TERMINOLÓGIA

Erdélyi Magyar Műszaki Tudományos Társaság

Magyar nyelvű szakelőadások
a 2001-2002-es tanévben

Kolozsvári Műszaki Egyetem
Számítástechnika Kar

Szerzők:
Barabás Tibor
Dr. Buzás Gábor
Enyedi Szilárd
Dr. Gacsádi Sándor
Sebestyén Pál György
Somodi Zoltán
Dr. Vári Kakas István

Kolozsvár, 2002

Támogató:
Apáczai Közalapítvány – Budapest

Kiadó:
Erdélyi Magyar Műszaki Tudományos Társaság

Felelős kiadó:
Égly János

Szerkesztő:
Sebestyén Pál György

Nyelvi lektor:
Matekovits Hajnalka

Borítóterv:
Prokop Zoltán

Tartalomjegyzék

Memória struktúrák teljesítményének mérése

Somodi Zoltán, gyakornok
Kolozsvári Műszaki Egyetem, Számítástechnika és Automatizálás Kar,
Számítástechnika Tanszék

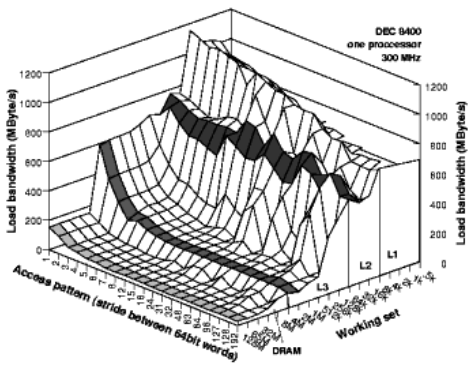
1. Bevezetés

A számítógépek teljesítménynövekedésének egyik akadálya a processzor és a memória sebessége közötti szakadék. Az múlt évtizedben a mikroprocesszorok órajele 40%-kal nőtt évente, míg a memória (DRAM) sebessége csak 11% körüli évenkénti növekedést ért el. Hagyományos megoldás erre a problémára a gyorstárolók (cache) kialakítása, ahol azokat az adatokat és utasításokat tárolják, amelyeket a processzor gyakran használ, így gyors elérést biztosítva számára.

Ezek a tendenciák nagyon érzékennyé teszik a számítástechnikai rendszerek összteljesítményét a memória-architektúrák legkisebb változására is. Így a memória rendszerek tervezői rendkívül érdekeltek olyan módszerek kidolgozásában, amellyel meg tudják határozni a különböző rendszerek teljesítményét még a tervezés fázisában.

De mitől is függ egy memória-rendszer teljesítménye? A Zürichi Műszaki Egyetem kutatói olyan programot fejlesztettek ki, amely méri, és grafikusan ábrázolja a memória adatátviteli sebességét. A *MemPerf* program két paramétert vezet be:

- *terhelés (working set)*, amely kilobájtban méri a memóriába írt vagy olvasott adattömb méretét;
- *hozzáférési minta (access pattern),* amely azt mutatja, hogy az egymást követő memória-hozzáférések mennyire történtek egymás utáni címekre. A hozzáférési mintát a *lépés (stride)* határozza meg, amely azt mutatja, milyen távol van két egymást követő hozzáférés címe egymástól. Ha a lépés 0, akkor a címek egymás mellett vannak, tehát a hozzáférést folytonosnak mondjuk. Ha a lépés n, akkor a két cím között n*64 bit távolság van.



1. ábra. *Adatátvitel különböző hozzáférési mintákkal*

A MemPerf a fenti paraméterek értékének változtatása mellett méri a memória és a processzor közötti adatátviteli sebességet megabájt/s-ban. Így kapjuk meg az *1. ábrán* látható 3 dimenziós képet, amely az adatátviteli sebességet ábrázolja a terhelés és a hozzáférési minta függvényében. A mérés egy 300 MHz-es DEC 8400-as processzorral működő rendszeren készült, amelyben a fő memóriaegységen kívül (DRAM) még egy 3 szintű gyorstároló–hierarchiát is találhatunk.

Amint az várható volt, az adatátvitel akkor történik a maximális sebességen (kb. 1100 MB/s), ha a processzorhoz legközelebb álló L1 cache-ből olvasunk egymás utáni címekről adatokat (vagyis mikor a lépés nulla). Ez akkor történik meg, mikor az olvasott adattömb olyan kicsi, hogy teljes egészében belefér az L1 cache-be. Az ábrán jól látható az is, hogy ha a terhelés nagy, vagyis nagyobb adattömböt olvasunk a memóriából, akkor az már nem fér el ugyanazon a memóriaszinten, és szükség van a következő szint eléréséhez is. Tehát minél nagyobb a terhelés, annál kisebb az átviteli sebesség. A 700 MB/s, 120 MB/s és a 28 MB/s szintjén húzódó vízszintes síkok mutatják a memóriarendszer egyes szintjeinek teljesítményét különböző terheléseknél. A szürke sávok jelzik a legnagyobb terhelés mellett mért teljesítményt az L2, L3 illetve DRAM hozzáférések esetében. Ezen sávok meredek csökkenése azt jelenti, hogy az adatátvitel sokkal jobb folytonos és kis lépésű minták esetében.

A fenti példából kiindulva is láthatjuk, hogy egy memória-rendszer teljesítménye nagyon függ a hozzáférési mintától. Tehát mikor egy memória-rendszer teljesítményét akarjuk mérni különböző alkalmazások futása közben, először meg kell határoznuk azt, hogy az illető alkalmazás milyen minta szerint használja a memóriát, a memória-kérések hány százaléka használja az L1 cache-t, L2 cache-t illetve a fő memóriát. Erre több módszer is létezik. A következő fejezetekben két ilyen módszer kerül bemutatásra. Az első a Pentium processzorok teljesítmény-figyelő lehetősége, a másik pedig a nyomvezérelt memóriaszimulálás.

2. Teljesítmény-figyelő Pentium processzoroknál

A modern processzorokban léteznek olyan speciális regiszterek, amelyek segítségével különböző események megjelenésének számát kaphatjuk meg.

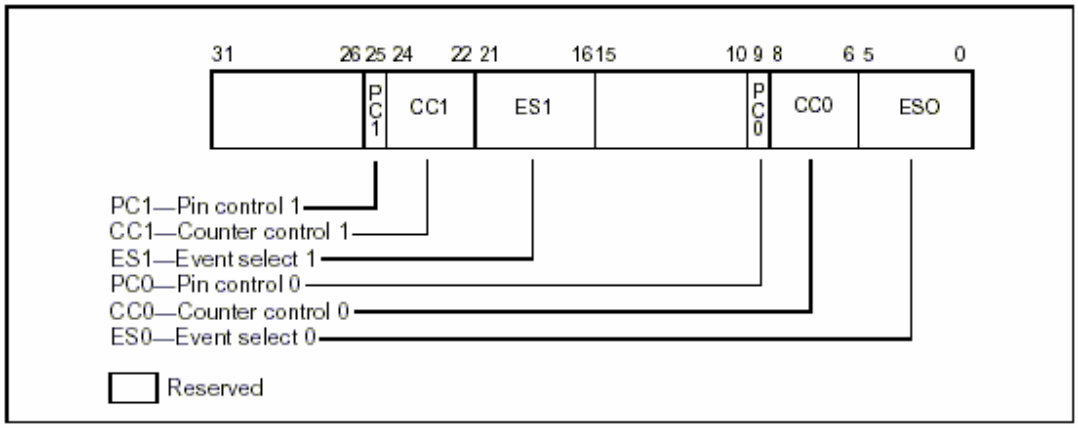
A teljesítmény-figyeléssel először az IA-32 architektúrájú Pentium processzoroknál találkozhatunk, amely egy sor teljesítmény-figyelő számlálóval (*performance-monitoring counter)* rendelkezik. Ezek a számlálók lehetővé teszik, hogy figyeljük és mérjük a processzor különböző paramétereit. Az így kapott információt fel tudjuk majd használni a rendszer beállításánál, illetve a kompilátorok teljesítményének javításánál.

A Pentium processzor két 40 bites teljesítmény-számlálóval rendelkezik, amelyek eseményszámlálásra és időtartammérésre is alkalmasak. Ezeket a számlálókat 3 modell specifikus regiszter (*Model Specific Register – MSR)* segíti: egy kontrol- és eseménykiválasztó-regiszter (*Control and Event Select Register – CESR*) és két teljesítményszámláló-regiszter (CTR0 és CTR1). Ezen regiszterek tartalmát a RDMSR és a WRMSR utasításokkal olvashatjuk illetve írhatjuk, de csak, mikor a processzor a 0. privilegizált szinten (privilege level 0) működik. Minden számlálónak van egy hozzárendelt külső lába (PM0/BP0 és PM1/BP1), amelyet a számláló állapotának lekérdezésére használhat egy hozzákapcsolt hardver-eszköz.

A teljesítményszámlálók műveleteit a 32 bites CESR regiszterrel választhatjuk ki, amelynek
3–3 mezője van a két számláló számára:

- *ES0 és ES1* (Event Select): 6 bites eseménykiválasztó mező, amellyel kiválaszthatjuk a mérni kívánt eseményeket. A fontosabb események kódjait az *1. táblázatban* foglaltuk össze.
- *CC0 és CC1* (Counter Control): 3 bites számláló-vezérlő mező, amellyel a számlálók működési módját adhatjuk meg. A lehetséges vezérlő-kódok a következők:
  - 000–számláló kikapcsolva
  - 001–számlálja a kiválasztott eseményt a 0, 1 vagy 2-es privilegizált szinten
  - 010–számlálja a kiválasztott eseményt a 3-as privilegizált szinten
  - 011–számlálja a kiválasztott eseményt privilegizált szinttől függetlenül
  - 100–számláló kikapcsolva
  - 101–időtartamot mér (órajelet számlál) 0, 1 vagy 2-es privilegizált szinten
  - 110–időtartamot mér 3-as privilegizált szinten
  - 111–időtartamot mér privilegizált szinttől függetlenül
- *PC0 és PC1* (Pin Control): A CESR 9-es illetve 25-ös bitje, amellyel a külső eseményfigyelő-számláló láb (PM0/BP0 ill. PM1/BP1) működését határozhatjuk meg. Ha ennek a bitnek 1 az értéke, a láb akkor kerül aktív állapotba, ha a számláló túlcsordult. Ha a bit értéke 0, az aktív állapot azt jelenti, hogy a számláló értéke 1-gyel nőtt (inkrementálódott).

A CESR regiszter mezőfelosztását a *2. ábrán* tekinthetjük meg.



2. ábra. *A CESR regiszter Pentium processzoroknál*

Azon események, amelyeket a processzor számlálni és rögzíteni tud a CTR0 és CTR1 regiszterekben két kategóriába csoportosíthatók: előfordulás- és időtartam-események. Az előfordulás-események 1-gyel növelik a regiszter értékét minden előfordulásuk alkalmával. Ha egy óracikluson belül kétszer jelentkezik az esemény, a regiszter értéke 2-vel növekszik.

Az időtartam-események esetében a számláló azokat az órajeleket számlálja, amelyek alatt egy megadott feltétel igaz.

Az alábbi táblázatban néhány fontosabb eseményt foglaltunk össze, amelyet a Pentium processzorokkal mérni lehet.

1. táblázat. *Pentium processzorok fontosabb mérhető eseményei*

Eseménykód	Esemény neve	Leírás
00H	DATA_READ	Memóriából való olvasások száma (belső cache-hozzáférést is beleszámolva).
01H	DATA_WRITE	Memóriába való írások száma (belső cache-hozzáférést is beleszámolva, I/O műveleteket kivéve).
02H	DATA_TLB_MISS	Találat nélküli adat-cache puffer (TLB) hozzáférések száma.
03H	DATA_READ_MISS	Azon memória-olvasások száma, amelyek esetében az adatot nem lehetett a belső cache-ből kiolvasni (attól függetlenül, hogy az adat tárolható-e vagy sem a cache-ben).
04H	DATA_WRITE_MISS	Azon memória-írások száma, amelyek esetében az adatot nem lehetett a belső cache-be írni (attól függetlenül, hogy az adat tárolható-e vagy sem a cache-ben).
05H	WRITE_HIT_TO_ M-_OR_E-STATE_LINES	Írástalálatok száma az adat-cache-ben exkluzív vagy módosított állapotú cache-sorok esetében.
06H	DATA_CACHE_ LINES_ WRITTEN_BACK	Az összes „mocskos” sor száma, ami visszaíródott, a visszaírás okától függetlenül.
0CH	CODE READ	Olvasott utasítások száma, attól függetlenül, hogy az utasítás tárolható-e vagy sem a cache-ben.
0EH	CODE CACHE MISS	Azon olvasott utasítások száma, amelyek nem voltak az utasítás-cache-ben.
0FH	ANY SEGMENT REGISTER LOADED	Szegmens regiszterek (LDTR, GDTR, IDTR és TR) írásának száma valós vagy védett üzemmódban.
12H	BRANCHES	Végrehajtott és végrenemhajtott elágazások száma (beleértve a feltételes elágazásokat, ugrásokat, függvényhívásokat, visszatéréseket, software megszakításokat és megszakítás-visszatéréseket).
14H	TAKEN_BRANCH_ OR_BTB_HIT	Végrehajtott elágazások és a BTB-találatok száma.
15H	PIPELINE FLUSHES	Pipeline kiürítések száma.
16H	INSTRUCTIONS_ EXECUTED	Végrehajtott utasítások száma (max. 2 utasítás egy órajelen belül).
18H	BUS_CYCLE_ DURATION	Órajelek száma, amíg a sínciklus folyamatban van.
1DH	I/O READ OR WRITE CYCLE	Sínciklusok száma egy I/O hozzáférés esetén.
22H	FLOPS	Lebegőpontos műveletek száma.

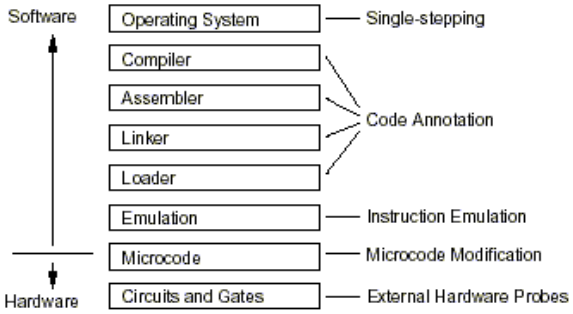
3. Nyomvezérelt memóriaszimulálás

Egy másik módszer a memóriarendszerek teljesítményének mérésére az ún. nyomvezérelt memóriaszimulálás (*trace-driven memory simulation)*. A módszer lényege egy olyan program írása, amely szimulálja a memória működését. Ezután a programnak bemenő adatként egy memória-hozzáférési szekvenciát adunk, amellyel utánozhatjuk azt, ahogy egy valós processzor használná a memóriát. A szekvenciát *cím-nyomnak* (*address-trace*), vagy egyszerűen *nyomnak* hívjuk. Az első látásra egyszerűnek tűnő módszernek sok akadálya lehet a gyakorlati alkalmazásnál. Nehéz feladat egy teljes és részletes nyomot felépíteni, főleg mikor egy bonyolult programról van szó, ami több folyamatból áll. Egy másik nehézség a nyomok nagysága, amely meghaladhatja a több gigabájtos méretet is. Végül pedig a szimulálás is egy nagyon időigényes feladat.

A nyomvezérelt memóriaszimulálást három főbb részre bonthatjuk: *nyomalkotás, nyomcsökkentés* és *nyomfeldolgozás.* Az alábbiakban röviden ismertetjük e három műveletet.

3.1. Nyomalkotás

A nyomalkotás az a folyamat, amely során meghatározzuk és rögzítjük egy program által végzett memória-hozzáféréseket illetve azok címeit. A nyom minőségét mérhetjük a nyom *teljességében* és *részletességében* vagy a *torzítás* mértékében, amit tartalmaz. Egy *teljes* nyom magába foglalja az összes memória-hozzáférést, amelyet egy program minden komponense végzett, beleértve minden felhasználószintű folyamatot és az operációs rendszer magját is. Akkor mondjuk, hogy egy nyom *részletes*, ha az egyszerű címek mellett más információkat is tartalmaz a hozzáférésekről, mint például a laptáblázatban beállt változásokat, vagy egy címkét, amin a hozzáférés milyensége (olvasás, írás, végrehajtás), mérete (szó, félszó, bájt) és a keletbélyegző jelenik meg. A nyom nem *torz*, ha az nem tartalmaz más, számunkra érdektelen memóriacímeket, amelyeket más folyamatok alkottak.



3. ábra. *Címnyomalkotás módszerei*

A címnyomot egy rendszer különböző absztrakciós szintjein alkothatjuk kezdve az áramköri és mikrokód szintről egészen a kompilátor és operációs rendszer szintjéig. A *3. ábra* ezt szemlélteti.

Külső hardver minták segítségével úgy alkothatjuk meg a címnyomot, ha a program futása alatt a címsínről egyenesen levesszük a címeket elektromos impulzusok formájában. Ezeket először egy külső pufferben tároljuk, majd ennek betelésekor a tartalmát kimentjük egy hagyományos tárolóra (szalag vagy lemez). Ennek a módszernek az egyik fő hátránya, hogy a modern processzorok esetében nem kapunk teljes nyomot, mivel ezeknél az első szintű cache-memória a chip-en található. Így a külső mintavételezésnél nem észlelhetjük a cache-hozzáféréseket. Egy másik hátrány, hogy a tárolt címnyomok nem mindig könnyen értelmezhetőek, nehéz megtudni a hozzáférés milyenségét és méretét. A módszer fő előnye viszont az, hogy teljes, torzítás nélküli nyomokat alkothatunk vele.

A külső mintavételezés magas ára arra késztette a kutatókat, hogy magasabb absztrakciós szinteken próbáljanak nyomot alkotni. Egy ilyen alternatíva a hardver és szoftver határvonalán való nyomalkotás. A módszer lényege a *mikrokód módosításában* áll úgy, hogy az utasítás végrehajtásának mellékhatásaként a memória egy bizonyos részébe íródjon be a hozzáférési cím, amit az illető utasítás használt. Így nagyon gyorsan teljes nyomhoz juthatunk. Ez viszont azzal jár, hogy a nyom kissé eltorzul, mikor betelik a memória, és ugyanakkor 10-20-szor csökkenti az utasítások végrehajtási idejét. Ezen kívül a modern mikroprocesszorok esetében a mikrokód a chipen van, így elég nehéz ahhoz hozzáférni és azt módosítani.

Egy szinttel fennebb lépve az *utasításkészlet emulálását* használhatjuk a nyom létrehozására. Az utasításkészlet-architektúra (*Instruction Set Architecture – ISA*) egy olyan

utasításgyűjtemény, amely megteremti a kapcsolatot egy számítógép hardvere és szoftvere között. Ebben az esetben ezt az utasításkészletet kell úgy módosítani, hogy az utasítások végrehajtásánál mellékhatásként memória-nyom keletkezzen. Ennek következtében a végrehajtási idő 10.000-szeressére is nőhet, ami a módszer fő hátránya. Emellett az így keletkezett nyom nem is teljes, mivel csak egy folyamat memória-hozzáféréseit figyeli, nem számítva az operációs rendszer által végzett hozzáféréseket. Mégis a módszer rugalmassága és könnyű használhatósága miatt népszerűvé vált a kutatók körében.

A leggyakrabban használt módszer viszont a *statikus kódmegjelölés*. Ezúttal magát a program forráskódját változtatjuk meg. Azon utasítások köré, amelyek memória-hozzáférést végeznek, olyan új utasításokat szúrunk be, amelyek valamilyen kimeneti eszközre (képernyő, lemez stb.) kiírják a hozzáférés céljának címét. Így egy új forráskód keletkezik, melyet lefordítva és futtatva létrehozzuk a címnyomot. A kódmegjelölést végezhetjük forrás szinten (az assembly állományt módosítva), objektum-modul szinten vagy futtatható (bináris) szinten. E módszer fő előnye az implementálás könnyedsége. Hátránya viszont, hogy nem minden program esetében van hozzáférésünk a teljes forráskódhoz. Egy másik hátrány, hogy a nyom részletessége nehezen kezelhető, sokszor „mindent vagy semmit” helyzetbe kerülünk. Végül, ezt a módszert nem használhatjuk többfolyamatos programok esetében, és nem kapunk információt az operációs rendszer memória-hozzáféréseiről.

A legmagasabb absztrakciós szinten van a *lépésenkénti végrehajtás* módszere. Előnyként sorolhatjuk fel a módszer olcsóságát, hordozhatóságát és könnyű használatát. A módszert lassúsága és korlátozottsága (csak egy folyamatot tud figyelni) miatt napjainkban nem nagyon használják.

3.2. Nyomcsökkentés

A nyom, megalkotása után, készen áll a feldolgozásra. Ez tulajdonképpen azt jelenti, hogy a generált nyomot bemenő adatként szolgáltatjuk egy memóriaszimulátornak. Sokszor viszont arra van szükségünk, hogy a nyomot tároljuk, és majd később dolgozzuk fel. Figyelembe véve, hogy egy modern mikroprocesszor, amely 1 GHz-es órajellel dolgozik, könnyen generálhat másodpercenként 50 gigabájt nyomot, láthatjuk, hogy fontos a nyomok csökkentése, hogy ezáltal csökkentsük a tárolási és feldolgozási szükségleteket. Szerencsére a címnyomok nagy helybeli és időbeli lokalitással bírnak, így sok lehetőség kínálkozik a csökkentésükre.

A módszerek bemutatása előtt nézzük meg, milyen kritériumok szerint lehet értékelni a különböző technikákat. Első, természetesen, a *csökkentési tényező*. Nem elhanyagolható a teljes nyom *újraalkotási ideje* sem. Végül pedig meg kell említenünk a *sűrítés tisztaságát*, vagyis azt, hogy a csökkentett nyomból milyen mértékben lehet újra előállítani a teljes nyomot.

Az első és legegyszerűbb módszer a nyomcsökkentésre a hagyományos *adat-sűrítő algoritmusok* alkalmazása. Ilyen algoritmusok pl: LZV, Huffman kódok stb. Az algoritmustól függően 10-100-szoros csökkentést tudunk elérni, de az újraalkotási idő is 100-200-szorosára nő.

Egy érdekes és gyakran használt módszer az ún. *jelentős események nyoma*. Ezt a statikus kódmegjelölésnél használhatjuk úgy, hogy a megjelölt forráskód csak a jelentős dinamikus események nyomát alkossa a teljes nyom helyett. Ehhez először elemeznünk kell az eredeti

forráskódot, és meg kell találnunk azokat az utasításokat, amelyek hozzájárulnak a címszámításhoz. Ezután meg kell határoznunk azokat a címeket, amelyek könnyen kiszámolhatóak, és azokat, amelyeket nehezen vagy egyáltalán nem lehet előállítani egy statikus elemzés során. A teljes nyom előállításánál a csökkentett nyomból az eredeti, nem megjelölt forráskód segítségével előállíthatjuk azokat a címeket is, amelyek nem szerepelnek a nyomban.

