© Kiskapu Kft. Minden jog fenntartva

Memóriakezelés a gyakorlatban

Sorozatunk előző részében csak elméletben foglalkoztunk a virtuálismemória-kezeléssel. Ennek most vége, a következőkben bemutatjuk, miként is zajlik mindez a gyakorlatban.

Példánk alanyainak valódi éles operációs rendszereket választottunk, amelyekben olyan apróságokat is figyelembe kell venni, mint a hatékonyság és a biztonság. De mindenezek előtt beszéljünk egy fontos szempontról, amit mindig figyelemben kell tartani a memóriakezelő tervezésekor. Ez pedig az úgynevezett hordozhatóság. Az elmúlt hónapban hosszasan ecseteltük, hogy a mai processzorok már nagyon okosak, és sok mindenben megkönnyítik az operációs rendszer memóriakezelőjének életét. Példát is mutattunk, mégpedig az Intel Pentium processzorcsaládot, ami amellett, hogy támogatja a szakaszolt lapozást, még a védelem kérdését is megoldja – az úgynevezett védelmi szintek bevezetésével. Ezek után azt gondolhatnánk, hogy memóriakezelőt írni „könnyű". Az biztos, hogy nem gyerekjáték, de sokat segíthet a dolgon, ha a munka oroszlánrészét Pentium processzorunkkal végeztetjük el. De ha olyan memóriakezelőt írunk, ami teljes egészében a processzor szolgáltatásaira támaszkodik, jelentősen csökken rendszerünk hordozhatósága. Csakhogy nem minden processzor képes olyan magas szintű memóriakezelésre, mint a Pentium. Sőt akad olyan is, amelyik semmi ilyesmire sem képes. Például ilyenek a 80386-osnál régebbi Intel processzorok. Ha az egész memóriakezelésünket a Pentium szolgáltatásaira alapozzuk, akkor le kell mondanunk arról, hogy a rendszerünket valaha is ennél régebbi processzoron futtassuk. Az is igaz viszont, hogy manapság ritkán van szükség olyan operációs rendszerre, aminek ilyen múzeumba illő eszközön is futtathatónak kell lennie.

(Létezik azonban olyan operációs rendszer is, amit kifejezetten arra terveztek, hogy bármilyen IBM PC-megfelelő gépen fusson, ez a Minix. Ahhoz, hogy ez lehetséges legyen, a Minix nem használ sem szakaszolást (segmentation), sem pedig lapozást. Sőt még tárcserét (swapping) sem, azaz a folyamatokat egyáltalán nem cserélgeti a lemez és a központi tár között. Még a memóriában sem mozgatja a betöltött folyamatokat, mindegyik ott marad, ahová indításakor került. Emiatt a Minix alatt sajnos nem futtathatók olyan alkalmazások, amiknek a memóriaigénye meghaladja a fizikai tár méretét. Ez egy „éles" rendszeren valóban komoly hiányosság lenne, de a Minix esetében, amit kifejezetten (oktatási célokra) kis személyi számítógépekre terveztek, nem az. Cserébe, hogy a Minix nem használ virtuális memóriakezelést, minden IBM PC-n futtatható, és könnyen átültethető más kisgépekre).

Nemcsak a régi, de az újabb processzorok virtuális memóriakezelésében is komoly különbségek vannak. Vegyük például az UltraSPARC processzorokat, amelyek 64-bites címtartománnyal bírnak. Míg a Pentium háromféle virtuális memóriakezelést is támogat (lapozás, szakaszolás, lapozott szakaszolás), addig az UltraSPARC-on kizárólag a lapozást használhatjuk.

