vagy hatvan megszakítás történik.

A fent leírtakból megállapíthatjuk, hogy az operációs rendszer ütemező és megszakításkezelő része egymással szoros kapcsolatban áll és el nem választható. Ám még mielőtt rátérnénk magára az ütemezőre, nézzük meg, mit is csinál egy gépszintű megszakításhoz tartozó kiszolgálóeljárás. Az előző részben bemutattuk a Linux elvi felépítését.

Láthattuk, hogy az eszközmeghajtók az ütemező feletti szinten helyezkednek el. Ezek az erőforrásaink kezeléséért felelős eljárások szintén folyamatként futnak, ám a felhasználói alkalmazásoknál jóval alacsonyabb szinten: a rendszermag szintjén, úgynevezett magmódban. Ennek köszönhetően, ha némi megkötésekkel ugyan, de közvetlenül hozzáférhetnek a gép alkatrészeihez, és az ütemező is másképp bánik velük, mint a közönséges folyamatokkal. Ezért megkülönböztetésképpen a





továbbiakban ***eszközvezérlők***-nek fogjuk hívni őket, mivel az eszközvezérlő és a folyamat közötti különbség valójában elhanyagolható.

Az eszközvezérlők idejük nagy részét az adott rendszerhívásra várva blokkolt állapotban töltik. Amikor az megérkezik, a kiszolgálóeljárás egy üzenetet küld nekik, amelynek hatására blokkolt állapotból futásra kész helyzetbe kerülnek. Amint az ütemező biztosítja nekik a futási lehetőséget, elkezdik feldolgozni az eszköztől kapott adatokat. Például azt, hogy billentyűzetmegszakítás esetén melyik billentyű lett lenyomva, ezt az adatot továbbítják például a megfelelő parancsértelmező folyamatnak, amely blokkolt állapotban – ki tudja, mióta – már csak erre várt.

Most adtunk választ a cikkünk elején feltett kérdésre, hogy miért a folyamatkezelés a legfontosabb a Linuxhoz hasonló felépítésű operációs rendszerek életében. Ezért. Az eszközmeghajtók is tulajdonképpen folyamatok, amelyek semmit sem tudnak a megszakításokról, mivel azokat „elrejti” előlük az alattuk lévő szint. Az operációs rendszer összes további tulajdonsága legnagyobb mértékben a folyamatkezeléstől, pontosabban az ütemezőtől függ, amely az egész megszakításkezelő rendszerrel az operációs rendszer legbelső magját alkotja.

Ütemezési algoritmusok

Az ***ütemezési algoritmus*** nem más, mint az a módszer, amelynek alkalmazásával az ütemező el tudja dönteni, hogy az elkövetkező néhány ezredmásodpercig melyik folyamat lesz „méltó” a processzszor birtoklására. Természetesen csak a futásra kész állapotba tartozó folyamatok játszanak szerepet, a blokkoltakkal nem kell foglalkozni.

1. kép A folyamatfa, amelyben minden folyamatnak van szülőfolyamata, de elméletileg korlátlan számú gyermekfolyamata lehet. A gyermekfolyamatok is további folyamatokat hozhatnak létre.

2. kép A forgatás- (round robin) ütemezés: az ütemező sorban végigmegy az összes folyamaton. Minden folyamat ugyanannyi ideig futhat, mint a többi.

3. kép Elsőbbségi ütemezés: az ütemező elsőként mindig a legmagasabb elsőbbségű folyamatokat futtatja, a többiekre csak ezután kerülhet sor.

Sokféle ütemezési algoritmus létezik. Hogy melyik rendszer melyiket használja, az a megoldandó feladattól függ. Mindenesetre van néhány alapelv, amelyhez ragaszkodni kell.

Az első a hatékony CPU-kihasználás, azaz úgy beosztani a folyamatokat, hogy a lehető legkevesebb idő teljen a feladat megoldása szempontjából hasztalan tevékenységgel. Ilyen „hasztalan tevékenységnek” számít például a váltás két folyamat között, amely roppant időigényes dolog, mivel pontosan fel kell jegyezni, hogy az előző folyamat hol tartott a futásban. Ha túl gyakran váltogatunk folyamatokat, akkor több váltás lesz, azaz a processzor idejének viszonylag nagy százalékát erre fogja pazarolni. Az sem jó, ha túl ritkán váltunk, mert akkor egy folyamatnak sokat kell várnia, amíg ismét rákerül a sor. Ha csak számításokat végző programokat futtatunk, az utóbbi eset még nem is lenne annyira súlyos, de egy felhasználói beavatkozást igénylő alkalmazás (például egy játék-

program) esetében a felhasználó nem biztos, hogy meg lenne elégedve a használhatóságával.
A másik dolog, amire ügyelni kell, hogy minden folyamat testvériesen részesedjen a rendelkezésre álló processzoridőből. Egyrészt ne forduljon elő, hogy az algoritmus hibájából egy folyamat sokkal több időt kap, mint társai.
A felhasználók biztosan ferde szemmel néznének egy olyan rendszerre, ahol például a *pi* kétmillió tizedesjegyig tartó kiszámítása pár másodperc alatt megvan, ellenben egy könyvtár listázására perceket kell várni. Másrészt ügyelni kell arra is, nehogy egy-egy folyamatot véletlenül „kifelejtsünk" a listából. Arra kell törekedni, hogy nagyjából mindegyikre ugyanannyiszor kerüljön sor.
Ezek után nézzük, milyen ütemezési algoritmusok vannak. A legegyszerűbb a forgatás (***round robin***). Sorban végigmegyünk az összes folyamaton, és minden folyamat egy meghatározott időszeletet kap.