Egy példán keresztül mutatjuk be ezt a módszert.

Eredeti forráskód	**Jelölt forráskód**
1: sub r6, r3, r9	sub r6, r3, r9
2: add r1, r6, r8	add r1, r6, r8
3:	add t1, r1 ,4
4:	st t1, trace(t2)
5:	add t2, t2, 4
6: ld r2, 4(r1)	ld r2, 4(r1)
7: add r5,r6,r2	add r5,r6,r2
8: st r5, 8(r1)	st r5, 8(r1)

A címet a 2. sorban állítjuk elő az *r1* regiszterben. Mielőtt megtörténne a memória-hozzáférés, tároljuk a *trace* tömbben az *r1+4* címet. A 7. sorban végrehajtunk egy műveletet, de ezzel nem változtatjuk meg az *r1* értékét. A 8. sorban ugyanazon *r1* regiszter segítségével címezzük meg a memóriát, így már nem kell újra tárolni a címet.

A módszer előnye, hogy a nyomalkotás és a nyomcsökkentés folyamatát egybeolvasztja egyenesen csökkentett nyomot alkotva. Hátránya viszont az, hogy megnöveli, sokszor több mint 1000-szeresére az újraalkotás idejét.

Vannak esetek, mikor a nyom csökkentését úgy érhetjük el, hogy egyszerűen kiveszünk egyes címeket belőle. Ezt a módszert *nyomszűrésnek* hívják. Például, ha a nyomot csak cache-szimulációra akarjuk használni, amelynek van egy bizonyos mérete, akkor számunkra érdektelenek azok a címek, amelyek a méreten kívül esnek. Ezzel a technikával többszörösére csökkenthetjük a nyomot, de sajnos nem minden esetben alkalmazható.

Ha egy nagyon nagy vagy végtelen adathalmazt kell elemeznünk, sokszor hasznos lehet, ha statisztikai módszerekkel választunk ki egy részhalmazt, vagy *mintát*. *Nyom-mintavételezéssel* alkotott minták felhasználhatóak arra, hogy megítéljük néhány statisztikai adat értékét anélkül, hogy feldolgoznánk az egész nyomot. A módszernek két változata ismert: idő- és térbeli mintavételezés. Az első esetben egy időtartamot határozunk meg, amíg a mintavételezés tart. A második esetben a memóriának (címtérnek) csak egy bizonyos részébe eső címeket (pl. egy cache sor) tartjuk meg a mintában.

3.3. Nyomfeldolgozás

A nyomvezérelt szimulálás végső célja, hogy egy memória-rendszer működését megfigyelhessük, és teljesítményét megmérjük. A szimulálás e végső szakasza sokszor a legidőigényesebb feladat, mert egy memóriarendszeren belül is többféle konfigurációt állíthatunk fel (pl. cache mérete, sorok mérete, asszociativitási osztály, helyettesítési politika stb.). A feldolgozás gyorsítására két módszert dolgoztak ki.

Az első megközelítésben a feldolgozási folyamat párhuzamosítását felhasználva, *párhuzamos elosztott szimulálásokat* fejlesztettek ki. Ezek a memóriarendszer különböző konfigurációit egymástól függetlenül és egymással párhuzamosan szimulálják.

A számításigényes feladat felgyorsítására egy másik módszert olyan algoritmusok jelentenek, amelyek képesek a nyom egy lépésének feldolgozásával *egyszerre több memória-konfigurációt szimulálni*.

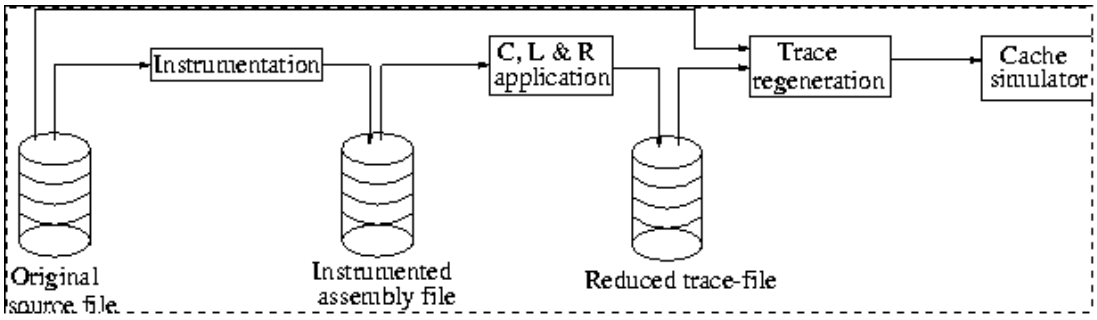
3.4. A szimulálás folyamata

Az eddigiekben megvizsgáltuk a nyomvezérelt szimulálás három összetevőjét külön-külön. Ebben a fejezetben röviden ismertetjük, hogy lehet ezeket az összetevőket összekombinálni ahhoz, hogy egy teljes szimuláló rendszert alkossanak.

Ahhoz, hogy két komponenst összerakjunk, meg kell találnunk azt a módot, ahogy kommunikálni fognak, ahogy az egyik komponens kimenő adatai a másik bemenő adataivá válnak.

A legegyszerűbb módszer erre az, ha felhasználjuk az operációs rendszer kínálta lehetőségeket. A különböző folyamatokat összeköthetjük egy *csővezetékkel (pipe)* vagy rendes *állományon keresztül***.** Az első esetben a termelő folyamat adatai (pl. nyomalkotó) egyenesen átadódnak a fogyasztó folyamatnak (pl. cache-szimulátor). A második esetben a termelő először egy állományba menti el az adatait, majd ebből az állományból a fogyasztó akármikor kiolvashatja az adatokat. A csővezetékes módszer előnye, hogy gyorsabb. Ha viszont állományokat használunk, az adatokat akármikor kiolvashatjuk onnan, és nem kell mindig lefuttatni a termelőt akkor, mikor futtatjuk a fogyasztót. Sokszor megtörténik, hogy a termelő folyamat sokkal gyorsabb, mint a fogyasztó (pl. gyorsabban generálja a címeket, mint ahogy azt a másik folyamat fel tudná dolgozni). Ilyenkor csakis az állományon keresztüli kommunikációt alkalmazhatjuk.

A *4. ábrán* egy vegyes, állományon keresztüli és csővezetékes, kommunikációs modellt láthatunk. Az eredeti forráskódot megjelöljük a jelentős események módszerével, így egy új forráskód-állományt kapunk. Ezt lefordítva és futtatva megalkotjuk a csökkentett nyomot, amit egy újabb állományban őrzünk. A nyom-újraalkotó felhasználva ezt a nyomot és az eredeti forráskódot, generálja a teljes címnyomot, amit már csővezetéken keresztül juttatunk el a szimulátorhoz.



4. ábra. *Teljes szimulálási folyamat*

Más alkalmazások a kommunikációt saját módszerrel oldják meg. Egyesek távolsági eljáráshívást használnak a folyamatok közötti kommunikációra, mások közös memóriát, amelyhez hozzáférése van a nyomalkotónak és a szimulátornak is.

A szimulátor program egyenként beolvassa a nyomból a címeket, és utánozza a memóriarendszer működését. Egyszerű változókkal mérhetjük a különböző események előfordulási gyakoriságát: cache találatok és hibák száma, összes memória-hozzáférés száma, írások száma, olvasások száma, a hozzáférések mintája (lépés, ciklusosság) stb. Végül ezekből az adatokból vonhatjuk le a memóriarendszer teljesítményére vonatkozó következtetéseinket.

Könyvészet

1] IA-32 Intel Architecture Software Developer's Manual – Volume 3: *System Programming Guide*

2] GROSS, T.–STRICKER, T.: *Global Address Space, Non-Uniform Bandwith: A Memory System Performance Characterization of Parallel Systems,* Proceedings of the ACM conference on High Performance Computer Architecture, February 1-5, 1997

3] UHLIG, R. A.: *Trace-driven Memory Simulation: A Survey*

Celluláris neurális hálózatok alapjai

Dr. Gacsádi Sándor, adjunktus
Nagyváradi Egyetem, Elektrotechnika és Informatika Kar,
Elektronika Tanszék

A neurális hálózatok olyan számítási feladatok megoldására létrejött párhuzamos feldolgozást végző adaptív eszközök, melyek eredete a biológiai rendszerektől származtatható.

A hagyományos algoritmikus számítási rendszerekkel szemben a neurális hálózatok számos feladat megoldásánál nemcsak alkalmasabbnak, hanem alapvetően hatékonyabbnak is bizonyultak. Ilyen feladatok tipikusan a különféle felismerési problémák (számok, karakterek, szöveg illetve kép felismerése), optimalizálási feladatok, komplex ipari, gazdasági vagy pénzügyi folyamatok időbeli viselkedésének előrejelzése. A neurális hálózatok hasonlóan fontos alkalmazási területe a nemlineáris rendszerek vizsgálata, identifikációja, szabályozása, amilyen például a robotika, ahol néha az algoritmikus megoldás eddig nem vezetett eredményre.

A neurális hálózatok tudománya egy alternatíva lehet más hagyományos eljárásokkal szemben, bár nem egy abszolút megoldás bármilyen problémára. Természetesen csak ott érdemes alkalmazni a neurális hálózatokat ahol hasznosabbak mint a hagyományos eljárások.

1. Neurális hálózatok

Neurális hálónak nevezzük azt a párhuzamos, elosztott működésre képes információ feldolgozó (számítási feladatok megoldására létrejött) eszközt, amely nagyszámú, hasonló vagy azonos felépítésű, egymással összeköttetésben lévő építőelemből (idegsejt) álló, tanulási képességgel és nagy működési sebességgel rendelkező adaptív rendszer [1,2].

A számítás fogalmát széleskörűen, mint egy bemenet-kimenet (*input-output) operátort* (funkciót) értelmezzük *(f) (black box)*

$$f: u(t) \rightarrow y(t), \tag{1}$$

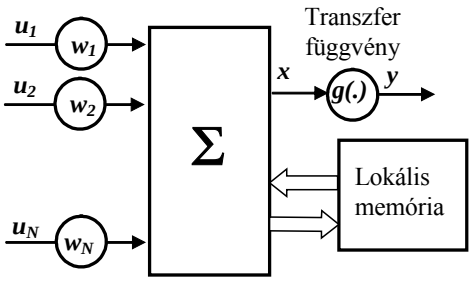
ahol az $u(t)=(u_1(t), u_2(t),\ldots, u_n(t))^T$, (*input*) vektor,

$y(t)=(y_1(t), y_2(t),\ldots, y_n(t))^T$, *(output*) vektor.

A vektorokban összefogott jelek, $u_i(t)$, $y_i(t)$ az idő függvényei.

A *jelek***,** mint információ hordozók, az alábbi három osztályba tartoznak: szimbólumok (karaktersorozatok), diszkrét elektromos jelek és folytonos elektromos jelek.

Az *elemi processzor* (*neuron)* több bemenetű egy kimenetű eszköz, rendelkezik lokális memóriával, melyben akár bemeneti, akár kimeneti értékeket tárol (1. ábra).



1. ábra. *A neuron általános felépítése.*

A bemeneti és tárolt értékekből az aktuális kimeneti értéket tipikusan nemlineáris transzfer függvény alkalmazásával *(g)* hozza létre *(activation function)*. A neuronok rendelkeznek változó értékeket (a hálózat bemeneti jelei, más neuronok kimeneti jelei) hordozó bemenetekkel, illetve rendelkezhetnek állandó értéket hordozó bemenettel. A neurális háló működése lehet diszkrét vagy folytonos idejű.

A *stabilitás* a rendszer azon képessége, hogy egyensúlyi állapotba képes kerülni. Általában a stabil rendszereket azzal jellemezhetjük, hogy stabil állapotukból kimozdítva, tranziens lezajlása után, vagy ugyanoda térnek vissza, vagy pedig egy másik stabil állapotba kerülnek. A neurális háló stabil állapotának elérése úgy is megfogalmazható, hogy legyen a működése konvergens.

Neurális hálózatok egyik legfőbb jellemzője az *adaptációs, tanulási* képesség. A tanulás (súly vektor keresés) arra irányul, hogy egy adott szituációban megfelelő viselkedést mutasson a rendszer. A hálózatok adaptációs képessége arra irányulhat, hogy egy eddig megfelelő viselkedésű rendszer a változó körülményekhez való alkalmazkodás céljából módosítja viselkedését.

Neurális hálózatokban a tanulás alábbi főbb formáival találkozhatunk:

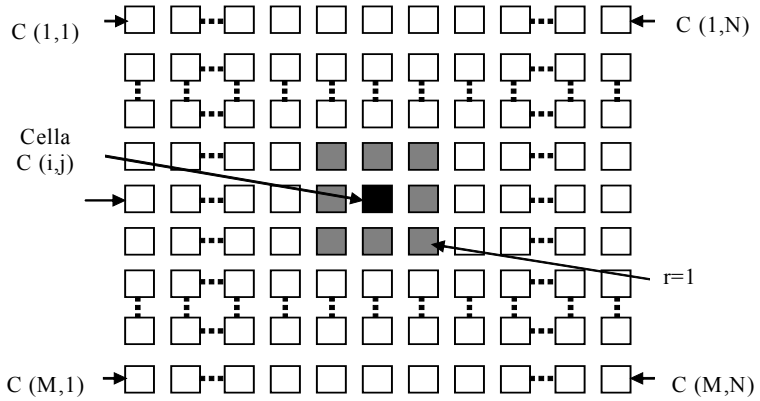
- *Ellenőrzött* tanulásnál (felügyelt tanulás) *(supervised learning)* a hálózat összetartozó bemeneti és kimeneti értékei, a tanító párok rendelkezésünkre állnak.
- *Nemellenőrzött* tanulásnál (önszervező) *(unsupervised learning)* nem állnak rendelkezésünkre adott bemenetekhez tartózó kívánt válaszok.
- *Analitikus* tanulásnál (elméleti) a megfelelő viselkedést biztosító hálózat kialakítása elméleti úton, a feladatból határozható meg.

A neurális hálózatok gyakorlati alkalmazásával kapcsolatban felmerülő kérdések (egy adott feladat megoldására):

- milyen a hálózat felépítése (struktúrája);
- milyen tanulási formát választunk;
- hogyan válasszuk meg a tanulási és a tesztelő készletet;
- milyen megvalósítást választunk (soft vagy hard – ha egyáltalán lehetséges ilyen megoldás).

2. Celluláris neurális hálózatok

A celluláris neurális/nemlineáris hálózat (*CNN-Cellular Neural Network* [3]) két-, esetleg többdimenziós, szabályosan elhelyezkedő, lokális interakcióval rendelkező, nemlineáris dinamikájú *cellából* felépített analóg processzor tömb. Legegyszerűbb esetben a hálózat egy M*N-es négyzetráccsal reprezentálható (2. ábra), amelynek minden cellája a saját közvetlen környezetével van összekötve.



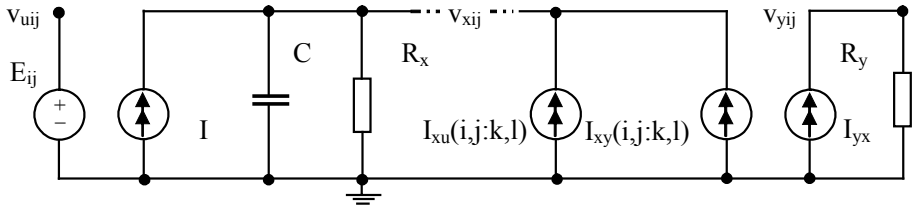
2. ábra. *A CNN architektúrát reprezentáló M*N-es négyzetrács.*

A C(i,j) cella (az i-edik sor j-edik oszlopában lévő elem) *r-sugarú környezetén* az

$$N_r(i,j) = \{C(k,l) \mid \max\{|k-i|, |l-j|\} \le r\},\ 1 \le i \le M;\ 1 \le j \le N \qquad (2)$$

cella-halmazt értjük (2. ábra).

Egy elemi cella felépítése a 3. ábrán látható.



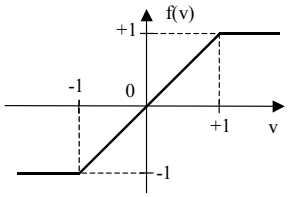
3. ábra. *A CNN cella felépítése.*

Egy cella részei: független áram és feszültség generátorok, vezérelt áram generátorok, egy kondenzátor, két ellenállás. A szomszédokkal való kapcsolat a vezérelt áram generátorokon keresztül történik:

$$I_{xu}(i,j;k,l)=B(i,j;k,l)v_{ukl}\ ;\ I_{xy}(i,j;k,l)=A(i,j;k,l)v_{ykl}. \qquad (3)$$

A hálózat nemlineáris viselkedését a szakaszonként lineáris függvény adja:(4. ábra)

$$I_{yx} = \frac{1}{R_y} f(v_{xij}) \; ; \; f(v) = \frac{1}{2}\left[|v+1| - |v-1|\right] \tag{4}$$



4. ábra. *A CNN cella kimeneti karakterisztikája.*

A cella állapotának szomszédaitól való függését *súlytényezők* határozzák meg, melyek együttesét *template*-nek nevezzük (5. ábra).

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{a}_{i-1,j-1} & \mathbf{a}_{i-1,j} & \mathbf{a}_{i-1,j+1} \\ \mathbf{a}_{i,j-1} & \mathbf{a}_{i,j} & \mathbf{a}_{i,j+1} \\ \mathbf{a}_{i+1,j-1} & \mathbf{a}_{i+1,j} & \mathbf{a}_{i+1,j+1} \end{bmatrix}; \boldsymbol{B} = \begin{bmatrix} \mathbf{b}_{i-1,j-1} & \mathbf{b}_{i-1,j} & \mathbf{b}_{i-1,j+1} \\ \mathbf{b}_{i,j-1} & \mathbf{b}_{i,j} & \mathbf{b}_{i,j+1} \\ \mathbf{b}_{i+1,j-1} & \mathbf{b}_{i+1,j} & \mathbf{b}_{i+1,j+1} \end{bmatrix}; \mathbf{I}=\mathbf{I}_{ij} \tag{5}$$

5. ábra. *3*3-as általános template alakja.*

A template-nek van *visszacsatoló A* (*feedback* – kimeneteket súlyzó), és *előrecsatoló B* (*control* – bemeneteket súlyzó) része. A CNN dinamikáját az A és B template-eken kívül az *I eltolási áram* (*I-bias, treshold*) is szabályozza.

A hálózat dinamikáját az alábbi egyenletrendszer írja le.

– állapotegyenlet:

$$\begin{aligned} C\frac{dv_{xij}(t)}{dt} &= -\frac{1}{R_x}v_{xij}(t) + \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)v_{ykl}(t) \\ &+ \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)v_{ukl}(t) + I_{ij} \end{aligned} \; ; \tag{6}$$

– bemeneti egyenlet:

$$v_{uij} = E_{ij}; \tag{7}$$

– kimeneti egyenlet:

$$v_{yij}(t) = \frac{1}{2}\left[\left|v_{xij}(t)+1\right| - \left|v_{xij}(t)-1\right|\right]; \tag{8}$$

– peremfeltételek:

$$\left|v_{xij}(0)\right| \le 1 \; ; \left|v_{uij}(t)\right| \le 1 \; ; A(i,j;k,l) = A(k,l;i,j) \; ;$$
$$C > 0; \; R_x > 0, \; 1 \le i,k \le M; \; 1 \le j,l \le N \; ; \; C(k,l) \in N_r(i,j). \tag{9}$$

– a hálózat időállandója:

$$\tau_{CNN} = C * R_x. \tag{10}$$

Amennyiben *A* és *B* template-ek értékei pozíció-függetlenek, vagyis nem függenek i és j értékeitől, azt mondjuk, hogy a template *térinvariáns*. Másrészt, a v_u , v_x , v_y , I, jelölés helyet az u (bemeneti jel), x (állapot), y (kimeneti jel), z (eltolási áram) jelölést használjuk. Ha az eltolási áram értéke celláról cellára változik, *eltolási áramtérképről* (*bias map*) beszélünk. Leggyakrabban azonban az eltolási áram értéke is helyfüggetlen:

$$\sum_{C_{kl}\in N_r} A_{ij,kl}y_{kl} = \sum_{|k-i|\le r}\sum_{|l-j|\le r} A_{k-i,l-j}y_{kl};$$
$$\sum_{C_{kl}\in N_r} B_{ij,kl}u_{kl} = \sum_{|k-i|\le r}\sum_{|l-j|\le r} B_{k-i,l-j}u_{kl}, \qquad z_{ij}=z. \tag{11}$$

– állapotegyenlet:

$$x' = \frac{dx_{ij}}{dt} = -x_{ij} + \sum_{C_{kl}\in N_r} A_{ij,kl}\,y_{kl} + \sum_{C_{kl}\in N_r} B_{ij,kl}u_{kl} + \mathbf{z_{ij}}; \tag{12}$$

– kimeneti egyenlet:

$$y_{ij} = f(x_{ij}) = \frac{1}{2}\left[\left|x_{ij}+1\right| - \left|x_{ij}-1\right|\right]; \tag{13}$$

– peremfeltételek:

$$\left|x_{ij}(0)\right| \le 1;\left|u_{ij}(t)\right| \le 1. \tag{14}$$

A *nemlineáris template*-ek esetében a template-értékek nem konstansok, hanem a lokális környezethez tartozó cellák bemeneti, kimeneti illetve állapot értékeinek különböző függvényei. A nemlineáris template-eket a *nemlineáris függvények argumentumai (a,b,c,d)* alapján is osztályozzuk. Így megkülönböztetünk nemlineáris *A, B, C és D* template-eket az alábbiak szerint:

$$\begin{aligned} A_{ij,kl}y_{kl} &= a(y_{ij}, y_{kl}),\\ B_{ij,kl}u_{kl} &= b(u_{ij}, u_{kl}),\\ C_{ij,kl}x_{kl} &= c(x_{ij}, x_{kl}),\\ D_{ij,kl} &= d(u_{ij}, x_{ij}, y_{ij}, u_{kl}, x_{kl}, y_{kl}). \end{aligned} \tag{15}$$

Az ilyen esetekben a hálózat dinamikáját az alábbi differenciálegyenlet-rendszer írja le:

$$\begin{aligned} x' = -x_{ij} &+ \sum_{C_{kl}\in N_r} A_{ij,kl}y_{kl} + \sum_{C_{kl}\in N_r} B_{ij,kl}u_{kl} + z_{ij} +\\ &+ \sum_{C_{kl}\in N_r} C_{ij,kl}x_{kl} + \sum_{C_{kl}\in N_r} D_{ij,kl}(u_{kl}, x_{kl}, y_{kl}). \end{aligned} \tag{16}$$

A hálózatban előreírható, hogy pozíció-függően csak bizonyos cellák állapota változzon illetve másoké változatlan maradjon a tranziens során. Erre szolgál a *rögzített állapot maszk* (*fixed state mask*), melynek bináris értékei engedélyezik, illetve gátolják a cellák állapot-változását.