A félreértések elkerülése végett: ez nem azt jelenti, hogy az UltraSPARC „elmaradottabb" a Pentiumhoz képest. Inkább arról van szó, hogy a Pentium 32-bites szakaszokkal (segment) dolgozik, ezért a méretük viszonylag kicsi (legfeljebb egymillió lap). Egy 64-bites címtartomány esetén a szakaszok több billió lap méretűre is duzzadhatnak, amiket nem lehet a hagyományos laptáblás módszerrel nyilvántartani. Egyébként a Pentium processzoroknál is előfordulhatna ehhez hasonló gond, ha a programoknak egyszerre több ezer szakasszal nyílna lehetőségük bűvészkedni. Szerencsére azonban sem a Windows, sem a Unixok nem engedik meg, hogy egy folyamat egy szakasznál többet birtokoljon.

A két processzor között különbség van a TLB-használatban is, pontosabban a TLB-hibák kezelésében. Míg Pentium esetében ilyenkor a TLB újrafeltöltése gépszinten megy végbe, addig a SPARC egyből az operációs rendszerhez fordul. Az észbe vésendő tanulság tehát az, hogy minél jobban kihasználjuk az adott processzor virtuális memóriakezelő szolgáltatásait, annál nehezebb lesz rendszerünket más kiépítésekre átültetni. Megfordítva: minél kevesebb dolgot tételezünk fel az adott gépről, annál jobban nő rendszerünk hordozhatósága.

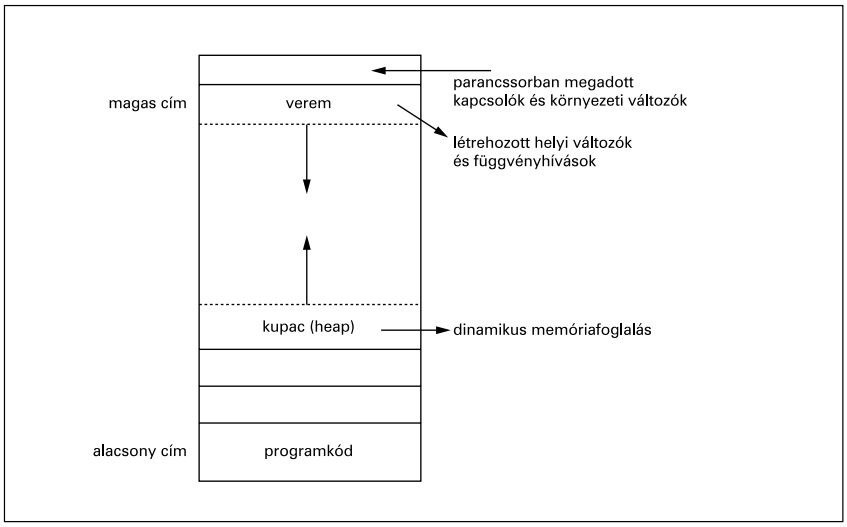
Virtuálismemória-kezelés Windows NT/2000/XP-ben

A Windows nem éppen korunk legnépszerűbb operációs rendszere, sőt bizonyos körökben egyenesen utálatnak örvend. Mi azonban mégis vagyunk olyan bátrak, hogy szóljunk néhány szót a Windowson belüli virtuális memóriakezelésről.

Lehet, hogy egy kicsit furán veszi ki magát, amiért egy, a „Linux-barátokat" megcélzó lapban Windowszal terheljük olvasóinkat. Ennek azonban részünkről megvan az alapos oka, mégpedig az, hogy az NT memóriakezelése nem rossz (most magáról az elvről beszélünk, a megvalósításról inkább nem nyilatkozunk), ezért hasznosnak találjuk – még mielőtt rátérnénk kedvenc operációs rendszerünk memóriakezelőjének boncolgatására –, hogy bemutassuk, miként is működik ez egy másik rendszerben. Fel szeretnénk hívni rá a figyelmet, hogy mi kizárólag az NT „családba" tartozó rendszerekről beszélünk, és nem a 3.1/95/98/ME nevű találmányokról.