Ennél a fontossági (***priority***) ütemezés kicsit többre képes, ahol szintén minden folyamat egyenlő, de egyesek egyenlőbbek. Azaz a folyamatokat fontossági sorrend – elsőbbség – szerint szervezzük. Amelyik a legnagyobb elsőbbséget birtokolja, az kapja meg gyakrabban a processzort. A másik változata a ***sorsjáték-ütemezés***. Az összes folyamatnak kiosztunk egy adott számú sorsjegyet, amelyek közül az ütemező minden alkalommal „kihúz" egyet. A szerencsés nyertes (az a folyamat, amelyik a kihúzott sorsjegyet birtokolja) fogja megkapni a lehetőséget a futásra. A fontosabb folyamatok több sorsjegyet birtokolhatnak, ezáltal nagyobb eséllyel indulhatnak a processzorért való küzdelemben.
A Linux-rendszerben az eszközvezérlők rendelkeznek a legnagyobb fontossági szinttel, így ha akad futásra kész eszközvezérlő, akkor az ütemező mindig őket fogja kiválasztani. Fontos, hogy a folyamatokkal ellentétben a eszközvezérlők addig futnak, amíg nem blokkolódnak, azaz ebben az esetben a készre futtatás elvét alkalmazzuk. A felhasználói folyamatok esetében a forgatásos algoritmust alkalmazzuk. Ha nincs futtatható eszközvezérlő vagy folyamat, az ***IDLE*** (tétlen) különleges folyamat kapja meg a vezérlést. Ennek nincs más dolga, mint addig unatkozni, amíg nem találunk egy másik futásra kész folyamatot.
Sajnos nem szabadultunk meg ilyen könnyen a folyamatoktól. A következő részben még megtárgyalunk néhány olyan fogalmat, amit nem árt, ha ismerünk (például a vezérlési szálakat), valamint belekezdünk egy másik nagy témakörbe: a folyamatok közötti kapcsolattartásba (IPC).

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.

Nézzünk képeket Linux alatt!

A Linux alatti képfeldolgozás egyelőre még gyerekcipőben jár, már most is lehet azonban szinte bármilyen feladathoz jó minőségű programokat találni. A mindennapi felhasználó mindenképpen gyors, megbízható, kényelmesen beállítható és használható képnézegető programot szeretne. Aki régóta használ Linuxot vagy valamilyen Unixot grafikus felülettel, az biztosan találkozott már az `xv` programmal, ami igen nagy tudású képszerkesztő és -néző program. Bár kitűnően használható, most egy sokkal egyszerűbb és még inkább felhasználóbarát programmal ismerkedünk meg dióhéjban: a `gqview`-val.
Nagyon sok beállítási lehetőséget kínál, mégsem veszünk el a sűrűjében. Nézzük az alapvető lehetőségeket:

- A szöveges menüsor alatt látható ikonsorból a fontosabb lehetőségek azonnal elérhetőek, bekapcsolhatók. Balról az első ikon az előnézeti képek ki-, illetve bekapcsolását szabályozza. Ezt akkor érdemes használnunk, ha kevés kép van egy könyvtárban – ötszáz képnél már elég sokat kell várnunk, hogy mindet kicsiben is megnézhessünk –, vagy gyorsan szeretnénk valamit megtalálni, de a fájl nevét nem tudjuk.
- A második ikon a házikó, értelemszerűen hazavezet bennünket a saját könyvtárunkba, így elkerülhető a könyvtárakon keresztüli evickélés.
- A harmadik ikon a frissítés, ezzel újraolvastathatjuk a könyvtárat és az éppen aktív képet. Ennek akkor lehet jelentősége, ha a képen egy külső programban változtatunk vagy új képet mentünk az aktív könyvtárba.
- A következő két ikon képei önmagukért beszélnek: az elsővel a képet tetszőleges méretűre nagyíthatjuk, a következővel pedig kicsinyíthetjük.
- Az utolsó előtti ikonnal a *Beállítások* (Options) menüpontot érhetjük el közvetlenül, erről lentebb még lesz szó.
- A sorban az utolsó ikon pedig a bal oldali eszköztárat lebegteti, így szinte a teljes képernyőt használhatjuk a kép megjelenítésére (leszámítva azt a kellemetlenséget, hogy az eszköztártól ilyenkor bizonyos képrészeket nem látunk).
  A teljes képernyős képnézéshez nyomjuk meg inkább a `v` betűt, így a program valós teljes képernyős módba kapcsol.
  A szóköz billentyű előre lapoz, a BACKSPACE pedig visszafele.

A Beállítások

Itt történik a program igazi finomhangolása. A *Szerkesztés, Beállítások* (Edit, Options…) menüpontnál, vagy – mint az előbb is írtam –, az ikonsorban az utolsó előtti (villáskulcs) ikonra kattintva érhetjük el.
E helyütt tehetjük meg az aprólékos beállításokat: mi legyen az alapértelmezett könyvtár, mekkora előnézeti képet készítsen (ez a lehetőség nálam nem működött, akármekkorára állítottam is, mindig ugyanakkorák voltak a képek, csak a köztük lévő távolság nőtt), hogyan jelenítse meg a képet – gyorsabban és rosszabb minőségben vagy lassabban, de jobb minőségben –, különféle szűrési feltételeket is megszabhatunk, illetve megadhatjuk kedvenc képszerkesztőnket, hogy a kívánt kép szerkesztését azonnal el tudjuk kezdeni.

Csontos Gyula