Kétdimenziós hálózatok egymás fölé helyezésével és összekapcsolásával (lokális összekötetés harmadik dimenzióba kiterjesztésével) kapjuk a *többrétegű* hálózatot.

Az előbbiekben ismertetett folytonos idejű CNN mellett létezik diszkrét idejű változata is, amelyet *DTCNN*-nek (*Discrete Time CNN*) neveznek.

3. Képfeldolgozás celluláris neurális hálózattal

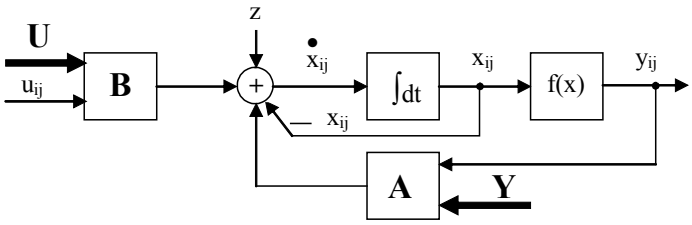
A 3*3 térinvariáns template-eket használó CNN esetében (*standard)*, a következő jelöléseket vezetjük be:

$$\boldsymbol{A}=\begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,+1} \\ a_{0,-1} & a_{0,0} & a_{0,+1} \\ a_{+1,-1} & a_{+1,0} & a_{+1,+1} \end{bmatrix};\boldsymbol{B}=\begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,+1} \\ b_{0,-1} & b_{0,0} & b_{0,+1} \\ b_{+1,-1} & b_{+1,0} & b_{+1,+1} \end{bmatrix}; \tag{17}$$

$$\sum_{C_{kl}\in N_r} A_{ij,kl}y_{kl} \overset{def}{=} \boldsymbol{A}\otimes\boldsymbol{Y_{ij}};\ \sum_{C_{kl}\in N_r} B_{ij,kl}u_{kl} \overset{def}{=} \boldsymbol{B}\otimes\boldsymbol{U_{ij}}; \tag{18}$$

Az állapotegyenlet az alábbi képlettel irható le, amely alapján a cella felépítése a 6. ábrán látható:

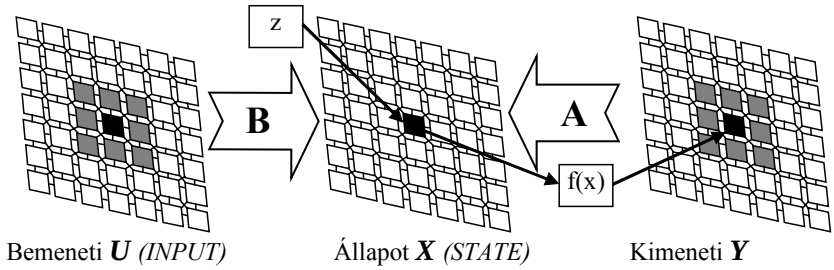
$$x' = -x_{ij} + \boldsymbol{A}\otimes\boldsymbol{Y_{ij}} + \boldsymbol{B}\otimes\boldsymbol{U_{ij}} + \mathrm{z} \tag{19}$$



6. ábra. *A CNN cella felépítése.*

A CNN szabályos, kétdimenziós elrendezéséből kifolyólag ésszerű képpontokkal *(pixel)* ábrázolni a cellák bemeneti, kimeneti és állapot értékeit, (7. ábra). A cella értékeit a szürke különböző árnyalataival reprezentáljuk a fehértől (-1) a feketéig (+1). Ennek következtében az állapot [-1, +1] tartományon kívül eső értékeihez is fehér illetve fekete színt rendelünk. Ennek megfelelőn az $U(t_0)$ a bemeneti kép (*INPUT*), az X azv állapot kép (*STATE)*, az Y a kimenő kép (*OUTPUT*).

Ennek megfelelően a kétdimenziós jelfeldolgozás (képfeldolgozás) egy standard CNN hálóval a 7. ábrán látható. Egy tetszőleges feladat CNN alapú megoldása abban rejlik, hogy a feladatot "meg kell fogalmazni a CNN nyelven", azaz meg kell adni az $U(t_0)$ bemeneti képet, $X(t_0)$ kezdeti állapot képet és a feladat megoldását jelentő template-et. Az eredményt rendszerint az Y kimenő kép adja a tranziens lezajlása után.

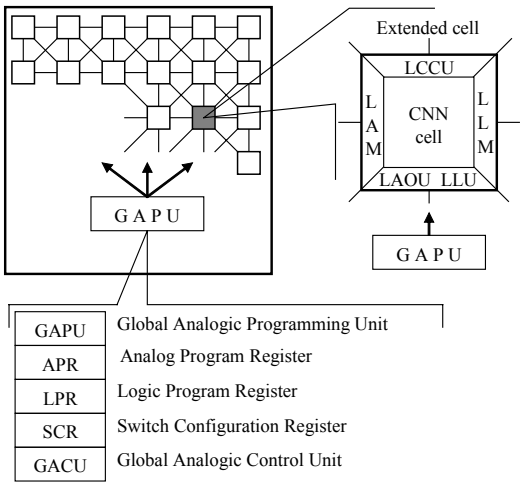


7. ábra. *Képfeldolgozás egy standard CNN hálóval.*

Legegyszerűbb esetben (r=1, 3*3 szomszedság) tehát a CNN dinamikáját 9 visszacsatoló, 9 előrecsatoló és egy eltolási áram-értékkel, azaz 19 számmal adjuk meg. Ezen 19 számból álló template a CNN egy elemi utasítása, amely önmagában egy komplex tér-időbeli dinamikát "kódol", megoldást adva ezáltal egy-egy konkrét feladatra. A template tervezés a CNN dinamikáját leíró egyenletekből származtatható egyenlőtlenségrendszer megoldását jelenti. Bizonyos esetekben ez triviális, más esetekben nehézkes, megoldhatatlan feladatnak tűnik. A CNN irodalomban már rendelkezésre állnak szisztematikus template tervezési módszerek, illetve a már ismert template-ek könyvtára [9,10].

Néhány template-ből és logikai műveletből készül az analogikai (analóg és logikai) algoritmus. A CNN kutatás fontos tárgyköre a CNN analogikai algoritmusok fejlesztése számos tudomány területen, főleg olyanokban ahol a probléma megoldása a kétdimenziós jelfeldolgozásra visszavezethető. Az analogikai algoritmusok kifejlesztése CNN megoldást adhat olyan problémákra is, amelyekre nem lehetséges egy bizonyos megfelelő template-et találni, viszont az algoritmus lépéseire már van illetve könnyű meghatározni template-et.

A celluláris neurális hálózat Turing értelemben univerzális, azaz elvileg tetszőleges feladat megoldható a segítségével [4,5]. A gyakorlatban természetesen csak bizonyos feladatokhoz érdemes alkalmazni, sok problémánál a hagyományos megoldás lényegesen egyszerűbb, gyorsabb.



8. ábra. *A CNN univerzális gép architektúrája.*

4. A CNN univerzális gép (CNN Universal Machine, CNN-UM)

A CNN univerzális gép [6] (*CNN Universal Machine, CNN-UM*) a bemutatott celluláris neurális hálózatra épül. Ez az első algoritmikusan programozható analóg tömbszámítógép saját nyelvvel és operációs rendszerrel [9] (8. ábra).

A CNN-UM univerzális celláit a globális analogikai programozó egység vezérli (*GAPU: Global Analogic Programing Unit*), amely analóg és logikai egységekből áll. A GAPU tartalmazza az analóg utasításokat (template) és logikai függvényeket tároló regisztereket (*APR: Global Analog Program Register, LPR: Global Logic Program Register*), a CNN univerzális cellák működését vezérlő lokális kapcsolok lehetséges konfigurációját tároló regisztert (*SCR: Switch Configuration Register*) és az analogikai utasítások sorozatát tároló globális analogikai vezérlő egységet (*GACU: Global Analogic Control Unit*). A CNN elemi processzorok (cellák) is kiegészülnek lokális analóg és logikai memóriával (*LAM: Local Analogic Memory, LLM: Local Logic Memory*), lokális analóg kimeneti egységgel (LAOU: Local Analog Output Unit), lokális logikai kimeneti egységgel (*LLU: Local Logic Unit*), és a cellát vezérlő lokális vezérlő és kommunikációs egységgel (*LCCU: Local Comunication and Control Unit*).

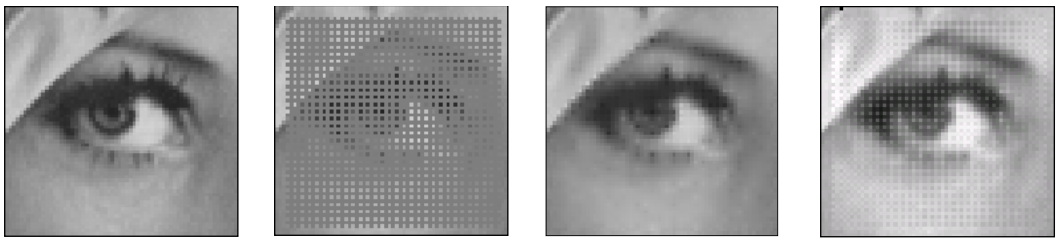
5. Kétdimenziós jel CNN interpolációja

Más típusú neurális hálózatokkal szemben a CNN-UM nagy előnye a chip-ben való integrálási lehetősége és megvalósítása mai, könnyen elérhető technológiával [7,9]. Ebből adódik a párhuzamos feldolgozás ami lehetővé teszi a jelfeldolgozást valós időben (*real time*). Ez különösen fontos a képfeldolgozásban ahol nagyon nagy a számításigény, és ezért sok időbe kerül az információ feldolgozása. A CNN-UM számos képfeldolgozási feladatban alkalmazható.

A következő példában a feldolgozandó képet (9b. ábra) úgy kapjuk, hogy minden második oszlopát és sorát kitöröljük egy eredeti képnek (9a. ábra), vagyis ezekre a helyekre üres pixelek kerülnek (szürkék). A feladat: meghatározni az ismeretlen üres pixelek értékeit azon pixelek értékeiből, amelyeket ismerünk. Az üres pixelek kitöltése kétdimenziós jel CNN interpolációján alapszik [8]. A felhasznált template a aintpol1.tem:

$$A = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = \begin{bmatrix} 0 & (20) \end{bmatrix}$$

A kép interpolációt elvégeztük egy CNN szimulátor segítségével, egy 300 MHz- es PC-én [9] (9c. ábra) és egy 64*64 pixel CNN chip-en [7,9] (9d. ábra). A teljes feldolgozási idő a szimulátor esetében 0,75s, míg a chip esetében csupán 3,4ms, a párhuzamos feldolgozás miatt.



(a) (b) (c) (d)

9. ábra. *CNN kép interpolációja. (a) eredeti kép;(b) bemeneti feldolgozandó kép; (c) PC szimulátor feldolgozás ;(d) CNN chip feldolgozás.*

Könyvészet

1] HERTZ, J.A.–KROGH, A.–PALMER, R.G.: *Introduction to the theory of neural computation,* Addison-Wesley Publishing Company, 1991.

2] HORVÁTH G.: *Neurális hálózatok és műszaki alkalmazásai,* Műegyetemi Kiadó, Budapest, 1998.

3] CHUA, L.O.–YANG, L.: *Cellular neural networks: Theory and Applications,* IEEE Transactions on Circuits and Systems, (CAS), Vol.35, pp. 1257-1290, 1988.

4] CHUA, L.O.–ROSKA, T.: *The CNN paradigm*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, (CAS-I), Vol.40, pp. 147-156, 1993.

5] CHUA, L.O.–ROSKA, T.–VENETIANER, P.L.: *The CNN is universal as the Turing machine,* IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, (CAS-I), Vol.40, pp. 289-291, 1993.

6] ROSKA, T.–CHUA, L.O.: T*he CNN Universal Machine: An analogic array computer,* IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, (CAS-II), Vol.40, pp. 163-173, 1993.

7] LINAN, G.–ESPEJO, S.–DOMINGUEZ-CASTRO, R.–ROCA, E.–RODRIGUEZ-VAZQUEZ, A.: *CNNUC3: A mixed-signal 64x64 CNN Universal Chip,* Proceedings of the International Conference on Microelectronics for Neural, Fuzzy and Bio-inspired Systems, (MicroNeuro99), pp. 61-68, Granada, 1999.

8] GACSÁDI A.–SZOLGAY P.: *Interpolation of 2D signals using CNN,* Proceedings of the the European Conference on Circuit Theory and Design, (ECCTD'01), pp. I 349-352, Espoo, Finland, 2001.

9] ***–*CadetWin-99, CNN application development environment and toolkit under Windows Version 3.0,* Analogical and Neural Computing Laboratory, Computer and Automation Institute, Hungarian Academy of Science, Budapest, 1999.

10] ***–*C S L-CNN Software Library (Templates and Algorithms), Version 7.3,* Analogical and Neural Computing Laboratory, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1999.

A mezőprogramozható kapumátrix áramkörök

Dr. Buzás Gábor, egyetemi tanár
Babeş-Bolyai Tudományegyetem, Fizika Kar,
Villamosságtan, mágnesességtan, elektronika Tanszék

Napjainkban nagyszámú, igen változatos feladatot ellátó programozható logikai áramkör van jelen a piacon. Ezek között a legegyszerűbbek az ún. *csak olvasható* ROM/PRPOM/EPROM/EEPROM típusú tárolók. A szakemberek jelentős része ezeket külön kategóriának tekinti, mivel ezek a bemeneti jelnek „csak" egy jól meghatározott tartalmat feleltetnek meg. Ezzel szemben az „igazi" programozható logikai áramkörök a bemeneti információnak valamilyen funkciót feleltetnek meg.

Mint a logikai áramkörök általában, a programozható logikai áramkörök is két nagy osztályba csoportosíthatók, éspedig a kombinációs és a sorrendi (szekvenciális) áramkörök osztályába. Az előbbiek legegyszerűbb képviselője a programozható logikai tömb (Programmable Logic Array – PLA). A sorrendi programozható logikai áramkörök legjelentősebb tagja a mezőprogramozható kapumátrix.

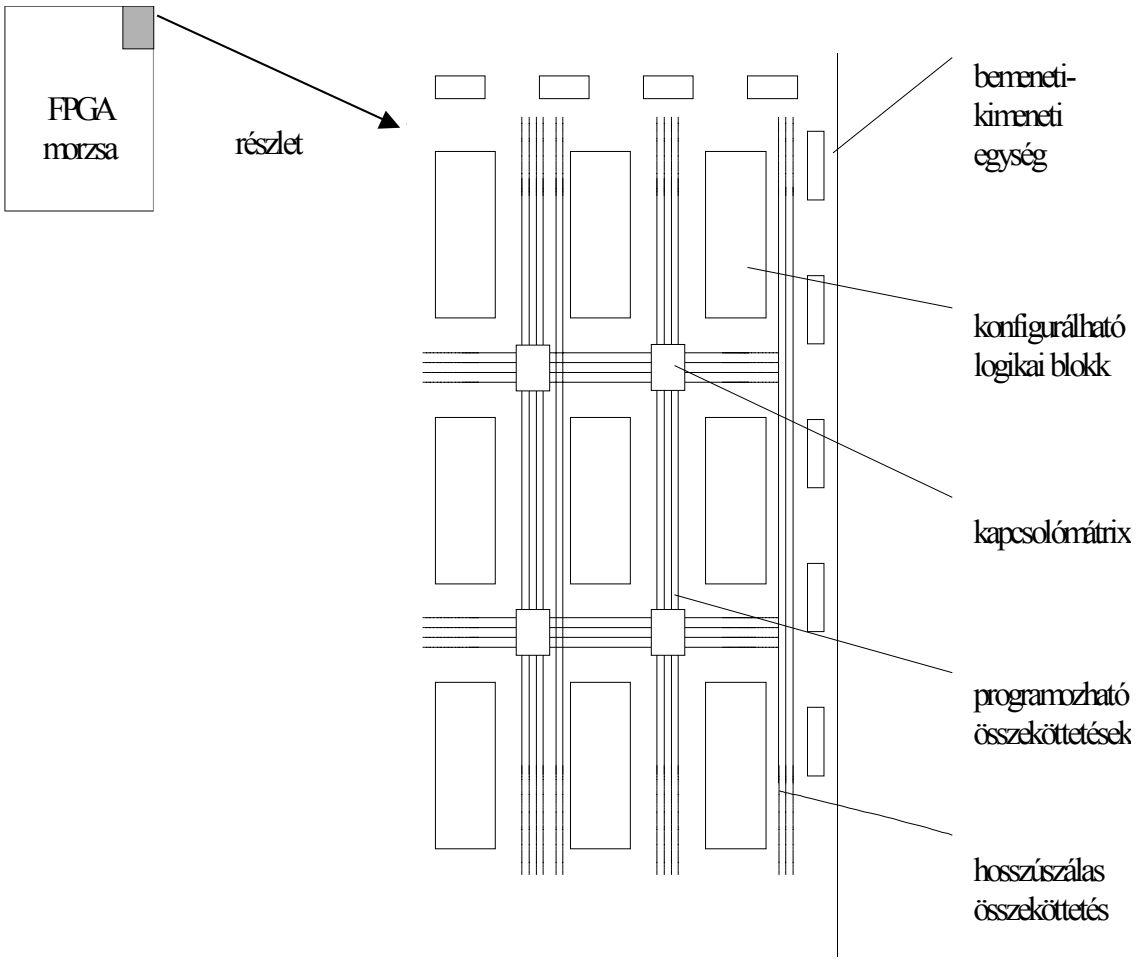
A *mezőprogramozható kapumátrix* (Field Programmable Gate Array – FPGA) az *alkalmazásorientált áramkörök* (Application Specific Integrated Circuit – ASIC) egyik alapvető típusa, amely egyesíti a szabványos integrált áramkörök kedvező tulajdonságait (nagy alkatrészsűrűség, nagyszámú azonos kapu, kedvező ár, egyszerű tervezés stb.) és a kapumátrixok rugalmasságát.

A bonyolultság tekintetében a mezőprogramozható kapumátrix az egyik legbonyolultabb áramkör, amelyet egy tokban kiviteleznek. A mezőprogramozható kapumátrix három fő típusú belső egységből épül fel, éspedig a mátrixszerű *konfigurálható logikai blokkokból* (Configurable Logic Block – CLB), a bemeneti-kimeneti egységekből és a programozható összeköttetésekből. Egy FPGA struktúra részletet szemléltet az 1. ábra.

Az ábra a teljes morzsafelület egy részét mutatja be részletesebben. A konfigurálható logikai blokk tartalmazza mindazon elemeket (logikai kapuk, regiszterek, billenő áramkörök stb.), amelyek valamely logikai függvény megvalósításához szükségesek. Bonyolultságukra és az alkatrészsűrűségre jellemző, hogy a legfejlettebb mezőprogramozható kapumátrix logikai blokkjai összesen kb. 500.000 logikai kaput, kb. 25.000 regisztert stb. tartalmaznak.

A hordozó kerületén a bemeneti-kimeneti egységek foglalnak helyet. Mindegyik, egymástól függetlenül bemenetként, kimenetként vagy kétirányú kapuként definiálható, és akár háromállapotú kimenettel is kompatibilisek lehetnek. Ugyancsak mindegyik *reteszelő áramkört* (billenőkör) is tartalmaz, amelyik felhasználható vagy kiiktatható. Számuk 64-256 közötti és lényegében meghatározzák a kivezetések számát.

A programozható összeköttetések egy kétszintes hálózatot alkotnak a konfigurálható logikai egységek között és ezeket a programozható kapcsolók (kapcsolómátrix) vezérlik. A kapcsolók rendszeint CMOS tranzisztorok, esetleg ellenállás biztosítékok. A konfigurálható logikai egységek, illetve a bemeneti–kimeneti egységek bemeneteit és kimeneteit a programozható kapcsolók kötik a legközelebbi vezetékhez. Ugyanakkor bizonyos számú egységet közvetlenül ún. *hosszúszálas összekötetéssel* is össze lehet kötni. Ezáltal csökkennek a késleltetések.



1.ábra. *FPGA struktúra részlet*

A mezőprogramozható kapumátrix programozását különleges beíróberendezéssel hozzák létre. A már programozott áramköröket ugyancsak különleges tesztelő berende-zésekben lehet kipróbálni. A mezőprogramozható kapumátrixok programozása tapasztalatot igényel. Főleg azért, mert a belső késleltetések nem megfelelő figyelembe vétele kritikus helyzeteket teremthet.

FPGA struktúrák egyszer programozható, vagy törölhető, (elektromos módszerrel) újraprogramozható változatban készülnek.

A legmodernebb FPGA struktúra a *memória-alapú mezőprogramozható kapumátrix*. Ezek a három alapegységen kívül statikus tárolót (10 K – 128 K szó) is tartalmaznak. A

bekapcsolás pillanatában az óhajtott konfigurációnak megfelelő tartalom az alkalmazási felületre telepített csak olvasható tárolóból (EPROM) vagy külső tárolóból az eszköz belső tárolójába töltődik. Ezt követően a belső tároló vezérli az összeköttetéseket létesítő logikát, vagyis a kívánt logikát létrehozó kapcsolókat. Az átprogramozás tehát az EPROM újraírására korlátozódik. Az eljárás kényelmes, gyors és nagyfokú ru-galmasságot biztosít.

Még ennél is modernebb (2001) a *flash tárolót* tartalmazó mezőprogramozható kapumátrix. Itt az átprogramozás a tokon belüli flash memória átírását jelenti. Egy ilyen áramkör lényegében egy nemfelejtő, egytokos *élve ébredő* (live-at-power-up) *rendszert* (System on Chip – SoC) képvisel.

A mezőprogramozható kapumátrixok alkalmazhatósága igen sokrétű. Leggyakrabban olyan különleges mikroprocesszor – vagy mikrovezérlő – feladatokat oldanak meg, amelyek megvalósítása kissorozatú alkalmazásokban nem kifizetődő. Konkrét példaként említhető, hogy nagyon sok űrkutatási és haditechnikai eszköz bonyolult vezérlési feladatait a fent említett áramkörökkel valósították meg.