A 3.1 igazából külön kategória, mivel gyakorlatilag csak egy MS-DOS-ra épített grafikus felület volt. A 95-től kezdve már olyan szolgáltatások is megjelentek, mint a virtuálismemória- vagy a folyamatkezelés. Az MS-DOS azonban továbbra is ott lapult legalul, tehát ha úgy vesszük, ez egyfajta kiegészítés volt, ami az MS-DOS-ból „valódi" operációs rendszert formált. Tette ezt azonban nagyon szerencsétlenül, ugyanis a 95-ös kódja 32-bites volt, ám az együttműködés megőrzése végett egy csomó 16-bitest is meghagytak, továbbá a legtöbb szolgáltatást magával az MS-DOS-sal végeztették el, például a fájlkezelést. A Windows 98 megjelenésével annyit

javult a helyzet, hogy bizonyos szolgáltatások magasabb szintre, a 32-bites kódok közé kerültek, így a rendszer üzembiztosabbá vált, de az MS-DOS-tól továbbra sem lehetett megszabadulni.
A Windows NT teljesen más, az égvilágon semmi köze sincs az MS-DOS-hoz. Saját fájlrendszerrel, memóriakezeléssel stb. rendelkezik, tehát valódi, önálló, 32-bites operációs rendszer.
Az NT kezdetben folyamatstrukturált felépítésű volt. Emlékezzünk vissza, ez azt jelenti, hogy a különböző alrendszerek önálló folyamatként futnak. Ha egy felhasználói folyamatnak szüksége van az egyik alrendszer szolgálataira, akkor üzenetet küld neki, majd a feladat elvégeztével visszakapja az eredményt, szintén üzenet formájában (ez az úgynevezett kiszolgálóalapú modell). Ennek az elgondolásnak az az előnye, hogy a rendszert rendkívül könnyen át lehet ültetni más felületekre.
A folyamatstrukturáltságnak azonban megvan az ára, mégpedig a hatékonyságbeli visszaesés. Ezért az NT 4.0-tól úgy határoztak, hogy amit lehet, bevisznek a rendszermagba, ezáltal is sebességnövekedést valósítva meg. Arról lehet vitatkozni, hogy ez mennyire számított jó döntésnek, mindenesetre a 2000 és az XP megjelenésével a Microsoft a tömbszerű (monolitikus) memóriakezelés mellett kötelezte el magát.
Mint már említettük, az NT 32-bites operációs rendszer, tehát 32-bites virtuális címtartományokkal dolgozik. Minden folyamat saját címtartományt kap, ami annyit tesz, hogy egy folyamat legfeljebb 4 GB memóriával gazdálkodhat. Valójában a program kódjának és az adatoknak csupán 2 GB áll rendelkezésükre, a felső 2 GB a rendszermag memóriájához való hozzáféréshez kell (ez csak szigorú megkötések mellett lehetséges).
Az NT a memóriát 4 KB-os laponként kezeli. A lapozáson kívül azonban a memóriakezelő sokkal több szolgáltatást képes kezelni, például az úgynevezett íráskori másolás (copy on write) módszert, vagy a fájlok memóriaként történő elérését (lásd később). Ezeket a szolgáltatásokat a felhasználói folyamatok a Win32 API-n keresztül érhetik el.
Az első és legfontosabb feladat, amit minden memóriakezelőnek végre kell tudnia hajtani, a memóriafoglalás (memory allocation). Ez a Windows esetében két lépésben megy végbe. Az első az úgynevezett foglalás (reserve), amikor magát a virtuális címteret foglaljuk le. A második az egyeztetés (commit), amikor pedig maga a virtuális memória kerül lefoglalásra.
Felmerül két kérdés: mi az érdemi különbség a két lépés között, illetve mi az értelme ennek az egésznek? Az elsőre válaszolva: a foglalás csak azt mondja meg az operációs rendszernek, hogy várhatóan mennyi memóriára lesz szükségünk, de tényleges memóriafoglalás nem megy végbe. Maga a tárterület csak az egyeztetés folyamán lesz lefoglalva. Értelemszerűen egyeztetést már csak előzőleg lefoglalt területen lehet végrehajtani, de amíg az adott területet nem egyeztettük a memóriakezelővel, addig minden hivatkozás hibát fog okozni.
Ezáltal megvalósítható, hogy egy folyamat „egyszerre több mindent csinálhasson". A szálak hasznosságára a legjobb példa a grafikus webböngésző és a sok kis képet tartalmazó weboldal esete. Ha ez az oldalt meg akarjuk nyitni a böngészőnkkel, és a böngésző minden képet egymás után töltene le, akkor lehet akármilyen széles sávú elérésünk, a weben való böngészés nem lesz az örömök forrása. Ha azonban minden kép letöltését külön szál segítségével oldjuk meg, akkor amíg a kiszolgálótól visszaérkező válaszra várunk, bőségesen jut idő a többi kép letöltésére is.
A nehézség annyi, hogy minden szálnak saját veremmel kell rendelkeznie, aminek egy folyamatos címtartományon kell lennie. Az NT általában 1 MB-nyi vermet foglal minden szálnak. Ha a foglalás azonban egy lépésben menne végbe, akkor minden létrehozott szál elvisz 1 MB-ot a fizikai memóriából, miközben valószínű, hogy ennek az egytizedét sem fogja kihasználni. Például egy olyan weboldal letöltéséhez, ami tíz 4 KB-os képet tartalmaz, csak a szálak verméhez 10 MB-nyi memóriára lenne szükségünk.
Ezzel szemben a kétlépcsős megoldás esetén az 1 MB-ot csak lefoglaljuk (de nem egyeztetjük), egyeztetni csak kétlapnyi memóriát fogunk. A verem teteje az első lap lesz, a második azért szükséges, hogy tudjuk, mikor éri el a verem azt a méretet, amikor további lapok egyeztetésére is szükség lesz.



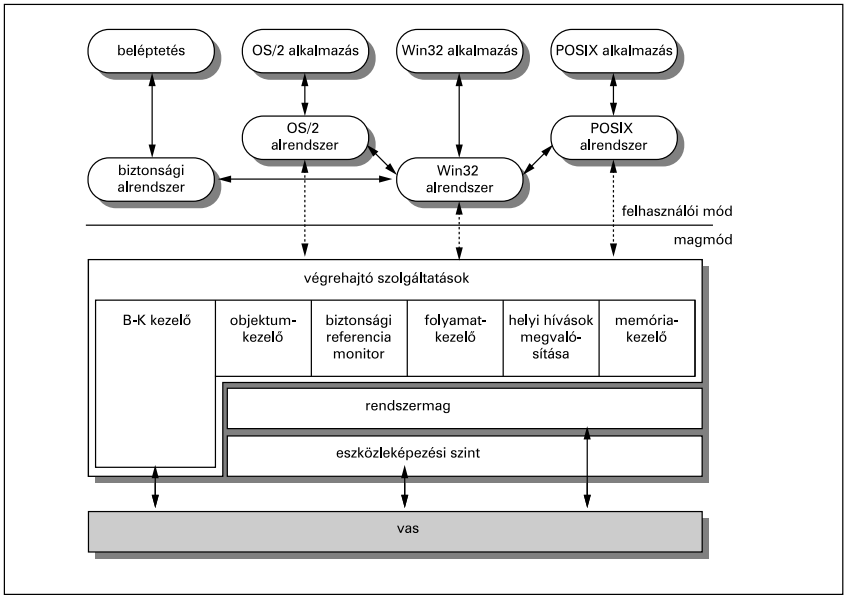
1. ábra
A folyamat rendelkezésére álló memóriarészt három (pontosabban négy) részre osztjuk fel. Legalulra kerül a programkód, utána jönnek az adatok és legfelülre a verem. Az adat és a verem egymással ellentétes irányban nő, a folyamat indításakor átadott értékek, illetve a környezeti változók pedig a verem fölé kerülnek