Viszonylag kevés gyártó szolgálja ki a piacot. Közülük a legjelentősebbek a Xilinx és az Actel Egyesült Államokbeli cégek.

Könyvészet

1] SZITTYA O.: *Analóg és digitális technika,* LSI, Budapest, 1999.

2] BUZÁS G.–SIMON A.: *Az analóg és digitális elektronika alapjai*, Ábel Kiadó, Kolozsvár, 2001.

Többszálú Java programok

Dr. Dollinger Robert, docens
Kolozsvári Műszaki Egyetem, Számítástechnika és Automatizálás Kar,
Számítástechnika Tanszék

A programok jelentős része párhuzamosan végrehajtható részletekre – vezérlési szálakra – bontható.

Így jobban kihasználható a számítógép központi egysége, a programok a külső – például felhasználói – eseményekre gyorsabban reagálhatnak.

Első program:

Elfáradt kiíró program, futás közben lelassul, mint egyedi vezérlési szál működik.

```
public class SleepingThread
{
        public static void main(String[] args)
        {
                for(int i=0; i<100;i++)
                {
                        System.out.print("A");
                        try{
                        Thread.currentThread().sleep(i*10);
                }catch(InterruptedException e){}
        }
}
```

A Thread osztály

A Thread osztály több olyan műveletet, úgynevezett módszert (methods) definiál, amelyek a többszálú programozást teszik lehetővé.

Kétszálú program:

A PrintChar osztály a Thread osztályból öröklődik. Az öröklődés segítségével egy osztály felhasználhatja a hierarchiában felette álló osztályokban definiált állapotot (adatszerkezeteket) és viselkedést (módszereket).

```
//the PrintChar thread class definition
class PrintChar extends Thread
{
        private char charToPrint;   //the character to be printed
        private int times;          //how many times to print
```

```
	//the thread constructor
	public PrintChar(char c, int t)
	{
		charToPrint=c;
		times=t;
	}


	//override the run() method and tell the thread what to do
	public void run()
	{
		for(int i=1;i<times;i++)
			System.out.print(charToPrint);
	}
}
```

A fő program:

```
public class TestThreads
{
	public static void main(String[] args)
	{
		//declare and create the two threads
		PrintChar printA=new PrintChar('A',100);
		PrintChar printB=new PrintChar('B',100);

		//start the threads
		printA.start();
		printB.start();
	}
}
```

Eredmény:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBAAAAAAAAAAAAAAABBBBBBBBB
BBBBAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBB
BBBBBBBBBBBB
```

Szinkronizált vezérlési szálak

A többszálú programozáshoz a Java nyelvi szinten biztosítja az automatikus kölcsönös kizárást. Ennek megvalósításához több módszer áll rendelkezésünkre.

Szinkronizált programrész (blokk)

Megnevezzük az objektumot, amelyen a kölcsönös kizárás történik.

```
//the run() method in PrintChar class with synchronized block
public void run()
{
	synchronized(System.out)
```

```
        {

                //PrintChar threads can enter this block only one at a time
                for(int i=1;i<times;i++){
                System.out.print(charToPrint);
                try{
                        sleep((long)(Math.random()*100));
                }catch(InterruptedException e){}
        }
        System.out.println();
        System.out.println("Document done!!!");
}
//end of synchronized block
```

Az első vezérlési szál zárolja a System.out objektumot.

Szinkronizált módszerek

Statikus (osztály) módszerek

A kölcsönös kizárás az osztályon történik.

```
public class CharPrinter extends Thread
{
        char value;
        int times;
        CharPrinter(char c,int n)
        {
                value=c;
                times=n;
        }

        static synchronized void printerMethod(char value,int times)
        {
                for(int i=0;i<times;i++)
                {
                        System.out.print(value);
                        try{
                                sleep((long)(Math.random()*100));
                        }catch(Exception e){}
                }
                System.out.println();
                System.out.println("Document done!!!");
        }

        public void run()
        {
                printerMethod(value,times);
        }
}
```

A printerMethod módszert egy külön CharPrinterSynchronizer osztályba helyezzük.

```
//the CharPrinterSynchronizer class definition
```

```
class CharPrinterSynchronizer
{
	static synchronized void printerMethod(char value,int times)
	{
		for(int i=0;i<times; i++){
		System.out.print(value);
		try{
			Thread.sleep((long)(Math.random()*100));
			}catch(Exception e){}
		}
		System.out.println();
		System.out.println("Document done!!!");
	}
}

//the modified version of the CharPrinter class
public class CharPrinter extends Thread
{
	char value;
	int times;
	CharPrinter(char c,int n)
	{
		value=c;
		times=n;
	}
	public void run()
	{
		CharPrinterSynchronizer.printerMethod(value,times);
	}
}
```

Objektum módszerek

A kölcsönös kizárás az objektumon történik.

```
//definition of the objects on which the threads synchronize
class CharPrinterSynchronizer
{
	synchronized void printerMethod(char value,int times)
	{
		for(int i=0;i<times; i++)
		{
			System.out.print(value);
			try{
				Thread.sleep((long)(Math.random()*100));
			}catch(Exception e){}
		}
		System.out.println();
		System.out.println("Document done!!!");
	}
}
```

A vezérlési szál definíciója:

```
//definition of the thread objects
public class CharPrinter extends Thread
```

```
{
	char value;
	int times;
	CharPrinterSynchronizer syncobject;
	CharPrinter(char c,int n,CharPrinterSynchronizer pcs)
	{
		value=c;
		times=n;
		syncobject=pcs;
	}

	public void run()
	{
		syncobject.printerMethod(value,times);
	}
}
```

A fő program:

```
//the application main class
public class SynchronizedThreads
{
	public static void main (String[] args)
	{
		CharPrinterSynchronizer lowersync=new CharPrinterSynchronizer();
		CharPrinterSynchronizer uppersync=new CharPrinterSynchronizer();
		CharPrinter a=new CharPrinter('A',100,uppersync);
		CharPrinter b=new CharPrinter('B',100,uppersync);
		CharPrinter c=new CharPrinter('a',100,lowersync);
		CharPrinter d=new CharPrinter('b',100,lowersync);
		a.start();
		b.start();
		c.start();
		d.start();
	}
}
```

Eredmény:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAaaaaaaaaaAAAAAAAAAAAAAaaaaaaaaaaaaAAAA
AAAAAAAAaaaaaaaaaaaaaaaaAAAAAAAAAAAAAA
"Document done!!!"
aaaaaaaaaaaaBBBBBBBBBBBBBaaaaaaaaaaaa
"Document done!!!“
BBBBBBBBBBBbbbbbbbbbbbbbbbbBBBBBBBBBBBBBBbbbbbbbbbbbbbbbBBBBBBBBBBBbbbbbbbbbbb
bbbbBBBBBBBBBBBBBBBbbbbbbbbbbbbBBBBBBBBBBBB
"Document done!!!“
bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
"Document done!!!"
```

Runnable Interfész

Lehetőséget nyújt arra, hogy többszálú Java programokat fejlesszünk olyan osztállyal, amely nem a Thread osztályból öröklődik.

Ez két lépésben történik:

– definiálunk egy Runnable objektumot, amely valamely osztályból öröklődik – ez tartalmaz egy run()módszer definíciót amelynek a kódja tartalmazza azt, amit a vezérlési szál végrehajt;

– definiálunk egy vezérlési szálat az előbbi Runnable objektumnak.

A Runnable objektum definíciója, az objektum mint FileWriter működik:

```
import java.io.*;
public class ThreadFileWriter extends FileWriter implements Runnable
{
	char value;
	int times;
	String filename;

	ThreadFileWriter(char c,int n,String fn) throws IOException
	{
		super(fn);
		value=c;
		times=n;
		filename=fn;
	}
}
```

A Runnable objektum definíciója (a run() módszer):

```
public void run()
{
	try{
		for(int i=0;i<times; i++)
		{
			this.write(value);
			System.out.println("Document "+filename+" "+i+ "characters written!");
			try{
				Thread.sleep((long)(Math.random()*100));
			}catch(Exception e){}
		}
	}catch(IOException ex){System.out.println(ex);}
	finally
	{
		try{
			this.close();
		}catch (IOException ex) {}
	}
	System.out.println("Document "+filename+" done!!!");
}
```

A fő program:

```
import java.io.*;
public class ThreadFileWriterTest
{
	public static void main(String[] args)
	{
		ThreadFileWriter tfw1=null;
		ThreadFileWriter tfw2=null;
		try{
```

```
                    tfw1=new ThreadFileWriter('A',10,"C:/out1.txt");
                    tfw2=new ThreadFileWriter('B',10,"D:/out2.txt");
                }catch(IOException ex){
                System.out.println(ex);
        }

        Thread t1=new Thread(tfw1);
        Thread t2=new Thread(tfw2);

        t1.start();
        t2.start();
}
}
```

A két file konkurens irása:

```
Document C:/out1.txt 0 characters written!
Document D:/out2.txt 0 characters written!
Document C:/out1.txt 1 characters written!
Document D:/out2.txt 1 characters written!
Document C:/out1.txt 2 characters written!
Document D:/out2.txt 2 characters written!
Document D:/out2.txt 3 characters written!
Document C:/out1.txt 3 characters written!
Document C:/out1.txt 4 characters written!
Document D:/out2.txt 4 characters written!
Document C:/out1.txt 5 characters written!
Document C:/out1.txt 6 characters written!
Document D:/out2.txt 5 characters written!
Document C:/out1.txt 7 characters written!
Document C:/out1.txt 8 characters written!
Document D:/out2.txt 6 characters written!
Document D:/out2.txt 7 characters written!
Document C:/out1.txt 9 characters written!
Document C:/out1.txt done!!!
Document D:/out2.txt 8 characters written!
Document D:/out2.txt 9 characters written!
Document D:/out2.txt done!!!
```

Thread Prioritás

```
        static int MIN_PRIORITY;        //has value 1
        static int NORM_PRIORITY;       //has value 5
        static int MAX_PRIORITY;        //has value 10

public class TestThreads
{
        public static void main(String[] args)
        {
                //declare and create the two threads
                PrintChar printA=new PrintChar('A',100);
                PrintChar printB=new PrintChar('B',100);

                //start the threads
                printA.start();
```

```
                //changing priority of the printB thread
                printB.setPriority(printA.getPriority()+1);
                printB.start();
        }
}

//the PrintChar thread class definition
class PrintChar extends Thread
{
        private char charToPrint;   //the character to be printed
        private int times;          //how many times to print

        //the thread constructor
        public PrintChar(char c, int t)
        {
                charToPrint=c;
                times=t;
        }

        //override the run() method and tell the thread what to do
        public void run()
        {
                for(int i=1;i<times;i++){
                        System.out.print(charToPrint);
                        try{
                                sleep(10);
                        }catch(Exception x){}
                }
        }
}
```

Elosztott rendszerek: az Internet alapjai

Dr. Vári Kakas István, docens
Nagyváradi Egyetem, Elektrotechnika és Informatika Kar
Informatika Tanszék

Mivel manapság a legközismertebb elosztott számítógépes alkalmazásokat az Interneten valósítják meg, az alábbiakban egy rövid áttekintést adunk az Internet felépítéséről és működéséről. Néhány, a számítógép-hálózatokban használt alapfogalom meghatározása után, a hálózati kommunikációval foglalkozunk, különös hangsúlyt fektetve az Interneten alkalmazott TCP/IP protokollkészletre. Ez utóbbival kapcsolatban megvizsgáljuk a hálózati, valamint a szállítási réteg által nyújtott szolgáltatásokat. Végül összefoglaljuk az Interneten használt címeket és neveket, és bemutatjuk a tartománynév-rendszer felépítését.

1. Alapfogalmak

Legegyszerűbben az *elosztott rendszert* úgy határozhatjuk meg mint több számítógép, az ezeket összekötő kommunikációs hálózat és a gépeken futó, egymással együttműködő programok összességét. Más megvilágításban az elosztott számítógépes rendszer nem más, mint egy számítógép-hálózat, amelyre egy elosztott alkalmazást (szoftvert) telepítettek. Két közismert példája az ilyen alkalmazásoknak az Interneten az elektronikus posta (e-mail) és a világháló, azaz a WWW (World Wide Web) rövidítéssel jelölt hipermédiás rendszer. Ezek az alkalmazások a tudományos életben ma már nélkülözhetetlenek, és lassan beépülnek az emberiség mindennapjaiba. Ugyanakkor a felhasználók igényei a megbízhatóbb szolgáltatások iránt állandóan növekednek. Erre utalva jegyezte meg humorosan egy tudós, hogy „az elosztott rendszer olyan, ahol ha egy gép amelyről sosem hallottál meghibásodik, ezzel gátol a munkádban”...

Az elosztott rendszereknek vannak olyan sajátos jellemzői, amelyek eltérnek a hagyományos, egyedi gépek tulajdonságaitól. Ezek a sajátosságok a hardver és szoftver erőforrások térbeli elosztottságából adódnak. Az elosztottság lehetővé teszi, illetve megkívánja olyan követelmények megvalósítását, amelyeket a tervezőknek figyelembe kell venniük egy új rendszer kidolgozásánál. A legfontosabb ilyen követelmények a következők:

- *erőforrás-megosztás*, amely lehetővé teszi közös hardver egységek vagy adatok elérését különböző, külön gépeken futó folyamatok számára;
- *konkurrencia,* azaz több folyamat egyidejű működése a rendszerben;
- *átlátszóság,* ami azt jelenti, hogy a felhasználónak nincs tudomása az objektumok (programok és adatok) fizikai leosztásáról;
- *nyitottság,* amely az interfészek leírásának nyilvánosságát jelenti;
- *skálázhatóság,* amely a rendszer kibővítését teszi lehetővé az architektúra elvének megváltoztatása nélkül;

- *hibatűrés,* vagyis egy esetleges hiba fellépése esetén is működőképes marad a rendszer;
- *biztonság,* amely elsősorban a felhasználói adatok védelmét, titkosságát feltételezi az illetéktelen hozzáférésekkel szemben.

Természetesen egy-egy konkrét esetben nem minden fenti követelmény valósul meg, sokszor akad olyan, amelynek a beépítése indokolatlanul megnövelné az anyagi ráfordítást.

Mint említettük, az elosztott rendszert (pontosabban a lazán csatolt elosztott rendszert) számítógép-hálózaton valosítják meg. Alapvetően a számítógép-hálózatokat két nagy csoporba oszthatjuk a kiterjedésük függvényében:

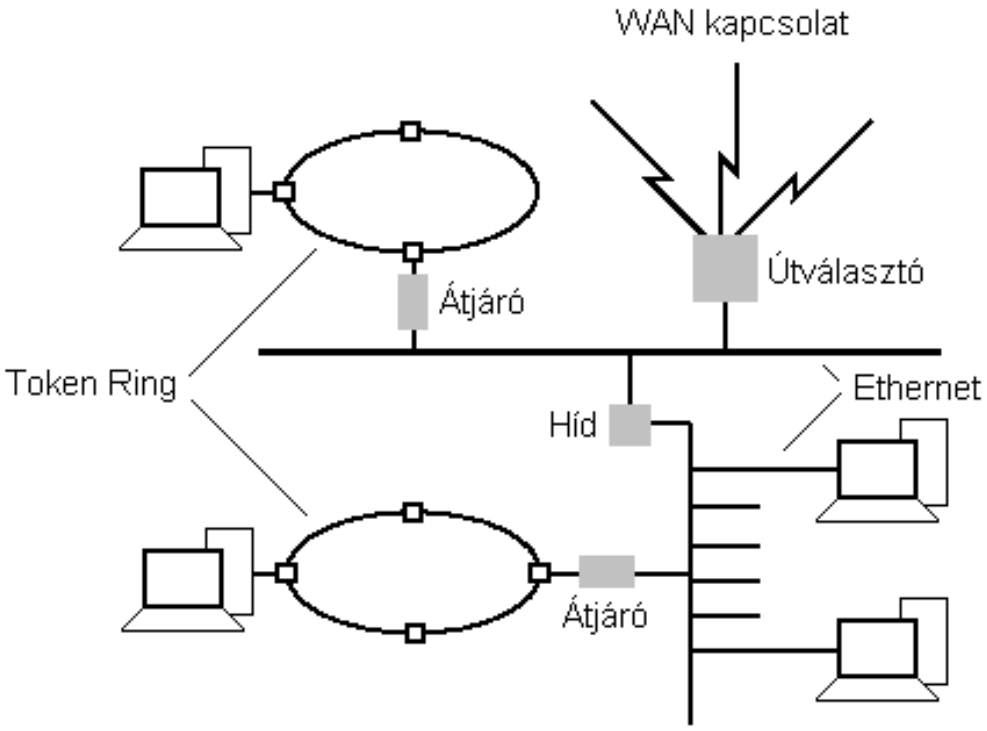
- *helyi hálózatok* (LAN – *Local Area Network*), amelyeknél a gépek egymáshoz közel, ugyanabban a szobában vagy épületben helyezkednek el;
- *nagy területű hálózatok* (WAN – *Wide Area Network*), amelyek nagy, országos vagy kontinensnyi területet fednek le.

A számítógép-hálózatok építőelemei a következők:

- *gazdagépek* (más néven *hosztok*), azaz olyan számítógépek, amelyek kiszolgáló (szerver), illetve kliens folyamatok futtatására alkalmasak;
- *útválasztók,* amelyek ugyancsak számítógépek, de feladatuk az adatcsomagok továbbítása két vagy több hálózat között;
- *átjárók*, amelyek lényegében egyszerűbb útválasztók;
- *hidak*, amelyek két egyforma típusú hálózat között továbbítanak adategységeket (úgynevezett kereteket);
- *jelismétlők*, vagyis erősítő áramkörök egyazon hálózaton belül;
- *átviteli közeg*, amely lehet vezeték, üvegszál, rádióhullám, műholdas átvitel.

Az adatok átvitelének technológiája szempontjából több típusú hálózatot különböztetünk meg. Néhány elterjedt típus az Ethernet (lineáris topológiájú), a Token Ring (körgyűrűs) és az ATM (*Asynchronous Transfer Mode*). Ezek részletes leírása az ajánlott szakirodalomban található meg.

Több számítógép-hálózatnak az összekapcsolásából létrejött hálózatot angolul *internetwork*nek hívjuk. Az *Internet* egy ilyen hálózat, éspedig az, amely szinte az egész világot lefedi és amelyen minden gép egységes, éppen az Internetre kidolgozott címzési rendszert és protokoll készletet használ. (Ha az Internet eszközei egy különálló, házon belüli hálozaton kerülnek megvalósításra, akkor *intranet*ről beszélünk). Az 1. ábra az Internet egy lehetséges részletét mutatja be.



1. ábra. *Internetre kapcsolt helyi hálózatok*

2. Hálózati kommunikáció

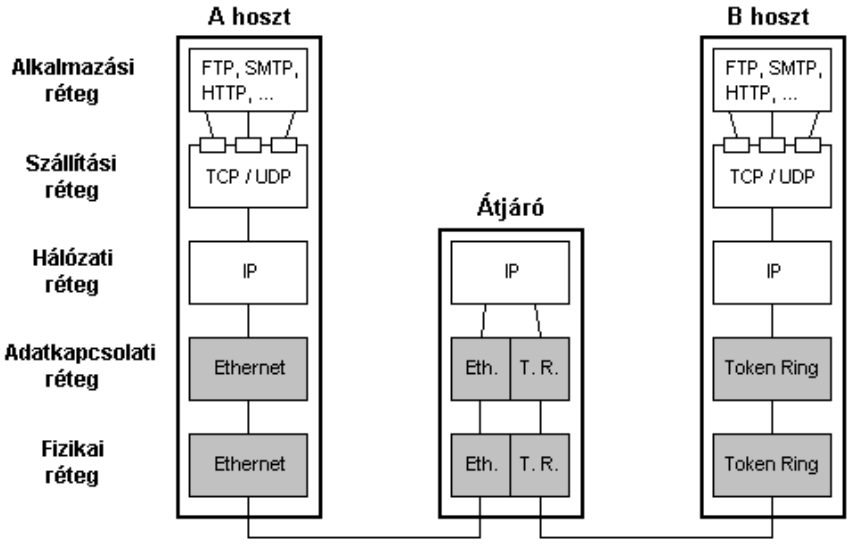
A hálózati kommunikáció lényege, hogy két különböző számítógépen futó felhasználói folyamat között biztosítsa az adatok átvitelét. E probléma megoldására célszerűnek tűnt a résztvevő szoftver és hardver elemeket hierarchikus rétegekben elhelyezni mindegyik gépen. Két ugyanazon a szinten levő réteg között meg kell határozni egy *protokollt*, amely az üzenetcsere szabályait és az adatok formátumát rögzíti. Egy gépen belül, két szomszédos réteg közül az alsó egy szolgáltatást nyújt a fölötte levő rétegnek egy, mindkét réteg számára előre meghatározott interfészen keresztül. Az interfészben leírják a szolgáltatás hívásának, valamint a válasznak a formáját és a tartalmát.

A Nemzetközi Szabványügyi Szervezet (ISO) kidolgozott egy úgynevezett *OSI (Open Systems Interconnection)* hétrétegű referenciamodellt nyitott rendszerek számára. Ennek az ajánlásnak a rétegei sorrendben, alulról felfelé, a kovetkezők:

- *fizikai réteg,* amely az átviteli közeghez való csatlakozást biztosító áramköröket tartalmazza;
- *adatkapcsolati réteg,* amely az adatblokkok (keretek) hibamentes átviteléért felelős;
- *hálózati réteg,* amely két, egymással közvetlenül összekötött gép között továbbítja az adatcsomagokat a megfelelő úton;
- *szállítási réteg,* amely a forrás- és célgép közötti adatok (üzenetek) átvitelét biztosítja;
- *viszonyréteg,* amely a két felhasználói program közötti dialógust kezeli;
- *megjelenítési réteg,* amely a felhasználói adatokat a géptől független megjelenítésre alakítja át;

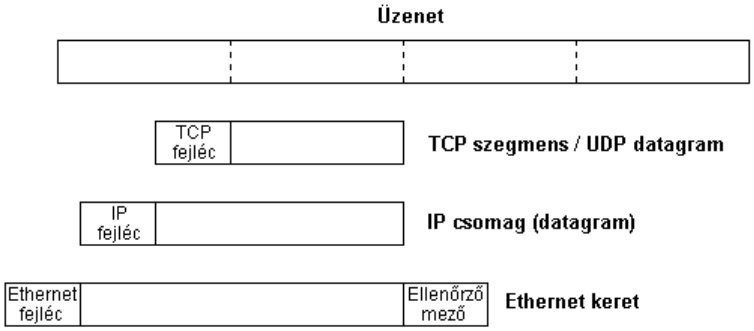
– *alkalmazási réteg,* amely egységes kommunikációs interfészt biztosít a felhasználók számára.