Vajon mit nyerünk ezzel? Hatékonyabb működést! A folyamatok futtatásának fizikai memóriaigénye ugyanis csökken, mivel tényleges erőforrás-használatra csak az egyeztetés után kerül sor. Nem is beszélve arról, hogy a rendszer terhelése nélkül foglalhatunk le nagy terjedelmű és egybefüggő címtartományokat.
Ha netán kétséges lenne a kétlépcsős memóriafoglalás előnyös mivolta, gondoljunk a szálak veremkezelési kérdésére. A vezérlési szálakról (vagy más néven a pehelysúlyú folyamatokról) már szó esett valamikor nagyon régen (Linuxvilág 2002. szeptemberi szám). Akkor azt mondtuk, hogy a szálak utasítások sorozatából álló valamik, amiket a gép majd szépen sorban végrehajt. Azt is mondtuk, hogy minden folyamatban legalább egy ilyen szálnak kell lennie, de egyes rendszereken több is lehet.

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

Ennek köszönhetően az előző példában említett szálak fizikailag csak 8 KB-ot fognak felfalni.
Érdemes szólnunk pár szót az NT lapkezeléséről is. Minden lapnak három állapota lehet: szabad, foglalt és egyeztetett. Ha egy lap szabad, akkor semmilyen adat sincs rá leképezve, és egy ilyen lapra való hivatkozáskor mindig hiba történik. A foglalt lapokra nem képezhető adat, nem lehet rájuk hivatkozni, és ez az állapot egészen addig így marad, amíg a lekötöttség meg nem szűnik.
Az egyeztetett lapokra történő hivatkozások a vas segítségével képeződnek le. Ez a művelet csak akkor sikeres, ha az éppen a memóriában tartózkodik. Ha nincs benne, akkor laphiba történik, és az operációs rendszernek a lapot be kell töltenie a lemezről. Minden memóriában lévő egyeztetett lapnak létezik egy mása a lemezen, amelyet az adott lap árnyéklapjának nevezünk (ide kerülnek majd, ha kivesszük őket a memóriából). Az árnyéklapok úgynevezett lapozófájlok formájában vannak tárolva. Fontos, hogy a foglalt, illetve a szabad lapokhoz nem tartozik árnyéklap (így amikor rájuk hivatkozunk, a rendszer nem tud mit betölteni a lemezről, tehát hiba keletkezik – értsd: kék képernyő).
Az NT azt is támogatja, hogy akár egy állományt is leképezhessünk a virtuális címtartományra. Ezt a következőképpen képzelhetjük el: tegyük fel, van egy 10 bájtos állományunk, és azt képeztük le a 140–150 virtuális címtartományra. Ezután, ha a folyamat kiolvassa a 140-es memóriacímen található adatot, akkor a fájl első bájtját kapja vissza (írásnál ugyanígy). Magyarán anélkül tudunk írni, illetve olvasni a fájlból, hogy bármilyen B-K műveletet hajtanánk végre. Fontos megjegyezni, hogy gyakorlatilag semmi különbség nincs az egyeztetett lapok és egy állomány virtuális címtartományba való leképezése között. Egyszerűen csak arról van szó, hogy az utóbbi esetben az árnyéklapok nem valamelyik lapozófájlban, hanem magában az állományban foglalnak helyet. (Az is igaz, hogy ilyenkor könnyen előfordulhat olyan eset, hogy a memóriában más van, mint a lemezen. Ez azonban az állomány frissítésével könnyen orvosolható. Ezt az NT a `flush` nevű függvény segítségével oldja meg).
Az NT lehetőséget ad az osztott memória használatára is. Ez azt jelenti, hogy ugyanazt az állományt két vagy több folyamat is leképezheti saját címtartományába, vagy ugyanazt az állományt különböző virtuális címekre. Ez a módszer gyors megoldást kínál a folyamatok közötti kapcsolattartásra, mivel az osztott lapokon az egyik folyamat által előidézett változtatásokat az összes többi folyamat azonnal látja.
Az NT által támogatott megosztás másik formája az íráskori másolás (copy on write). Ez azt jelenti, hogy az osztott lapokat bármelyik folyamat bármikor olvashatja. Ha azonban valaki írni szeretne, akkor laphiba lép fel, és a rendszer a folyamat számára egy másolatot hoz létre a módosítandó lapról. Az érdemi különbség a két módszer között az, hogy ebben az esetben az összes folyamat úgy érzékeli, mintha egyedül ő képezte volna le az adott fájlt.