Az Internet nem alkalmazkodott pontosan az OSI modellhez, ugyanis az utolsó három réteget összevonták. Az Internetre kidolgozott protokollkészlet elnevezése TCP/IP (*Transmission Control Protocol / Internet Protocol*). Amint az a 2. ábrán is látható, a két legalsó réteg függ az illető hálózat technológiájától, tehát a TCP/IP készlet a hálózati és a szállítási rétegekre vonatkozik. Az alkalmazási réteg természetesen az illető alkalmazásnak megfelelő (az ábrán feltüntetett alkalmazások: FTP – *File Transfer Protocol*, SMTP – *Simple Mail Transfer Protocol*, HTTP – *HyperText Transfer Protocol*).



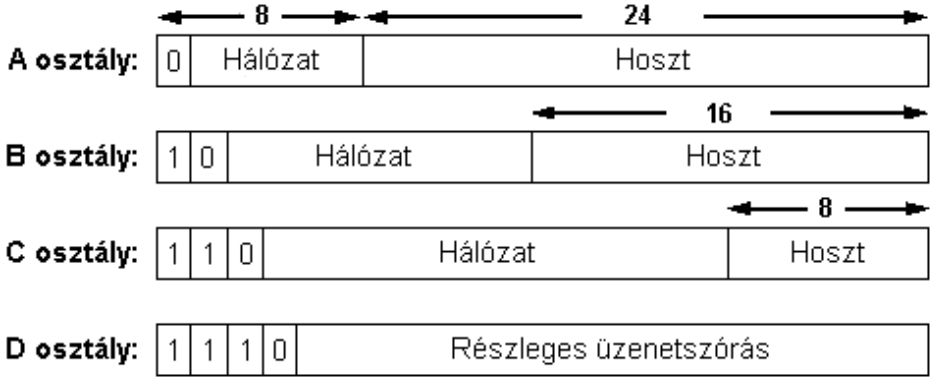
2. ábra. *Az Internet protokollkészlete*

A falhasználói folyamat adatai végighaladnak a küldő gép protokollszintjein egészen a legalsó rétegig, ahonnan az átviteli közegbe kerülnek. Mindegyik réteg a fölötte levő szinttől kapott adatokat becsomagolja, azaz ellátja olyan információkkal egy sajátos fejléc formájában, amelyeket a fogadó gépen azonos szinten levő réteg fel fog használni a protokollnak megfelelően. Meg kell jegyezni, hogy egy bizonyos réteg számára nem szükséges értelmezni a fölötte levő réteg adatmezőit, ezeket egységesen adatokként kezeli. A céloldalon az érkezett adatok kicsomagolása zajlik, miközben ezek az adatok felfelé haladnak a legfelső szintig. A különböző rétegekre jellemző adatformátumokat és az adategységek elnevezéseit a 3. ábra mutatja be. Az adatkapcsolati rétegre példaként feltüntetett Ethernet keret egy ellenőrző mezővel végződik, amelyet a fogadó oldalon újraszámítanak és összehasonlítanak a kapott értékkel, így biztosítva a hibamentes átvitelt. Megtörténhet, hogy valamelyik szinten az adatok hossza meghaladja a protokollban megengedett korlátot, amit MTU-val (*Maximum Transfer Unit*) szokás kifejezni. Ekkor az adatokat kisebb egységekre kell feldarabolni és ezeket átadni az alatta levő rétegnek.



3. ábra. *A különböző rétegekre jellemző adatformátumok*

A hálózati réteg funkcióját az *Internet Protocol (IP)* írja le. Ez egy kapcsolatnélküli kommunikációt valósít meg a forrás- és a célgép között, ami azt jelenti, hogy az elküldendő üzenetből keletkezett maximum 64 Kbájtos csomagok egymástól függetlenül kerülnek átvitelre. Ebből következik, hogy a csomagok külön utakon, illetve a küldéstől eltérő sorrendben érkezhetnek célba. Minden csomag tartalmazza a célállomás címét, amit *IP-címnek* vagy *Internet címnek* neveznek. Egy ilyen cím 32 bitből áll, de a könnyebbség kedvéért minden bájtját inkább a tizes számrendszerben szokták kifejezni. Így például a 11000001 00000000 00000101 10000011 címre 193.0.5.131-ként tudunk hivatkozni. Az IP-címeket több osztályba sorolták (A, B, C, D) a címzett gépet tartalmazó hálózat méretétől függően (4. ábra). A D osztályba tartozó címeket részleges adatszórásra (*multicast*) használják. A csomagok pontról pontra vándorolnak az Interneten és az IP réteg végzi el minden közbeeső útválasztón ezek helyes irányítását a bennük található cím alapján. Mindegyik csomaghoz hozzárendelnek egy élettartamot (TTL – *Time To Live*), amelyből minden csomópontnál levonnak egyet és ha ez elfogy, akkor a csomag megsemmisül. Az Internet hálózati rétege által nyújtott átvitel nem megbízható, mert nem történik semmilyen visszajelzés arról, hogy az adatcsomagok célbaértek-e vagy sem.



4. ábra. *IP-címek osztályozása*

A csomagok helyes irányítása az útválasztókban található *útválasztási tábla* alapján történik, amely

[cél IP-címe, következő csomópont IP-címe]

alakú párosításokat tartalmaz. Ezek a bejegyzések lehetnek statikusak (valaki beírja) vagy dinamikusak (a szomszéd útválasztó táblájából átvett). Tehát az útválasztó meg tudja határozni a következő csomópont IP-címét, de az adatok továbbításához lényegében ennek a pontnak a hardver (fizikai) címére van szükség, amelyet az adatkapcsolati réteg az adatkeret fejlécében helyez el. Ennek a címnek a meghatározását, azaz a címfeloldást, az ARP (*Address Resolution Protocol*) nevű protokoll végzi el. Ez szétszórja a helyi hálózatban a feloldandó IP-címet, amire csak az a gép fog válaszolni, amely ennek a címnek a tulajdonosa. A válasz természetesen tartalmazza a keresett hardver címet. A már feloldott címeket

[IP-cím, hardver cím]

bejegyzések formájában egy átmeneti (úgynevezett ARP *cache*) tárban helyezik el, ahonnan gyorsabban kikereshető a szükséges cím.

Az Internet szállítási rétegének faladatait a TCP vagy az UDP protokollal lehet ellátni. A *TCP (Transmission Control Protocol)* egy kapcsolatorientált kommunikációt hoz létre a két fél között, ami először feltételezi egy „virtuális áramkör” létrehozását (ez egy úgynevezett három utas kézfogással valósul meg). Ez a kapcsolat az összes adat átvitele alatt fennmarad és csak azután kerül bontásra. Így lehetőség nyílik egy akármilyen hosszú bájtos adatfolyam továbbítására, amelyet szegmensekre darabolva ad át a TCP réteg a hálózati rétegnek. A kapcsolat kétirányú és áramlásvezérelt, azaz a küldő fél hozzáigazítja a sebességet a fogadó esetleges lassúbb átvevőkészségéhez. A TCP feladata a feldarabolt és az IP réteg által független csomagokban eljuttatott adatok eredeti sorrendjének a helyreállítása. A TCP protokoll megbízható (hibamentes) átvitelt biztosít. Ennek érdekében minden szegmens célbaérkezését nyugtázás követi, és az esetleges hibák felderítésére ellenőrző összeget alkalmaznak. Hiba esetén a fogadó kéri az adategység újraküldését. Ugyancsak újraküldés történik, ha egy bizonyos idő után nem érkezett meg a nyugtázás, de a protokoll biztosítja a nemkívánt kettőzések kiszűrését is. Fontos észrevenni, hogy ezzel a szállítási protokollal lehetett elérni egy megbízható átvitelt egy nem megbízható hálózati réteg (IP) szolgáltatásainak igénybevételével.

A szállítási réteg egy 16-bites *portszámot* használ a cél meghatározására, mely azonosító megtalálható a TCP szegmens fejlécében is. Mindegyik alkalmazásnak csatlakoznia kell a szállítási réteg egyik portjához, például az FTP a 21-es, a Telnet a 23-as portszámot használja.

Az *UDP (User Datagram Protocol)* a szállítási réteg másik lehetséges formája az Interneten. Ez kapcsolatnélküli kommunikáció által szállít korlátozott hosszúságú datagramokat a két port között. Mivel az átvitel nem megbízható (nincs visszajelzés, nem biztosított az adategységek sorrendje), dacára az egyszerűségének, csak ritkább esetekben használják.

Az Internet protokollkészletének igénybevételekor egy alkalmazás a következő információkat kell közölje a programozói felületen keresztül:

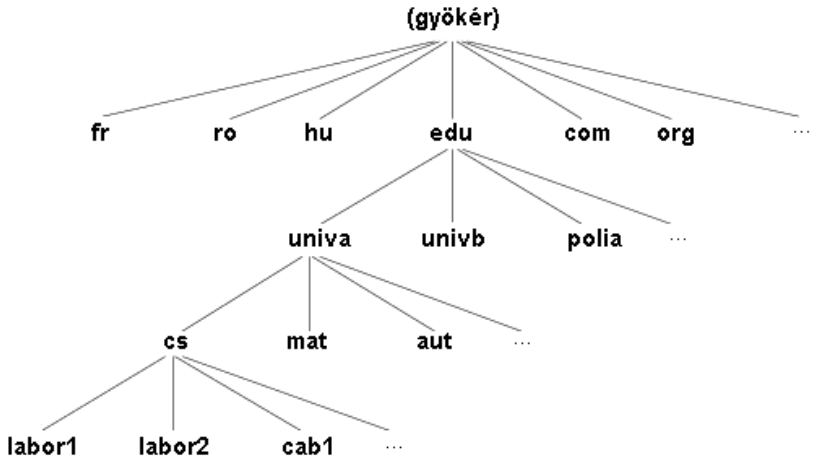
- szállítási protokoll típusa (TCP vagy UDP)
- küldő gép IP-címe
- küldő program csatlakozóhelyének portszáma
- címzett gép IP-címe
- címzett program csatlakozóhelyének portszáma

Az alkalmazások úgynevezett *socket*-ekhez csatlakoznak a kommunikációs csatorna két végén, amelyek egységessé teszik a távoli folyamatok elérését. Egy socket létrehozásánál a < portszám, IP-cím> azonosítót kell megadni.

3. Címek, nevek az Interneten

Az előző fejezetben láthattuk, hogy a protokollhierarchia rétegei sajátos típusú, de számokkal kifejezhető címet használnak. Alkalmazási szinten már kényelmesebb szöveges azonosítókkal (tartománynév, URL, e-mail cím) dolgozni. Összegzésképpen tekintsünk meg egy-egy példát a különböző címekre:

- Hardver cím: 20:3A:50:2:F1:74 (egy hálózati kártya 6 bájtos fizikai címe 16-os számrendszerben kifejezve)
- IP-cím: 194.64.36.1
- Portszám: 80
- Tartománynév: labor1.cs.univa.edu
- Erőforrás-azonosító URL (*Uniform Resource Locator*):
  - http://www.cs.univa.edu/kutatas/elosztott/cikkek
- E-mail cím: dekan@cs.univa.edu



5. ábra. *A tartománynév-rendszer egy részlete*

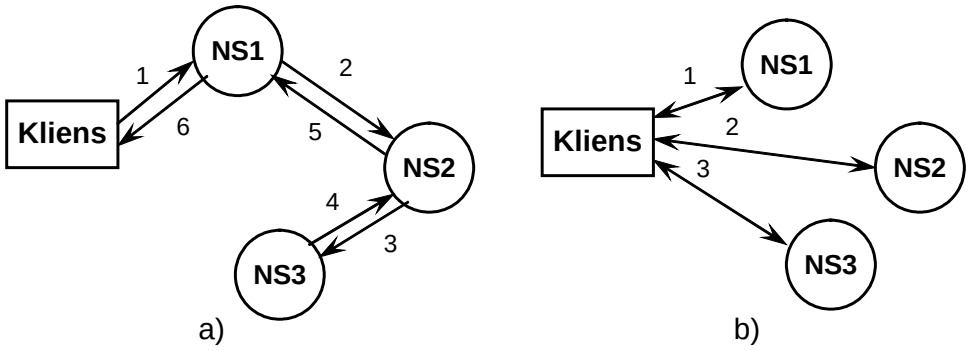
A könnyű értelmezés érdekében az Interneten levő számítógépeket hierarchikusan felépített tartományokba osztották különböző földrajzi és szervezeti kritériumok szerint (5. ábra). Az így létrehozott *tartománynév-rendszerben* (DNS –*Domain Name System*) egy gép azonosítása a hoszt.aldomén.domén általános alakban történik.

A TCP/IP protokollkészlet nem tudja kezelni a tartományneveket, ezért az alkalmazási szinten működő névszolgáltatónak meg kell tudnia a névhez tartozó IP-címet. Ezt a névfeloldást a névszolgáltatók (NS – *Name Server*) végzik, amelyek

[tartománynév, IP-cím]

formájú bejegyzéseket, valamint a szomszédos NS-ek címét tartalmazzák. A névhierarchia egymást nem átfedő zónákra van osztva, és minden zónában legalább egy névszolgáltató található. A névfeloldásra kétféle módszert lehet alkalmazni (6. ábra):

- *rekurzív lekérdezésnél* a kliens a legközelebbi névszolgáltatóhoz fordul, amely ha nem tudja feloldani a nevet, továbbítja a kérést a következő szerverhez, az pedig még tovább, amíg valamelyik szolgáltató fel nem oldja a nevet, és visszaküldi a választ ugyanazon az útvonalon;
- *iteratív lekérdezésnél* a kliens sorban kapcsolatba lép több szolgáltatóval addig, ameddig az egyik probálkozása sikerrel jár.



6. ábra. *Rekurzív (a) és iteratív (b) lekérdezés*

A lekérdezési folyamatban az egyszerűbb UDP szállítási protokollt használják. A gyorsabb névfeloldás érdekében a már feloldott neveket átmenetileg tárolják a hosztban, ezért először itt kell keresni és csak utána fordulni egy névszolgáltatóhoz.

Három fejezetbe foglalva egy rövid áttekintést nyújtottunk az Internet felépítéséről és működéséről, különös hangsúllyal a TCP/IP protokollkészletre. Az alkalmazás szintű szolgáltatások közül a névszolgáltatás került említésre. Egyéb alkalmazásokkal és a velük kapcsolatos problémák megoldásával más alkalommal fogunk foglalkozni.

Könyvészet

1] ANDREW, S.–TANENBAUM,: *Computer Networks,* Prentice-Hall, 1996.

2] KIS Balázs: *WINternet,* Szak Kiadó, 1997.

3] BOIAN, Florian Mircea: *Programarea distribuită în Internet,* Editura Albastră, 1999.

Angol-magyar-román szójegyzék

acknowledge	nyugtázás, visszaigazolás	confirmare
address	cím	adresă
address resolution	címfeloldás	rezoluția adresei
application layer	alkalmazási réteg	nivelul aplicație
binding	kötés	legare
bit	bit	bit
bridge	híd	punte
broadcast	üzenetszórás	difuzare
buffer	puffer	tampon
byte	bájt	octet
cache	gyorsító (átmeneti) tár	memorie (intermediară) rapidă
checksum	ellenőrző összeg	sumă de control
class	osztály	clasă
client	kliens	client
communication medium	kommunikációs közeg	mediu de comunicație
computer network	számítógép-hálózat	rețea de calculatoare
congestion	torlódás	congestie
connectionless	kapcsolatnélküli	fără conexiune
connection-oriented	kapcsolatorientált	orientat pe conexiune
data	adat	date
data link layer	adatkapcsolati réteg	nivelul legăturii de date
data unit	adategység	unitate de date
datagram	datagram	datagramă
datastream	adatfolyam	flux de date
default gateway	alapértelmezés szerinti átjáró	poarta implicită
distributed system	elosztott rendszer	sistem distribuit
domain	tartomány, domén	domeniu
domain name	tartománynév	nume de domeniu
e-mail	elektronikus levelezés	poşta electronică
file	fájl	fişier
flow control	áramlásvezérlés	controlul fluxului
fragmentation	feldarabolás	fragmentare
frame	keret	cadru
gateway	átjáró	poartă
handshake	kézfogás	„înțelegere”
header	fejléc	antet
host	gazdagép, hoszt	gazdă
identifier	azonosító	identificator
interface	interfész	interfață

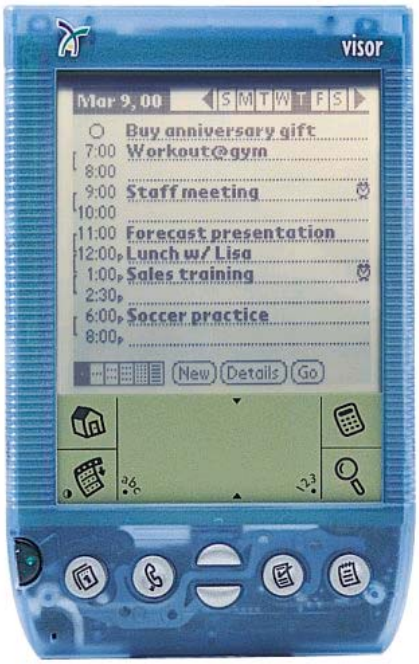
internetwork	hálózatok közötti (hálózat)	inter-rețea
iterative navigation	iteratív navigálás	navigare iterativă
Local Area Network	helyi hálózat	rețea locală
location transparency	hely-átlátszóság	transparența locației
message	üzenet	mesaj
multicast	részleges üzenetszórás	difuzare parțială
name	név	nume
name resolution	névfeloldás	rezoluția numelui
name server	névszolgáltató	server de nume
network layer	hálózati réteg	nivelul rețea
node	csomópont	nod
optical fibre	üvegszál	fibră optică
packet	csomag	pachet
packet loss	csomagvesztés	pierdere de pachet
personal computer	személyi számítógép	calculator personal
physical layer	fizikai réteg	nivelul fizic
port	port	port
presentation layer	megjelenítési réteg	nivelul prezentare
primary server	elsődleges kiszolgáló	server primar
process	folyamat	proces
protocol	protokoll	protocol
record	bejegyzés	înregistrare
recursive query	rekurzív lekérdezés	interogare recursivă
reliable	megbízható	fiabil
repeater	jelismétlő	repetor
replication	sokszorozás (replikálás)	replicare
resource	erőforrás	resursă
router	útválasztó	ruter
routing	útválasztás	dirijare
secondary server	másodlagos kiszolgáló	server secundar
segment	szegmens	segment
sequencing	sorrendbe helyezés	secvențiere
server	szerver, kiszolgáló	server
session layer	viszonyréteg	nivelul sesiune
sliding window	csúszóablak	fereastră glisantă
socket	logikai csatlakozó	soclu
standby	tartalék	de rezervă
switching element	kapcsolóelem	element de comutare
timeout	időzítés	temporizare
token	vezérjel	jeton
transport layer	szállítási réteg	nivelul transport
unreliable	nem megbízható	nefiabil
user	felhasználó	utilizator
virtual circuit	virtuális áramkör	circuit virtual
Wide Area Network	nagy területű hálózat	rețea de arie largă
workstation	munkaállomás	stație de lucru

Zsebtitkárok, belső- és külsőségeik

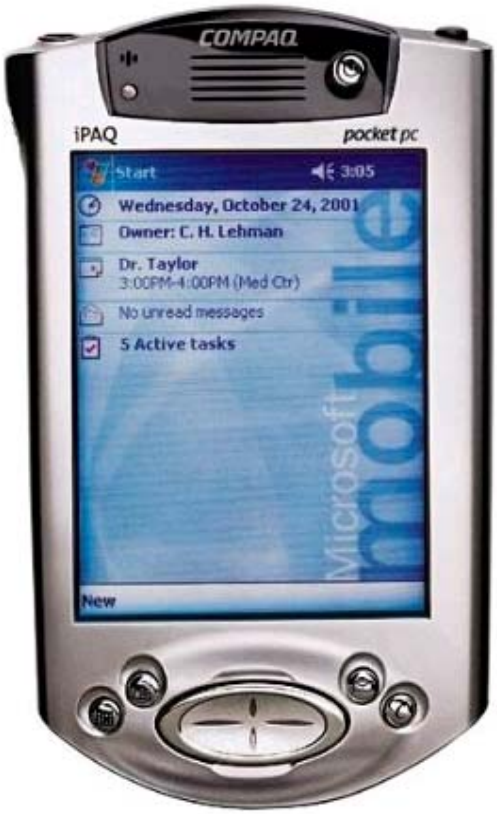
Enyedi Szilárd, okl. mérnök, MsC
Kolozsvári Műszaki Egyetem, Számítástechnika és Automatizálás Kar,
Automatizálás Tanszék

Mi az, hogy zsebtitkár?

Íme két, Nyugaton kedvelt zsebtitkár:



Handspring Visor Deluxe, kék



Compaq iPAQ

A zsebtitkár elektronikus zsebkönyv-helyettesítő. Angolul *PDA*-nak, „Personal Digital Assistant”-nak nevezik.

Az elektronikus változatnak több előnye is van a papírral szemben. A legfontosabbak:

– több adatot képes „észben tartani”;
– az adatok keresése sokkal gyorsabb;
– időzítői is vannak, azaz december 26-án délután három órakor csereg, hogy *„Kolléga, vigyázz, mert holnapután van a nagymamád macskájának a szülinapja!”*; ezt a papírzsebkönyv nem tudja.

Többfajta zsebtitkár létezik, a legegyszerűbbektől a legdrágábbakig. Az olcsóbb fajták tényleg csak a titkári feladatokat látják el, azaz van határidőnaplójuk, telefonkönyvük és más alapprogramok.

Az arisztokrata zsebtitkárok (értsd „drága”) sokkal többet tudnak: zenelejátszás, fényképezés, játékok, stb.

Bölcső, Hotsync?

Nem kell megijedni, mert nem sír.