2. ábra
A Win32 alrendszer felhasználói módban fut és vele tartanak közvetlen kapcsolatot a windowsos alkalmazások. A többi (Posix és OS2) alrendszer is a Win32-re épül, de az ezeket használó alkalmazásoknak nincs lehetőségük közvetlenül a Win32-höz fordulniuk

Rendszerhívások a Windows NT-ben

Érdemes pár szót szólnunk arról is, hogy a felhasználói folyamatok (vagyis az alkalmazások) miképpen érhetik el az operációs rendszer szolgáltatásait.
Az NT esetében az alkalmazások rendszerhívásokat nagyon ritkán hívnak közvetlenül. Ennek talán a legfőbb oka az, hogy a Microsoft módszeresen elhallgatja az NT rendszerhívásainak a listáját (pontosabban nem teszi közzé őket teljes egészükben, és a különböző változatok megjelenésével folyamatosan módosítgatja őket). Az alkalmazások ezért a környezeti alrendszerekhez (environmental subsystems) fordulnak segítségül, amelyek különböző függvények hívását teszik lehetővé.
A legfontosabb alrendszer a Win32, amelyet a Windows-alkalmazások hívnak, de NT alá létezik egy Posix és egy OS/2 alrendszer is a Unix, illetve az OS/2-ből átültetett programok számára. (Az igazat megvallva a Posix alrendszer az NT-ben fabatkát sem ér. Egyrészt nem önálló alrendszer, a Win32-re támaszkodik, de az ezt használó alkalmazások számára nem teszi az összes Win32 függvényt elérhetővé, magyarán szólva a Posix alrendszerre támaszkodó programok nem tudnak szabadon Win32-es függvényeket hívni). Emiatt már-már lehetetlen egy komolyabb unixosos alkalmazás NT alá történő átültetése. Valószínűleg azért születhetett meg mégis ez a Posix rendszer, mert az Egyesült Államokban az állami intézményekben csak olyan rendszereket lehet használni, amelyek tartalmazzák a Posix-megfelelőséget. (A Microsoft nyilván nem akart elesni ettől a piactól).
A rendszerhívások nyilvánossá tétele helyett inkább egy felhasználói kezelőfelületet (interface) hoztak létre, amit Win32 API-nak (Application Programming Interface) neveznek. Ez jól van dokumentálva, és hihetetlen mennyiségű függvényt tartalmaz. Ezek a függvények vagy meghívnak egy rendszerhívást, vagy a Win32 alrendszerhez fordulnak, de a legtöbb a felhasználói módban maradva hajtja végre a feladatát.
A Win32 API eszmeisége kicsit furcsa (azok számára, akik nem Windowson

nevelkedtek). Míg a legtöbb operációs rendszer (például a Unix) esetében arra törekednek, hogy minél kevesebb rendszerhívás legyen (pontosabban csak annyi, amennyi feltétlenül szükséges ahhoz, hogy a rendszer minden szolgáltatását kényelmesen elérhessük), addig a Windowsra a halmozás a jellemző. Ez alatt azt értjük, hogy arra törekednek, hogy minél több függvénnyel árasszthassák el a programozót, amik fedjék le úgymond az „élet minden területét", és ugyanazt a dolgot több különböző módon is el lehessen végezni. Például külön API-függvény létezik arra, hogy egy fájlt teljes egészében egy másik helyre másoljunk át.
(A Win32 API egyébként a 95/98/ME rendszerekben is megtalálható, igaz, egy kicsit kevesebb szolgáltatással. Például nem érhetők el a különböző biztonsági szolgáltatások, mivel ilyesmit ez a rendszer egyáltalán nem tartalmaz, továbbá bizonyos függvényeket másképpen kell paraméterezni).