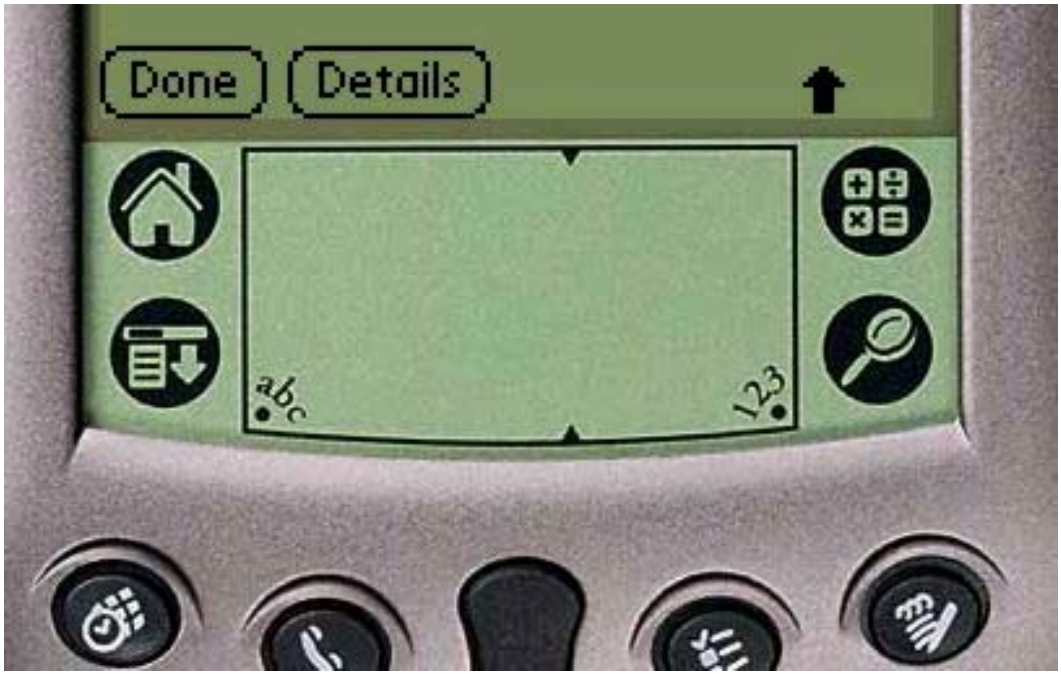
Feltevődik a kérdés: ezek a titkárok nem tudnak adatokat váltani az asztali számítógéppel? De bizony! Hogyan? Egyszerűen beállítjuk őket a „bölcsőbe”, ami tulajdonképpen egy fejlettebb csatlakozó. Ez egy kábelen keresztül kapcsolódik a számítógép soros vagy USB csatlakozójához.

Mi a haszon belőle? Az, hogy megírhatom az összes határidőmet, telefonszámomat, vagy akármit az asztali számítógépen, majd ezeket az adatokat átküldöm a titkárra, egy sajátos szoftver segítségével. Sőt, „Hotsync-et” is végezhetek, ami nem tesz mást, mint hogy a titkár és az asztali számítógépen levő titkár program adatait ide-oda másolja, amíg nem lesz ugyanaz mindkét helyen.

Touchscreen

A legtöbb zsebtitkárnak „touchscreen” kijelzője van. Ez azt jelenti, hogy meg lehet spórolni a billentyűzetet, mert egyenesen a képernyőre lehet írni. Mivel? Műanyagceruzával, ami a titkár tartozéka.

A Palm típusú titkároknak a kijelzője két részre van osztva: az egyik a tulajdonképpeni kijelző rész (a felső), a másik a Graffiti rész (az alsó).

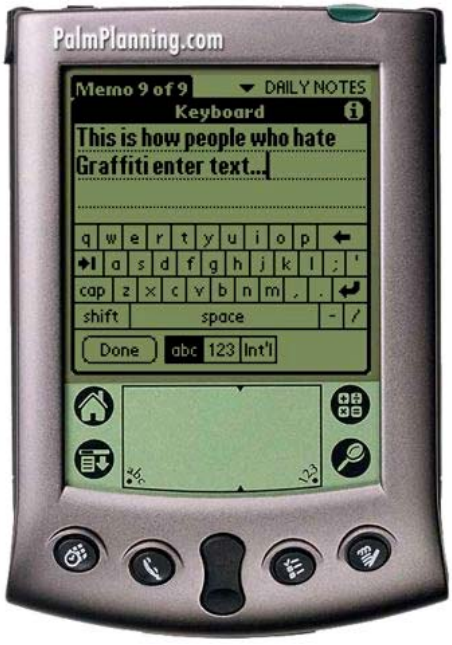


A Graffiti rész.

A Graffiti egy írásfelismerő szoftver. Sajnos nem ismeri fel a kézírást, meg kell tanulni a Graffiti ábécét. Ez tulajdonképpen a betűk és számok olyan alakja, amit le lehet írni anélkül, hogy a ceruzát felvennénk a papírról, azaz a képernyőről. A Graffiti „silkscreen" („selyemernyő") hátránya, hogy ezt a részt nem lehet kijelzésre használni, noha vannak olyan módosított Palm gépek, melyek „virtuális" Graffiti-t használnak, azaz amikor nem írunk, a Graffiti részt is fel lehet használni kijelzésre. Ilyen például a TRGPro.

A Graffiti ábécé egy része. Mindegyik karakter írását a pontnál kell kezdeni.

Egy másik megoldás a „virtual keyboard", mikor is a képernyőn megjelenik egy billentyűzet, aminek a „gombjait" a ceruzával lehet „lenyomni", azaz megérinteni.



Virtual Keyboard.

1. Operációs rendszer

Nos, aki nem tudná, a Linux egy operációs rendszer. A Microsoft cég azt mondja, a Windows is operációs rendszer. Sokan nem értenek egyet. De nem ez a lényeg.

Mivel a zsebtitkárok annyi mindent tudnak, nem lehet csak úgy programozni őket. Nekik is van operációs rendszerük.

A legegyszerűbbeknek egyszerű az operációs rendszerük is. Ez gyakorlatilag azt jelenti, hogy csak azt tudják, amit gyártáskor beléjük programoztak.

A REX 6000-es zsebtitkár. Befér egy PCMCIA bővítő hasadékba, azaz hitelkártya méretű. Nem bővíthető a szoftvere.

Másrészt, fejlettebb társaiknak fejlett operációs rendszerük van. Mit jelent ez? Ez azt jelenti, hogy nemcsak a telefonkönyvet, határidőnaplót és a többi alapfeladatot tudják ellátni, hanem új programokat lehet írni, amit ezeken a titkárokon futtatni lehet. Ezért ezeket az okos kis gépeket *tenyérszámítógépeknek* is nevezik.

- PalmOS
- Windows CE / Pocket PC
- EPOC
- Linux

Ezeken kívül vannak olyan tenyérszámítógépek, melyek IBM PC összeférők, azaz bármely DOS, Windows, Linux vagy más x86-ra jó operációs rendszer feltelepíthető rájuk.

2. A gyártók (operációs rendszer és zsebtitkár)

Palm

Nos, operációs rendszer ügyben és zsebtitkár ügyben pillanatnyilag a Palm cég (www.palm.com) vezeti a piacot. A PalmOS operációs rendszer az ő fejlesztésük. Jelenleg a PalmOS 5.1-es változatán dolgoznak.

Eredetileg a PalmOS-t a Motorola DragonBall EZ, illetve VZ processzorra írták; most a Palm cég át akar térni az Intel StrongARM processzorokra az új zsebtitkár nemzedékkel.

A Palm cég egyik leghasznosabb fogása (saját maguk és a felhasználók számára) az volt, hogy a PalmOS operációs rendszert eladták más cégeknek is, így a cégek csak a hardvert kell fejlesszék, ha egy új Palm-összeférő zsebtitkárt akarnak gyártani.

A PalmOS az egyik leghatékonyabb operációs rendszer: általában elég neki egy 25 MHz-es processzor és 4 MB memória. A különböző programok általában 100 KB alatti méretűek, és az elemek egy hónapig is eltartanak.

Néhány Palm zsebtitkár

Handspring

A Handspring cég (www.handspring.com vagy a magyarországi fiók, www.imc.handspring.hu) Visor nevű zsebtitkáraiban ugyancsak a PalmOS szállítgatja a biteket.

A cég érdekessége, hogy alapítói a Palm cég volt főnökei. Nem tetszett az új vezetés (a Palm céget megvette az US Robotics, amit később a 3COM vett meg; de lehet, hogy fordítva volt), úgyhogy kiléptek.

A Handspring Visorok fő megkülönböztetője, hogy különleges *Springboard* kiterjesztő kártyát lehet beléjük csúsztatni, akár a Nintendo Gameboy játékokba. Ilyen Springboard modul sokfajta van: digitális fényképezőgép, MP3 lejátszó, memóriabővítők, GPS stb.

Néhány Handspring Visor zsebtitkár

Handera

A Handera TRGPro volt az első zsebtitkár, amely CompactFlash memóriabővítő kártyát elfogadott, azaz van a titkáron egy hasadék, amibe egy CF kártyát be lehet csúsztatni, és ezzel titkárunk memóriája 8 MB-ról 512 MB-ra ugorhat.

A Handera TRGPro zsebtitkár.

Sony

A Sony is belépett a PalmOS titkárok körébe. És, amint el is várták tőle, szép (és drága) titkárokkal hozakodott elő.

Képek a Sony Clié PEG-NR70V zsebtitkárról; PalmOS 4.1., beépített billentyűzet digitális kamera és MP3 lejátszó.

Kyocera

A Kyocera cég, és újabban más cégek is, piacra dobott egy PalmOS mobiltelefont. Ezen futtatható bármely PalmOS program, ezenkívül telefonálni is lehet vele.

Kyocera Smartphone. PalmOS-alapú.

Windows CE / Pocket PC

Ezt az operációs rendszert nyilvánvalóan a Microsoft cég fejleszti. Fő jellegzetességei: fejlett multimédia képességek, nagy memória- és processzorsebesség igény, nagy áramfogyasztás.

Az első Windows CE gépek nagy méretűek, nagyon lassúak és elemfalók voltak.

Az új PocketPC operációs rendszerrel a Microsoft sokkal jobban termelt. Igaz, ennek is sok memória (16MB) és gyors processzor (200MHz) kell, ennek megfelelően a PocketPC-t futtató zsebtitkárok ára is magas. Az elemek pedig pár napig, egy hétig tartanak.

Compaq

A Compaq cég a PocketPC-re alapozta az iPAQ zsebtitkárait. Ezek nagy sikernek örvendenek, főleg a Compaq névnek és a minőségnek köszönhetően.

EPOC

Az EPOC operációs rendszer a Symbian cég (www.symbian.com) fejlesztése. A cég a Psion, Nokia, Ericsson és a Motorola cégek társulásából született, hogy az új generációs telefonoknak megfelelő operációs rendszert fejlesszen ki.

Megjegyzés: nem Sybian, hanem Symbian. A Sybian cég más termékeket gyárt.

Linux

Bizony, Linux.

A Linux operációs rendszer nagyon elterjedt. Ingyenes. A forráskód megkapható, átírható. Többfajta gépre telepíthető, többfajta processzorra. A www.linuxdevices.com vagy www.linuxpda.com honlapokon több olyan készüléket lehet találni, amit a Linux operációs rendszer működtet.

Több cég fogott neki a Linux alapú zsebtitkár gyártásnak, sőt a régiekre is lehet Linuxot telepíteni, például a Plam modellekre, vagy az iPAQ modellekre.

Agenda (www.agendacomputing.com) VR3 Linux-alapú zsebtitkár, három színben.

Sharp (www.sharpelectronics.com) SL5500 Zaurus zsebtitkár. Linux/Java alapú, QTopia grafikus szoftverrel, MP3 lejátszóval, billentyűzettel, színes kijelzővel.

DOS, Windows

Itt persze a Windows 3.1, 9x vagy NT változatokról beszélek, ami ugyancsak Microsoft gyártmány. Az egyetlen feltétel ezek telepítéséhez az, hogy a zsebtitkár IBM PC összeférő legyen.

HP 100LX DOS-összeférő tenyérszámítógép.

IBM PC110, 486SX alapú tenyérszámítógép; DOS-, Linux- és Windows összeférő.

Sony Vaio C1 Picturebook tenyérszámítógép. Crusoe x86-összeférő processzor, kiterjeszthető.

3. Összehasonlító táblázat

	Pocket PC	Palm	EPOC
Licensor	Microsoft	Palm	Symbian
Standard applications	PIM applications: Pocket Outlook Plus eBook reader, Internet Explorer, MP3 player, voice recorder, Street Maps, Pocket Word, and Excel	PIM applications: address book, calculator, calendar, expense, task, games, mail storage, security, memo pad, to-do list	PIM applications: address book, calculator, calendar, task, mail, security, memo pad, to-do list. Plus Wireless Application Protocol (WAP)/Web browser, voice recorder, word processor
Screen resolution	320 x 240 640 x 480	160 x 160	320 x 120 640 x 240 320 x 240 480 x 160
Display support	Monochrome, color	Monochrome, color	Monochrome, color
Display levels	4K - 65K color	16-bit grayscale. Some 256- and 65K color displays	16-bit grayscale 24-bit color
Battery life	Rechargeable lithium-ion with approx. 6-12 hours continuous life (or 1 week normal operation)	AAA alkaline with approx. 30 hours continuous life (or 4 weeks normal operation). Rechargeable lithium-ion on some models	Rechargeable lithium-ion with approx. 14 hours continuous life (or 1 week normal operation)
Processor	StrongARM, MIPS, and SH3 (131 MHz–206 MHz)	Motorola Dragonball 16 MHz–33 MHz	ARM 36 MHz
Standard memory	16 MB–32 MB	2 MB–8 MB	8 MB–16 MB
Weight range	6.4 oz–9.1 oz	4 oz–6 oz	8 oz–12.5 oz or 5.6-oz smart phone
Size (H x W x D) in inches	5.25 x 3.25 x 0.5 5.25 x 3.35 x 0.84	4.5 x 3.1 x 0.4 5.25 x 3.25 x 0.75	5.2 x 1.96 x 1.02 6.7 x 3.5 x 0.9
Synchronization	Microsoft Outlook and additional software to synchronize with other e-mail and scheduler software	Palm Desktop and Netscape. Requires conduit software for synchronization with Outlook and other e-mail and schedulers	Outlook, MS Schedule+, Lotus Organizer®, and Lotus cc:Mail®. Plug-in software available for Lotus Notes® Mail, Novell® GroupWise® and Symantec ACT!
Input	Virtual keyboard, character recognition, handwriting recognition	Virtual keyboard, Graffiti character recognition	Integrated keyboard

A fontosabb PDA operációs rendszerek fő jellegzetességei.

4. Szoftver bővíthetőség

A zsebtitkár programok általában két részből állnak:

- Az egyik rész az asztali számítógépen fut, a legtöbb Windows alatt. Ez a rész szűri át az adatokat, amit a zsebtitkárra majd „átküldünk”.
- A másik rész a zsebtitkáron fut, és az adatbázis kicsinyített másával dolgozik.

Szoftver fejlesztés

Mivel a fentebb ismertetett operációs rendszerek bővíthetőek, akárki nekifoghat és új szoftvert írhat hozzá.

Léteznek ingyenes és kereskedelmi fejlesztő eszközök minden gyártó honlapján, csak meg kell keresni, lehozni és telepíteni.

Általában C-ben lehet velük dolgozni, de léteznek fejlesztői környezetek Basic vagy más programozási nyelvekhez is.

Kész programok

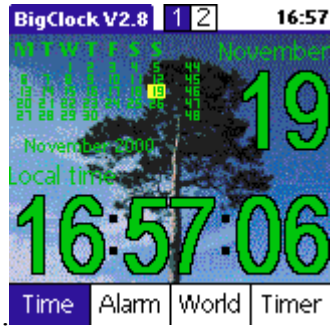
A PalmOS rendszerhez több tízezer kész szoftver létezik minden elképzelhető ágban. Az egyik legjobb PalmOS program forrás www.palmgear.com, vagy www.freewarepalm.com.

A legtöbb szoftver szerveren (www.download.com, www.tucows.com és mások) fenntartanak egy részleget zsebtitkár programokkal.

Egyes programok ingyenesek, másokat meg lehet vásárolni, akárcsak a PC programokat. És éppúgy léteznek feltörő kódok is a különféle PalmOS programokhoz. Léteznek szövegszerkesztők, melyekkel Word aktákat lehet megnézni vagy módosítani, játékok és még sok-sok más program.

Ugyanígy, léteznek különböző szoftverek a többi operációs rendszerhez is, igaz, nem ilyen mennyiségben.

A QuickWord szövegszerkesztő (www.quickoffice.com); az asztali gépre telepített résszel együtt képes Word aktákat feldolgozni



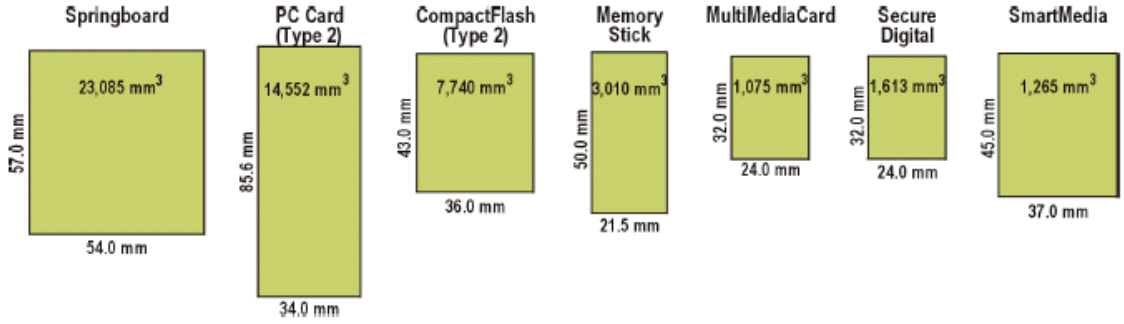
A BigClock program. Ezzel a zsebtitkárunkat asztalióraként is használhatjuk.

Az E-Racer program. Ha lehet, a jó úton kell menni.

5. Hardver bővíthetőség

Nos, ez egy másik érdekes rész. Egyrészt, a titkárokhoz mindenféle bővítőmodult lehet ragasztani; aztán lehet a memóriájukat bővíteni, ami egy jó dolog. És persze adatokat kellene cserélni a többiekkel, legalább a többi titkárral, ha nem a laptoppal.

Bővítőkártyák



A használatban levő PDA bővítőkártyák méretei.

A *Springboard* kártya, mint mondtam, a Handspring cég találmánya, és kizárólag a Visor titkárokhoz használatos. Igaz, hogy nagyobb, mint a többi, viszont az egész bővítőelektronika belefér.

A *PC Card,* vagy PCMCIA kártya, régi találmány, még a zsebtitkárok előtti korból. Főleg laptopoknál használják, manapság minden laptopnak van egy vagy két PCMCIA bővítő hasadéka. Létezik PCMCIA merevlemez, modem, hálózati kártya, stb.

A *CompactFlash* (CF) kártya is régi, bevált bővítőkártya. Főleg digitális fényképezőgépekben használják a fényképek tárolására, de jól bevált a zsebtitkárokban is. Kényelmes, hogy ugyanazt a kártyát lehet használni többfajta készülékben, például fényképezek, azután beteszem a kártyát a titkárba, és kicsit feldolgozom a képet. Aztán, mikor hazaérek, beteszem a kártyát az asztali számítógépbe, és elküldöm a fényképet az Interneten. Létezik modem, hálózati kártya, merevlemez CF kártya is. A CF kártyákat ATA merevlemezként is lehet használni, csak legyen megfelelő csatlakozó.

A *Memory Stick* (MS) a Sony cég bővítőkártyája. Főleg Sony termékekben használt. Itt is létezik modem, merevlemez stb. kártya.

A *MultiMedia* (MMC) kártyák újabbak. Körülbelül azt tudják, mint a CF kártyák.

A *Secure Digital* (SD) kártyák tulajdonképpen MMC kártyák, csak a tartalmukat komoly titkosítási algoritmusokkal védeni lehet.

A *SmartMedia* (SM) kártyák is főleg a digitális fényképezőgépekben használatosak; abban különböznek a CF kártyáktól, hogy elektronikájuk sokkal egyszerűbb: egy hatalmas, integrált memóriamátrixból állnak.

Memóriabővítés

Ez a legegyszerűbb bővítés. Csak egy megfelelő bővítőkártyát kell szerezni.

Persze itt is lehetnek gondok, például a Visor-oknak nincs a Springboardon kívűl semmilyen bővítő hasadékuk. Na de van Springboard a CF, MMC, MS, SD vagy SM kártyákhoz. Sprinboard bele a Visorba, a CF kártya pakk bele a Springboardba. Kész!

Az újabb Palm gépeken van SD bővítő hasadék; a Handera titkárok ismerik az SD és CF kártyákat; a Windows CE gépeknek születésből van CF hasadékuk. A Sony titkárok kedvelik a MS-et.

FlashPlus Springboard típusú CompactFlash bővítőkártya a Visorokhoz.

Adatcsere

Világos, hogy a titkár hogyan beszél az asztali számítógéppel: soros vagy USB csatlakozón, a bölcsőn keresztül. De hogyan beszél a többiekkel?

Az egyik megoldás az *infravörös* adatcsere. Mivel a legtöbb titkár rendelkezik beépített infravörös adó-vevővel, a legegyszerűbb ezzel dolgozni. Sajnos a távolság nem nagy, legfeljebb egy-két méter, a titkár típusától függően. Például így lehet „Hotsync-et" is végezni. Vagy más titkároknak elküldeni a pizza receptet. Vagy ketten játszani. Vagy...

Thinmodem 56K Springboard modem a Visorokhoz.

Egy másik lehetőség a régi szép *modem*. Telefonon. Persze a titkároknak nincs beépített modemjük, úgyhogy kell egy külső modem. Ide jönnek a bővítőkártyák.Létezik *LAN (hálózati) kártya* is.

Egy új ág a „*wireless*" („drótmentes") adatcsere. Két módszer indult el, de nagyon: a *Wi-Fi* (ISO 802.11b) és a *Bluetooth*. Léteznek bővítőkártyák mindkét fajta rádiós hálózathoz.

Más bővítések

Ilyen például a billentyűzet. Szép, érdekes az írogatás a képernyőre, de egy idő után fárasztó. Na és mi van, ha önéletrajzot akarok írni, húszkötetest?

A Targus Stowaway hordozható billentyűzet kinyitva.

A Targus Stowaway hordozható billentyűzet becsukás közben.

Pár kép a Flexis FX100 szilikon alapú, ellenálló billentyűzetről.

Pár kép a Flexis FX100 szilikon alapú, ellenálló billentyűzetről.

A Targus Handcam digitális Springboard fényképezőgép egység. 640x480, színes.

6. Használt magyarosított szavak és kifejezések

Összeférő = kompatibilis
Egység = modul

Ez a bemutató megtalálható HTML alakban a http://users.utcluj.ro/~szilard/ /Teaching/PDA/pda_hu.html oldalon is.

Osztott vezérlőrendszerek

Sebestyén Pál György, adjunktus
Kolozsvári Műszaki Egyetem, Számítástechnika és Automatizálás Kar,
Számítástechnika Tanszék

1. Bevezetés

Egy komplex ipari folyamat vagy berendezés vezérlése több szinten és pontban elhelyezett vezérlőegységet feltételez, amely információt cserél egy kommunikációs környezeten keresztül. A mai számítástechnikai és hálózati eszközök a szükséges technológiai hátteret nyújtják a komplex folyamatok irányításához. A mikroprocesszorok által az automatizálási eszközök (pl. érzékelők, vezérlőegységek, szabályozók, stb.) funkcionalitása megnőtt, pontosabb a működésük és egyszerűbb az adatok továbbítása. A kommunikációs hálózat lehetővé teszi a különböző pontban elhelyezett vezérlőeszközök egyesítését és együttműködését.

Az „intelligencia” elosztása több szempontból előnyt jelent:

- a feladatok megosztása csökkenti a rendszer komplexitását;
- a helyileg elhelyezett „intelligencia” csökkenti a rendszer válaszidejét a különböző külső eseményekre;
- a több pontban végzett feldolgozás biztonságosabb a központosított feldolgozással szemben, lehetőséget nyújtva a hiba-toleráns sémák megvalósításához;
- a vezérelt folyamat területi elosztása a vezérlőelemek elosztását igényli.

Ugyanakkor, egy osztott rendszer tervezése új feladatok megoldását igényli, mint például: a feladatok optimális elosztása, az együttműködő egységek szinkronizálása vagy az adatok és események időben való közvetítése.

Egy (el)osztott számítógéprendszer több, önállóan működő feldolgozóegységet, egy kommunikációs hálózatot és egy bizonyos, egységes szolgáltatást nyújtó szoftvert feltételez. Egy osztott rendszer egy olyan több-processzoros rendszer, amelyben laza a fizikai kapcsolat (a hálózat miatt), viszont szoros a logikai kapcsolat (az egységes program által). Ilyen szempontból az osztott rendszerek különböznek a párhuzamos rendszerektől, ahol a fizikai és logikai kapcsolat egyaránt szoros.

Egy osztott vezérlőrendszer egy olyan rendszer, amelyet egy folyamat vagy berendezés irányítására és felügyelésére fejlesztettek, és amely egy osztott több-processzoros rendszerre alapszik.

2. Az osztott vezérlőrendszerek fő jellemzői

Egy rendszer osztottnak tekinthető, amennyiben elosztottak a hardver és szoftver erőforrásai. Viszont szükséges, hogy az erőforrások elérhetők legyenek a rendszer bármilyen pontjából. A rendszernek biztosítania kell a konkurens hozzáférést a különböző hardver

eszközökhöz, a program részekhez és az elosztott adatokhoz. Az ismertebb hozzáférési modellek közül a kliens-szerver-, a termelő-fogyasztó- és a távoli programhívás-modellek említhetők. Egyszerre történő kérések esetén a rendszer egy megfelelő konfliktust megoldó módszert kell alkalmazzon.

Az osztott rendszer szerkezete nyitott kell legyen ahhoz, hogy különböző típusú eszközöket és berendezéseket könnyen lehessen a rendszerbe bekötni. E cél megvalósításához standard kommunikációs felületeket és protokollokat szükséges használni, amelyek nem függnek a termelőtől vagy a szerkezet típusától. Sajnos gyakran előfordul, hogy a különböző gyártók által termelt készülékek nem csatolhatók ugyanabban a hálózatban, mivel más kommunikációs protokollt alkalmaznak, vagy más hozzáférési módszert használnak. Ma több mint 30 ipari hálózattípust használnak vezérlési célokra.

Szükséges az interoperabilitás (együttműködési lehetőség) a különböző típusú vezérlőeszközök között. Ugyanabban a rendszerben egyszerű automatizálási eszközöknek (pl. hőmérséklet vagy nyomás mérő, vezérlőelem, stb.) és folyamat számítógépeknek, operációs rendszer nélküli készülékeknek és különböző operációs rendszerrel rendelkező gépeknek kell működniük. Az interoperabilitást több szinten lehet elérni: fizikai szinten (azonos feszültségek, áramok, csatlakozók), üzenet szinten (üzenet formátum), logikai szinten (azonos fogalmak), alkalmazási szinten (azonos adatstruktúrák és hozzáférési módszerek).

A skalabilitás egy másik fontos és szükséges jellemzője egy osztott rendszernek, amely a rendszer lépésenként való fejlesztésének lehetőségét követeli. Nem szükséges a már meglevő elemeket módosítani az új elemek rendszerbe való csatolása esetén. Elméletileg szükséges, hogy a rendszernek ne legyenek fizikai vagy logikai korlátai. A gyakorlatban gyakran előfordulnak kapcsolási korlátok, távolsági határok vagy címzési kötések.

Az osztott rendszernek rendelkeznie kell módszerekkel/eszközökkel, amelyek által transzparenssé (láthatatlanná) válnak a fizikai távolságok (pl. az adattovábbítási késések), a konkurens hozzáférési konfliktusok, a kommunikációs hibák, a teljesítménykülönbségek vagy más hátrányos, osztott rendszerekre jellemző tulajdonságok. Az ideális osztott rendszernek úgy kell működnie, mintha az összes erőforrás, program és adat egy pontban csoportosulna.

A megbízhatóság és a hibatolerancia gyakran jellemzi az osztott rendszereket. Mivel a rendszerben több feldolgozó egység létezik, lehetőség van egy olyan stratégia kidolgozására, amely megengedi a meghibásodott elemek felderítését, és káros hatásuk kiküszöbölését. Viszont tökéletes megoldás nincs, a rendszer csak az előre meghatározott hibákra nyújt megfelelő megoldást. A rendszer nincs védve olyan esetekben, amikor a meghibásodott elemek száma meghaladja a tervezett számot vagy egy új fajta hiba jelenik meg.

Az osztott vezérlőrendszerekben fontos a valós-idejű viselkedés. A rendszernek tartalmaznia kell olyan elemeket, amelyek által meg lehet határozni a maximális időt, amely alatt a rendszer válaszol a különböző külső eseményekre. Az osztott rendszerekben nehéz megtalálni egy optimális tervezési algoritmust, amely garantálja az összes taszk határidejének betartását.

3. Kommunikációs modellek

Bármilyen osztott rendszerben az alkalmazott kommunikációs modellek és eszközök fontos szerepet játszanak a rendszer architektúrájában és teljesítményében. Ezért fontos megtalálni a legalkalmasabb modelltm amely optimális egy adott alkalmazásban.

A nyitott hálózati kommunikáció szabványosításához az *ISO-OSI* (International Standard Organization – Open System Interconnection) modellt alkalmazzák, mint referencia-modell. A modell jellemzői közül a következők említhetők:

- a kommunikációs funkciókat 7 hierarchikus szintre osztják;
- egy bizonyos szint több szolgáltatást nyújt a felette való szintnek, és használja az alatta levő szint szolgáltatásait;
- biztonságos adatátvitelt biztosít egy kevésbé biztonságos környezeten keresztül;
- lehetőséget nyújt logikai csatornák létrehozására;
- a modell komplexitása miatt, ritkán használják vezérlési alkalmazásokban.

A legismertebb protokoll csomag, amely az Internet világháló alapját képezi, a TCP/IP (Transmission Control Protocol/ Internet Protocol) néven ismert protokoll szett. A TCP/IP a UNIX gépeken jelent meg először, mint az operációs rendszer kommunikációs funkcióinak a szabványa. Utólag, a TCP/IP kommunikációs modellt a különböző típusú számítógép-hálózatokban alkalmazták. Az osztott rendszerek szempontjából a TCP/IP protokollcsomag a következő hasznos eszközöket kínálja:

- csatlakozásos (TCP) és csatlakozás nélküli (UDP) kommunikáció;
- a folyamatok közti kommunikáció lehetősége;
- hasznos kommunikációs fogalmak: port, socket;
- bővíthető megnevezési rendszer (DNS, NIS, IP-címek);
- távoli eljáráshívás (RPC – Remote Procedure Call).

Ugyanakkor a TCP/IP modell nem biztosítja a csoportkommunikációt, a hiba-toleranciát, és kevésbé transzparens a hálózati adatátvitelre nézve.

Az *ipari digitális hálózatok* csoportjához tartoznak azok a hálózati protokollok amelyek a vezérlőrendszerek igényeire (valós-idejű viselkedés, biztonság, hiba-tolerancia és zaj-immunitás) épültek. Sajnos jelenleg több protokollstandard létezik, és emiatt nehéz egy rendszerben működtetni különböző gyártótól származó eszközöket. Ugyanakkor nehéz bekapcsolni egy ipari hálózatot egy általános használatú, internet típusú hálózatba. Jelenleg ezen a területen a kutatási és fejlesztési tevékenység főleg az interoperabilitási és együttműködési gondokkal foglalkozik. Az ipari hálózatok másik hátránya a magas szintű protokollok hiánya. Nincsenek magas szintű kommunikációs fogalmak és eszközök, és emiatt a programozás gyakran bonyolulttá válik.

A *kliens-szerver* egyike a legismertebb és legtöbbet használt kommunikációs modelleknek, amelyeket az osztott rendszerekben használnak. A modell egy bizonyos szolgáltatást biztosító szerver-programot és több, a szerver által nyújtott szolgáltatásokat felhasználó kliens-programot feltételez. Szolgáltatásnak tekinthető egy közös adatbázis (állomány

rendszer) kezelése, pontos időjelzés, az ipari folyamat bizonyos paraméterei értékeinek a begyűjtése, események rögzítése és továbbítása, stb. A kliens-szerver modell különböző típusú erőforrásokhoz nyújt egységes hozzáférési lehetőséget; a kliens-program nem kell ismerje az erőforrás belső struktúráját vagy működését; a konkurens kérések esetén a szerver-program felelőssége az esetleges konfliktusok megoldása.

A *csoport-kommunikációs modell* lehetővé teszi a biztonságos és konzisztens adatcserét meghatározott csoporttagok között. A modellben alkalmazott módszerek biztosítják azt, hogy egy üzenet eljusson a csoport minden tagjához, vagy ha nem, akkor az üzenet semmisnek minősített. A modell hasznosnak bizonyul ott, ahol szükséges az együttműködés, mint például, amikor több csomópont közreműködik egy bizonyos szolgáltatás biztosításában; a modell könnyíti a hiba-toleráns sémák megvalósítását és a közös információk egységes továbbítását.

Az *objektum-orientált modellek*, mint például a COM, DCOM vagy CORBA egy egységes hozzáférési mechanizmust nyújtnak, amely az objektum fogalmat használja mint címezhető egység. Egy objektum tartalmazhat adatokat és eljárásokat. A szolgáltatást igénylő program (kliens) nem kell ismerje az objektum helyét, struktúráját vagy állapotát. A rendszer megtalálja az objektum helyét egy szimbolikus név alapján, és dinamikusan megfelelő hozzáférési eljárást (stub-procedure) generál. A rendszer nyitott olyan értelemben, hogy különböző platformok köthetők össze, és különböző nyelvben irt programok kommunikálhatnak semmilyen előzetes módosítás nélkül. Sajnos az aránylag bonyolult kommunikációs mechanizmusok sok időt vesznek igényben, és ezért a modellt kevésbé ajánlják idő-kritikus rendszereknek.

Az *MMS (Manufacturing Message Specification) modellt* kimondottan az ipari környezetnek fejlesztették ki. Az MMS egy kommunikációs protokoll, de ugyanakkor javasol egy adat-ábrázolási modellt, amely megfelel az ipari folyamatokban használt információk struktúrájának. A központban levő fogalom a virtuális feldolgozó készülék (VMD – Virtual Management Device), amely tartalmazza mindazokat az adatokat és eljárásokat amelyek egy bizonyos fizikai vagy logikai automatizálási készülékhez tartoznak. Az MMS modell meghatároz olyan objektum-típusokat, amelyek előfordulnak egy folyamatvezérlési rendszerben, mint például: ki- és bemeneti objektumok, esemény-objektumok, adat-rögzítő objektumok, eljárás-objektumok, stb. A modell minden objektumtípusnak meghatároz egy sajátos üzenet-szettet.

4. Tervezési feladatok/gondok

Az osztott-rendszerek tervezése több sajátos és aránylag komplex feladatot jelent a rendszerfejlesztő számára. Ezeket a feladatokat főleg a kommunikáció meg a fizikai távolságok okozzák. Számba kell venni, azt hogy az adatcsere véges időben történik, a kommunikáció nem tökéletes, és gyakran hibák jelennek meg. Nincs lehetőség egy bizonyos pillanatban pontosan megismerni az osztott-rendszer állapotát (relativitáseffektus), és lehetséges, hogy a rendszer különböző pontjában más állapot érzékelhető.

Az egyik fontos feladat egy konzisztens időreferencia felépítése. A vezérlőrendszerekben az idő egy lényeges paraméter: használják a vezérlési függvények meghatározásában, alkalmazzák a periodikus eljárások indításához, vagy hibás működést lehet érzékelni nagyobb késések esetén. Egy osztott-rendszerben szinkronizálni szükséges a párhuzamosan dolgozó egységeket és eljárásokat. Az események sorrendjét az idő alapján lehet meghatározni, amennyiben minden elosztott pontban ugyanazt az időreferenciát használják. De egy egységes időreferenciát szinte lehetetlen megvalósítani, mivel minden helyi óragenerátornak egy saját frekvenciája van, és a szinkronizálást csak egy bizonytalan kommunikációs hálózaton lehet elérni. Ezért egy globális időreferenciát csak egy véges pontossággal lehet elérni. Van amikor nem annyira a pontos idő számít, mint inkább az események közti helyes sorrend meghatározása. Nincs megengedve, hogy a rendszer bizonyos pontjából egy effektus-esemény az okozója előtt legyen érzékelve. Ilyen esetek megoldására az úgynevezett logikai órát alkalmazzák. A logikai óra előállításához egy olyan algoritmust kell alkalmazni (lásd a Lamport féle algoritmust), amely garantálja a kauzális sorrend betartását.

Egy másik fontos gond a konkurens hozzáférés. Feltételezzük a következő helyzetet: egy banki folyószámlán 1000 lej létezik, és egy osztott rendszer két különböző pontjából pénzt akarnak kivonni a számláról; feltételezzük hogy az első kliens leolvassa az értéket és azután a második kliens is leolvassa ugyanazt az értéket; az első kliens levon 100 lejt és beírja a számlára a megmaradt összeget (900 lej); ezután a második kliens levon 200 lejt és beírja az összeget a számlára (800 lej). Könnyen észrevehető az elkövetett hiba. Egyszerű megoldás volna letiltani a számlát más kliensek részére addig, amíg az első kliens el nem végezte a művelet-sorozatot, vagyis egy teljes tranzakciót. Sajnos e megoldás nagyon lelassítja a rendszert. Ezért olyan tranzakciókon alapuló rendszereket fejlesztettek, amelyek hatékonyabban használják ki a közös adatokhoz való hozzáférési időt. Egy másik gondot okozhat, ha egy folyamatban levő tranzakció valamilyen okból megszakad. A rendszer kell biztosítsa a tranzakciók atomi voltát, ami azt jelenti hogy a tranzakció egészében végre van hajtva, vagy egy megkezdett, de be nem fejezett tranzakció esetén vissza kell állítani a tranzakció megkezdése előtti állapotot.

Az osztott vezérlőrendszerek biztonságos működése egy másik feladat a rendszerfejlesztő számára. Az igaz, hogy az osztott rendszerek lehetőséget adnak a biztonság növelésére, de ez nem történik magától. A tervező sajátos módszereket kell alkalmazzon, amelyek által a biztonságos működés biztosítva van. Az első lépes meghatározni a különböző fajta hibaforrásokat és hibatípusokat. Ezek után meg kell határozni a rendszer viselkedését egy hiba megjelenése esetén. A hibákat, hatásuk alapján, a következőképpen lehet osztályozni/modellezni:

- fail-stop – hibásodás esetén a hibás elem leáll – az e fajta hiba aránylag könnyen érzékelhető és kezelhető;
- fail-silence – a meghibásodott elem nem közvetít adatokat – a megoldás hasonló az előzőhöz;
- Bizantin-fail – a meghibásodott elem jó meg rossz adatokat közvetít – nagyon nehezen lehet kiszűrni a hibát.

A hibák hatásának kiszűrését másolatok létrehozásával lehet elérni. A másolat lehet fizikai, amikor több elem végzi el ugyanazt a funkciót, vagy lehet időbeli, amikor ugyanazt a

műveletet ismételten végezik el. A hiba érzékeléséhez és kijavításához egy szavazáson alapuló módszert lehet alkalmazni. Fontos megemlíteni, hogy a rendszer csak azon meghibásodások ellen van védve, amelyeket a tervező előre kigondolt, és amelyek ellen létezik egy hatékony hibatoleráns módszer.

5. Következtetések

Az osztott rendszerek több előnyt nyújtanak a vezérlőrendszerek megvalósítása esetén. De ugyanakkor új feladatok merülnek fel a tervező számára. Egy biztonságos és hatékony tervezéshez szükséges az osztott rendszerekre jellemző eszközök és módszerek használata. Fontos megfelelően megoldani a következő feladatokat: a konkurens adatokhoz és szolgáltatásokhoz való hozzáférést, a megnevezési rendszer kiépítését, a tranzakciók konfliktus nélküli lebonyolítását, a valós-idejű kérelmek megoldását és a hibák kiküszöbölését.

Robottechnikai alapfogalmak

Barabás Tibor, adjunktus
Nagyváradi Egyetem, Elektrotechnika és Informatika Kar,
Automatizálás Tanszék

1. Mi az ipari robot?

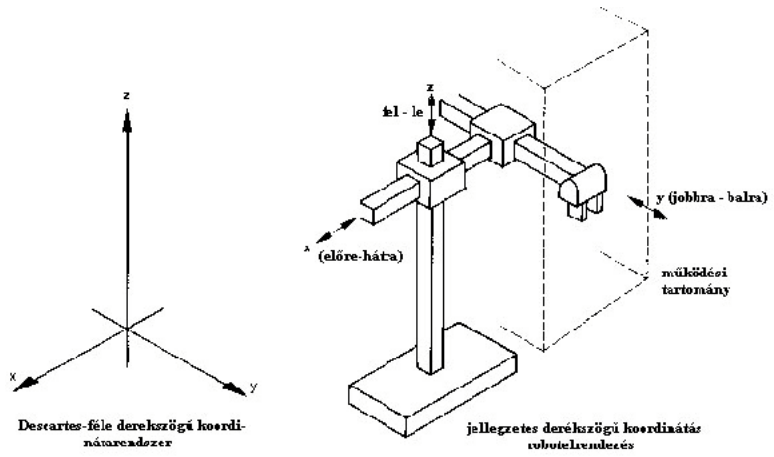
A Robotic Industrial Association (RIA), a korábbi Robot Institute of America, a Society of Manufacturing Engineers társintézményének definíciója szerint egy **„***ipari robot újraprogramozható, többcélú manipulátor, amelynek feladata anyagok, alkatrészek, szerszámok vagy egyéb, specializált eszközök mozgatása változtathatóan programozott mozdulatokkal, különféle feladatok elvégzése érdekében”*.

A kulcsszó ebben a megfogalmazásban az *újraprogramozható,* mert ez a beépített, számítógépes irányító rendszerre vonatkozik. Ez különbözteti meg a robotokat a számvezérlésű szerszámgépektől, amelyeket nem lehet új feladatokhoz átalakítani.

A számítógép tárolja az utasításokat, „megmondja” a robotnak, hogy milyen mozgásokat kell elvégeznie az adott feladat befejezéséhez. Mivel azonban újra-programozható, a feladat megváltozásakor a robot „tanulás” révén, új mozgásokat elsajátítva alkalmazkodhat az új feladatokhoz.

2. Melyek az ipari robotok jellegzetes típusai?

Az ipari robotok, „anatómiájukat” tekintve, lehetnek *derékszögű, hengeres, gömb* jellegűek vagy *csuklós karúak*.

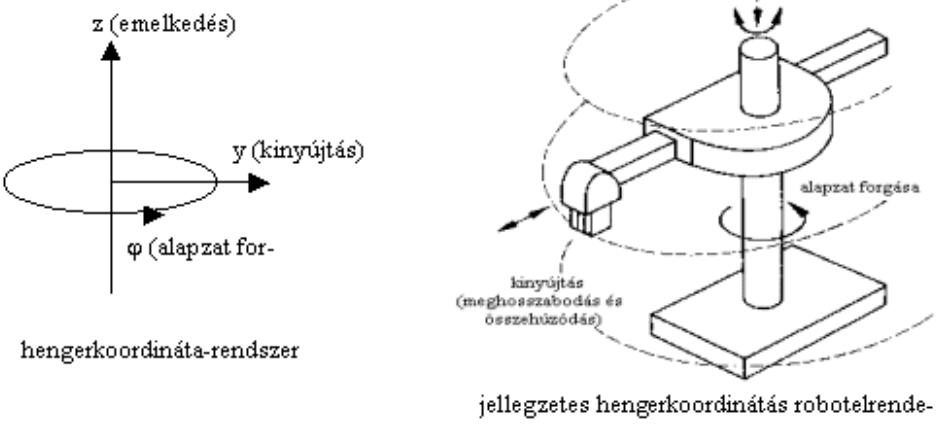


1. ábra

A *derékszögű elrendezéseket* (1. ábra), néha Descartes-jellegűeknek is nevezik, mivel a robotkar tengelyeit a Descartes által kifejlesztett koordinátarendszer x, y és z koordinátáival lehet leírni. Ahogyan például egy ábrán lévő pont helyzetét az x és y tengelyekhez viszonyítva lehet meghatározni, pontosan ugyanúgy lehet a robotkar mozgását egy képzeletbeli x tengely mentén előre és hátra illetve egy képzeletbeli y tengely mentén jobbról balra megadni. A robotkar fel és le irányuló mozgásait egy képzeletbeli z tengely menti elmozdulásokkal lehet megadni (amelyek a rajzpapír síkjából kilépve a mélységet adnák meg), és ez az irány jelenti a harmadik dimenziót.