Virtuálismemória-kezelés a Unixban

Manapság rengeteg Unix van forgalomban, ezért elég nehéz általánosságban beszélni róluk. A következőt fogjuk tenni: kiválasztunk egyet közülük (legyen mondjuk a Linux), és annak a virtuális memóriakezelését fogjuk töviről-hegyire bemutatni. Ez lesz a következő rész témája.
Most azonban – bevezetésként – megnézzük, hogyan is kezeli a virtuális memóriát a Berkeley Unix.
Az első kérdés: mi az a Berkeley Unix? A történet az 1970-es évek elejére nyúlik vissza, amikor is egy Bell Labs nevű intézményben megszületett az első Unix, a labor tulajdonosa az AT&T volt. Ez a rendszer egy PDP-11 nevű gépen futott, amelyek az Egyesült Államok egyetemein nagy népszerűségnek örvendtek. Ezért az AT&T a Unixot forráskódostul olcsón az egyetemek rendelkezésére bocsátotta, azok pedig egyből kaptak az alkalmon (mivel a PDP-11 eredeti operációs rendszere hihetetlenül használhatatlan volt).
A hangsúlyt arra helyeznénk, hogy a Unixot a forráskóddal adták oda, tehát minden egyetem kedvére továbbfejlesztgethette. Így történt ez a kaliforniai egyetemen is (ez a Berkeley). Itt vezették be a lapozott virtuális memóriát, és ez volt az első rendszer, ami támogatta a TCP/IP-n keresztüli kapcsolattartást, amely később az egész Internet alapjául szolgált.
(Eközben az AT&T is tovább fejlesztette a Unixot, ebből lett a System V. Mindkét Unix-változatról elmondható volt, hogy egyik sem működött együtt a másikkal. A helyzet orvoslására létrehozták a Posix – Portable Operating System-IX – nevű szabványt, ami meghatározta a rendszerhívásokat és az alapvető segédprogramokat. A Posixban meghatározott rendszerhívásokon kívül számos Unix-változat más rendszerhívásokat is lehetővé tesz, de ami a Posixban van, az mindenhol megtalálható).
A Berkeley Unix a következő memóriamodellt használja: minden folyamathoz három szakasz tartozik. Az elsőben a kód, a másodikban az adat és a harmadikban a verem foglal helyet (ezek közül a kódszakasz mérete biztosan nem fog változni a folyamat futása során).
Ha az adott gép nem támogatja a szakaszolást (csak egy virtuális címtérrel rendelkezik), akkor a kódot berakják a memória aljára. A legtetejére a szakasz kerül, középre pedig az adat. Mindez úgy helyezkedik el, hogy az adat és a verem egymással ellentétes irányban növekedhessen.
Ha az a gép, amin a rendszer fut, képes a lapozásra, akkor nyugodtan lapozható az egész címtartomány, hiszen a folyamatok ebből semmit se fognak észrevenni. Ha nincs lehetőség a lapozásra, akkor a folyamatokat sajnos csak teljes egészükben cserélhetjük a lemez és a központi tár között. Ez az úgynevezett tárcsere (swapping), amiről már szó volt két résszel ezelőtt. Az effajta memóriakezelésnek az a legnagyobb előnye, hogy szinte minden kiépítésen könnyedén megvalósítható.
A jelenlegi Unixok azonban ennél többre is képesek. Például a Solaris (ami a System V-re épül) memóriakezelője szintén képes olyan dolgokra, mint az NT-é. Itt is lehetőség nyílik az állományok virtuális címtérbe való leképezésére vagy az íráskori másolás módszerének használatára.
Sorozatunkat a következő alkalommal a Linux memóriakezelésének részletes ismertetésével folytatjuk.

➲ http://www.eg3.com/adv-goo/posix.htm

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.

© Kiskapu Kft. Minden jog fenntartva