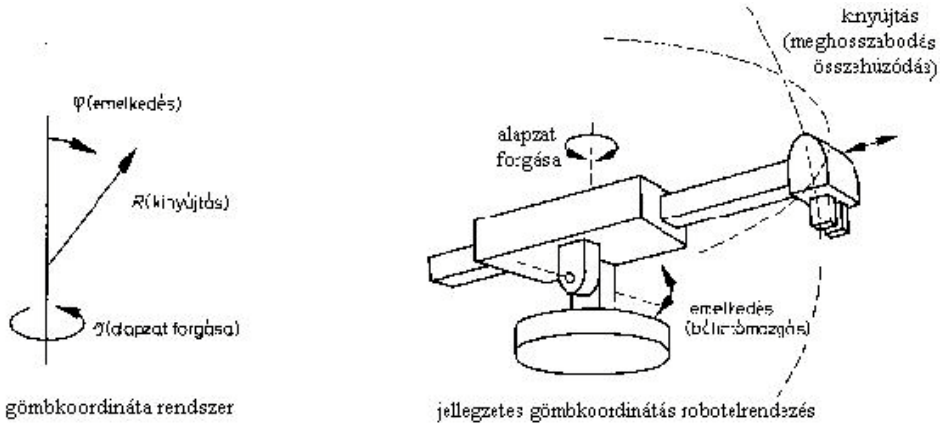
Ha a robotkar végére egy tollat erősítenénk akkor a toll a kar mozgásakor különböző nagyságú téglalapokat rajzolna le. Ezt az alakzatot, a kar által elérhető tartományt, működési tartománynak nevezik. Mivel a kar három tengely mentén mozoghat, ezért három szabadságfokú karnak mondják.

Egy *hengeres elrendezésű* kar (2. ábra) is három szabadságfokú, azonban ez csak az y és a z tengelyek mentén mozog egyenes vonalban. A harmadik szabadsági fok alapzatának a z tengely körüli forgása, és így működési tartománya henger alakú. Erre az esetre illik a vitorláshajó-hasonlat. Az y tengely ugyanúgy leng, mint a vitorláshajó vitorlarúdja, miközben fel és le csúszhat a z tengelyen. Ezenkívül kinyúlhat és összehúzódhat. Mivel lenghet, kinyúlhat vagy összehúzódhat, továbbá fel- és lefelé is elmozdulhat, az ilyen típusú robotkar végére erősített toll különböző sugárhosszúságú és különböző magasságú hengereket rajzolna. A robotkar vége által elérhető térbeli pontok hen-gerkoordinátákban (z, y, v) adhatóak meg.



2. ábra

A *gömbszerű elrendezés* a kar ringó mozgásával helyettesíti a hengeres elrendezés z tengely menti fel-le mozgását. Ebben az esetben már nincs árboc hanem csak a vitorlarúd, amely ring, imbolyog, kinyúlik és összehúzódik. A végére erősített toll különböző sugarú és méretű gömbfelületrészeket rajzolna (3. ábra).

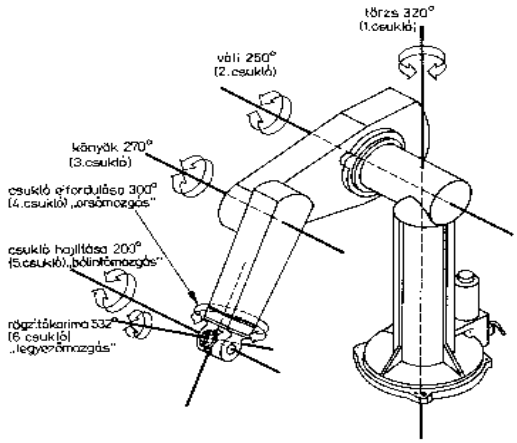


3. ábra

A *csuklós karos* elrendezés (4. ábra) a legnépszerűbb. A kar részei: a törzs, a váll, a felső kar, az alkar és a "csuklóelem". Míg a gömbszerű elrendezésű rendszer karja csak két tengely körül végezhet forgó mozgást, a csuklós szerkezetű kar mindegyik csuklója elfordulhat.

Az eddig tárgyalt mozgások a kar számára három szabadságfokú mozgásokat tesznek lehetővé. Az ipari robotok szabadságfoka azonban általában hat. A többi három a "csuklóelem" mozgási lehetőségei következtében jön létre, úgy ahogy az 4. ábrából kitűnik egy PUMA 560–as robot esetén, [1].

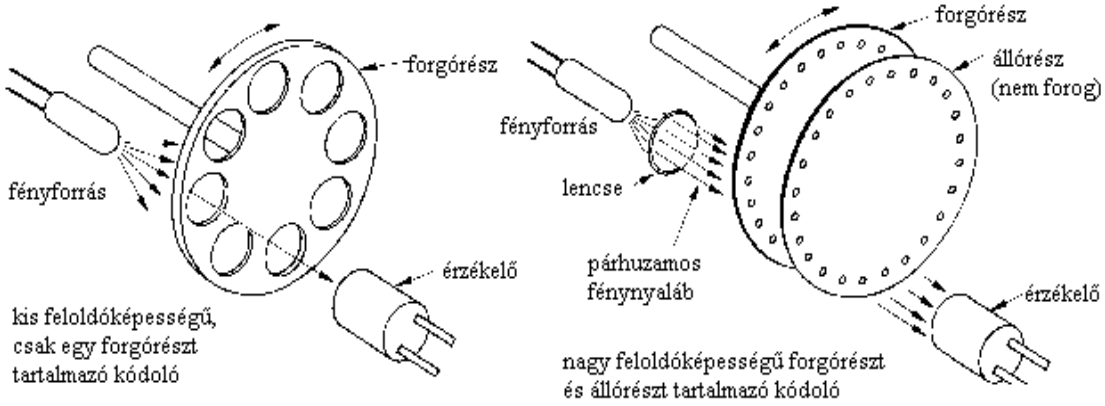
E mozgások a legegyszerűbben saját csuklónk mozgásai alapján érthetők meg, miközben a mutatóujjunkat kinyújtjuk. Ha kizárólag csuklómozdulatokkal akarnánk egy fel-le irányú, képzeletbeli vonalat rajzolni, akkor a kéz csuklója egy olyan képzeletbeli tengely körül forogna, amely párhuzamos azokkal a tengelyekkel, amelyek körül a 2. és a 3. csuklók forognak (4. ábra). E mozgás elnevezése *„bólintás"*. Ha kezünket balról jobbra, majd ismét visszafelé forgatjuk, akkor a kéz csuklója egy olyan, képzeletbeli tengely körül forog, amely merőleges a kéz csuklójának hajlító mozgására. E mozgás neve *legyezőmozgás*. Most forgassuk az egész alsókarunkat úgy, mintha a mutatóujjunkkal lyukat akarnánk fúrni. Ezt a mozgást *orsómozgás* néven ismerik.



4. ábra

3. Hogyan jön létre a robotkar ellenőrzött mozgása?

A robotkar típusától függetlenül minden egyes tengely illetve csukló szintjén találunk egy-egy (elektromos, hidraulikus vagy pneumatikus) motort, melyek működésbe hozzák a robotkar elemeit.



5.ábra

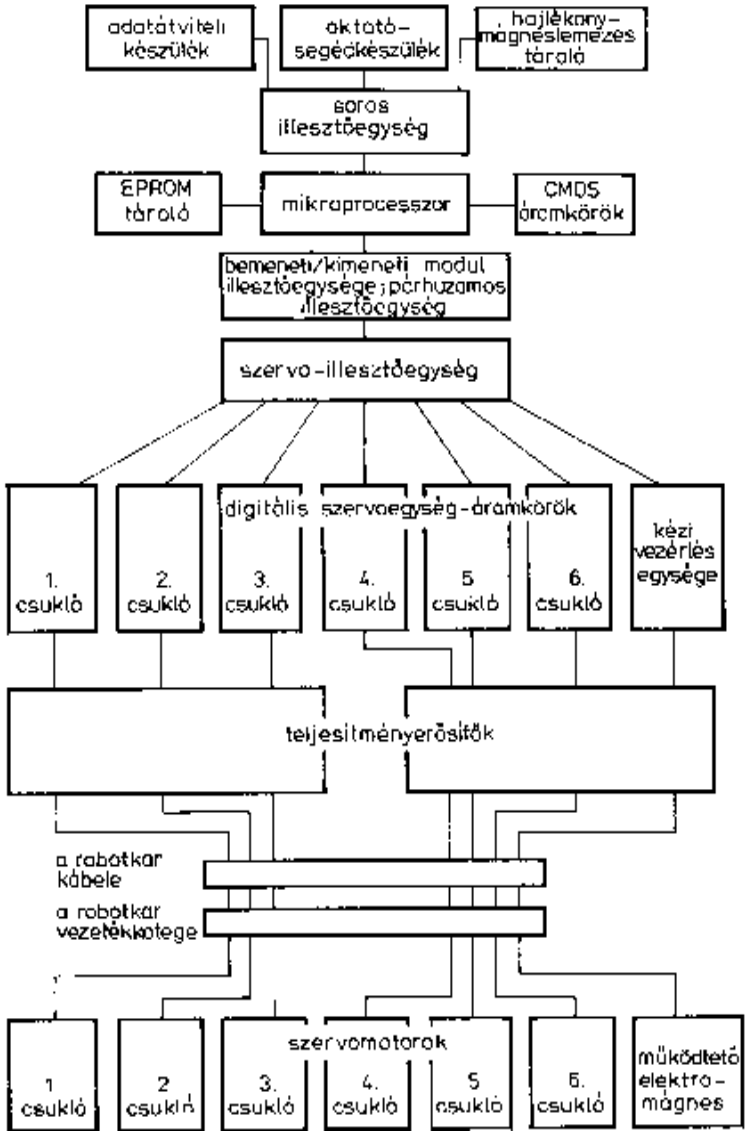
Minden egyes motor külön-külön vezérelhető. A vezérlőjeleket a robotkar mozgását felügyelő számítógép, a "robotvezérlő" hozza létre. Ez utóbbi folyamatosan követi a robotkar tényleges mozgását, például az egyes motorokra szerelt fény-megszakításos optikai kódolóelemek segítségével (lásd 5. ábra). Ezek a robotvezérlőt az egyes csuklók helyzetéről tájékoztatójelekkel látják el. A fénymegszakításos optikai kódoló úgy határozza meg a csukló elfordulásának mértékét, hogy megszámlálja, hogy a motor forgása közben hányszor szakad meg a kódoló fényforrásából kiinduló fényáram. Meghatározza a fénynyaláb megszakításainak sebességét is, és ebből a kar mozgási sebességére következtet.

4. Melyek a robotvezérlő jellegzetes elemei?

A 6. ábra egy PUMA 560–as, egyenáramú motorokkal felszerelt robot vezérlőjének a jellegzetes elemeit mutatja be, [1].

A robotvezérlőhöz, kívülről, periferiális készülékekhez csatlakoznak. Ezek: a *tanító (oktató) -segédkészülék*, egy *adatátviteli készülék* (amelyekre feltétlenül szükség van), továbbá egy *mágneslemezes tároló* és egy *bemeneti/kimeneti egység* (amely csak egyes esetekben szükséges).

A *tanító-segédkészülék* és az *adatátviteli készülék* hasonló célokat szolgál: mindkettő arra ad lehetőséget, hogy a robotvezérlőbe a kar mozgatására vonatkozó utasításokat lehessen bevezetni. A robot *tanító-segédkészülékkel* való programozását gyakran „végigvezetés" néven említik, mert a kart végigvezetik az egyes mozgásfázisokon. Katódsugár-csöves megjelenítővel vagy nyomtatóval felszerelt adatátviteli készüléket használva *"billentyűzetes" programozásról* szokás beszélni, mivel az utasításokat billentyűkkel juttatják be a rendszerbe.



6. ábra

A *mágneslemezes egység* a programokat ugyanolyan mágneslemezeken tárolja, amilyeneket a legtöbb személyi számítógéphez is használnak.

A *bemeneti/kimeneti egység* olyan adatátviteli eszköz, amely a robot és a többi szerszámgép tevékenységeit szinkronizálja.

Az *adatátviteli készülék* segítségével *betöltött* programokat a *CMOS* jelzésű tároló rendszer tárolja. A CMOS (ami a Complementary Metal Oxide Semiconductor = komplementer fémoxid jelvezető rövidítése) úgy tárolja az információt, hogy a program hibái egyszerű módon javíthatók, és a program könnyen módosítható. A kész program a mikroprocesszorba kerül, amely az EPROM tárolóból érkező utasításoknak megfelelően elvégzi a számításokat. Az eredményként kapott, szükséges, számított helyzetadatok a mikroprocesszorból egy digitális szervóáramkörbe kerülnek, amely ezeket összehasonlítja a kódolók által meghatározott,

pillanatnyi, ténylegesen érvényes helyzetadatokkal. Most következik a csuklóhelyzetek valóságos és elméleti értékeinek összehasonlítása és az eltérések meghatározása.

Az ebből az összehasonlításból származó korrigáló jeleket a digitális szervóáramkörök teljesítményerősítőkbe juttatják, amelyek feszültség- és áramszintjei megfelelőek az egyenáramú motorok táplálásához. Ezek a jelek vezetékeken keresztül jutnak a motorokhoz.

Könyvészet

1] ASIMOV, I.–FRENKEL, K. A.: *Robotok–Az emberformájú gépek,* Akadémiai Kiadó, Budapest, 1992.

BIST technológiák elosztott rendszerekhez

Szerző: Enyedi Szilárd, okl. mérn., MsC
BIST modul szerzője: Orghidan Radu, okl. mérn.
Tudományos vezető: dr. Liviu Miclea, docens
Kolozsvári Műszaki Egyetem, Számítástechnika és Automatizálás Kar
Automatizálás Tanszék

1. Rövid leírás

A dolgozat olyan kísérleteket mutat be, melyek három irányból próbálják megközelíteni a DBIST távirányítást:

- Java és CORBA felhasználásával;
- Web-en keresztül, HTML és PHP felhasználásával;
- GSM hálózaton keresztül, WAP/WML és PHP felhasználásával.

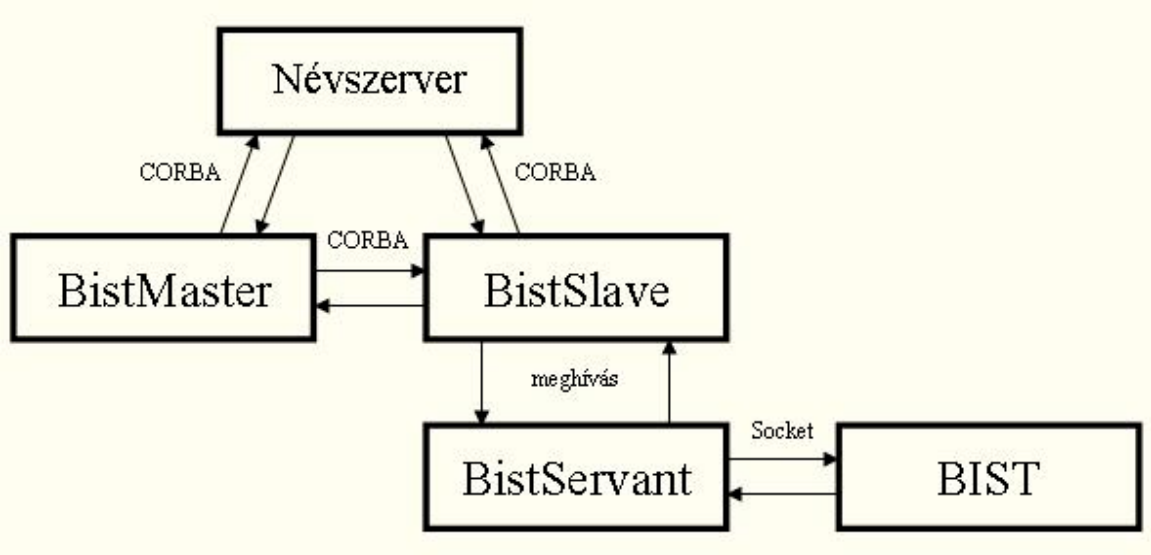
2. Általánosságok

- *DBIST?*
  - DBIST (Distributed Built-In Self-Test);
  - a modulok tesztelését egy beépített teszt modul végzi a központi modul kérésére;
  - a tesztek eredményei a központba jutnak;
  - a modulok együtt egy elosztott rendszert alkotnak.
- *CORBA?*
  - CORBA (Common Object Request Broker Architecture);
  - egy technológia, aminek felhasználásával egy program meghívhatja egy objektum függvényét a hálózaton keresztül az operációs rendszertől és a függvény programozási nyelvétől függetlenül;
  - a hívó és a hívott félnek van egy-egy ORB-ja (Object Request Broker), melyek egymással adatokat váltanak az IIOP (Internet Inter-ORB Protocol) protokoll betartásával.
- *Web?*
  - World Wide Web (WWW, vagy Web);
  - a webszerver (IIS, Apache) elküldi a kért weboldalt a kliensnek (Internet Explorer, Netscape, Opera, HotJava webböngésző).
- *PHP?*
  - PHP (Personal Homepage Preprocessor);
  - a PHP állományok PHP forráskódot tartalmaznak;

– ezt a forráskódot a webszerver futtatja a weblap klienshez való küldése előtt;
– mikor a webböngésző egy PHP oldalt kér a szervertől, az utóbbi futtatja a kért weboldalban (HTML állományban) talált programot, és az eredményt küldi el a kliensnek (webböngészőnek).

– *WAP? WML?*
  – WAP (Wireless Application Protocol);
    – lehetővé teszi az Internet elérést mobil készülékekről (mobiltelefonok, rádiókapcsolatú elektronikus zsebkönyvek stb.);
    – a mobil készülékek mikro-böngészőt tartalmaznak ("*microbrowser*"), mely lekéri a webszerverekről, feldolgozza, és a mobil készülék képernyőjére kiírja a WML oldalak tartalmát;
  – WML (Wireless Markup Language);
    – hasonlít a HTML-re, de sokkal egyszerűbb, a mobil készülékek erőforrásaihoz és főleg kijelzőjéhez talál.

3. Modulközti adatcsere a Java/CORBA változatban

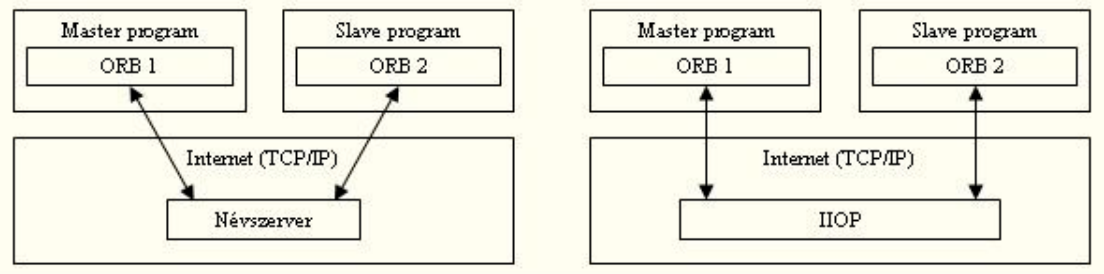


Adatcsere a modulok közt a Java/CORBA változatban

4. Kapcsolat névszerveren keresztül

– Mindkét fél (master és slave) bemutatkozik a névszervernek, mely bemutatja őket egymásnak, aztán az utóbbi kettő közvetlen kapcsolatot létesít egymással;

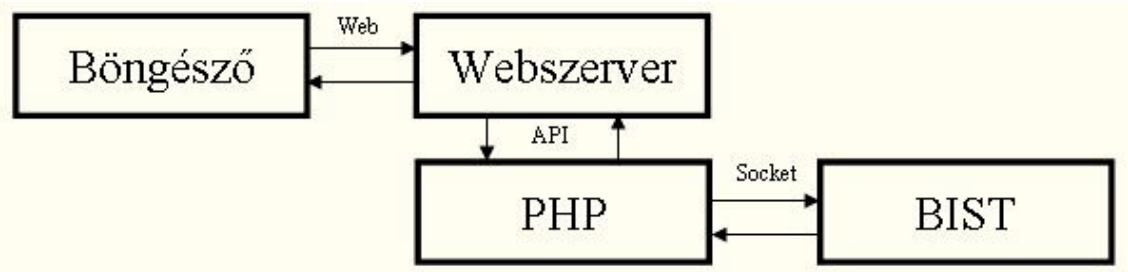
– A közvetlen kapcsolat felépítése után a két fél névszerver nélkül cserél adatokat, a névszervert akár ki is lehet kapcsolni (ezért “tranziens”).



1. Kapcsolatindítás névszerveren keresztül *2. Kapcsolat*

5. DBIST Web-en keresztül

– A böngésző lekéri a PHP oldalt a webszerverről;
– A szerver futtatja a PHP kódot (amit a kért oldal tartalmaz), és az eredményt (az elektronikus űrlapot) elküldi a kliensnek (a böngészőnek);
– A felhasználó kitölti az űrlapot (DBIST modul címe és a kért teszt neve) és visszaküldi a szervernek;
– A szerver újból futtatja a PHP kódot a kapott adatokkal; a PHP modul a BIST modulhoz küldi a kért tesztet; ez futtatja a tesztet és küldi az eredményt a PHP modulhoz;
– A szerver megalkotja az eredményt tartalmazó weboldalt, elküldi a böngészőhöz, ami kiírja a képernyőre.

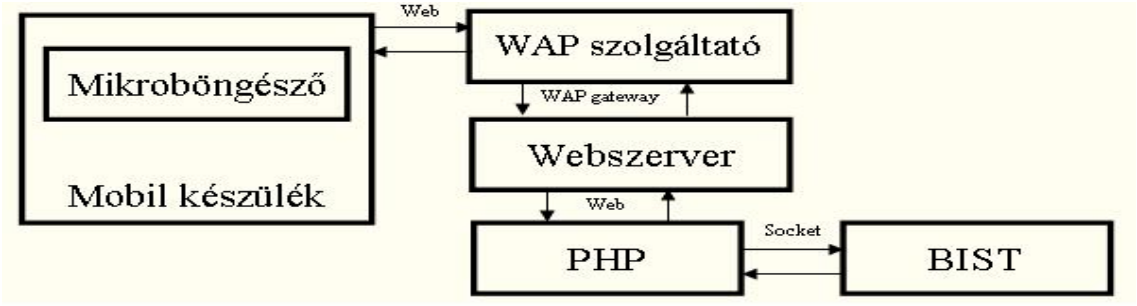


DBIST Web-en keresztül

6. DBIST WAP-on keresztül

– A mikro-böngésző (“microbrowser”) lekéri a PHP oldalt a webszerverről;
– A szerver megalkotja az elektronikus űrlapot (a PHP modul segítségével) és elküldi a mikro-böngészőnek;

- A felhasználó kitölti az űrlapot a BIST modul címével és a kért teszttel és visszaküldi a szervernek;
- A szerver PHP modulja feldolgozza a kapott eredményt – bekapcsolódik a BIST modulhoz, elküldi a kért teszt nevét és megkapja az eredményt;
- A szerver megalkotja az eredményt tartalmazó weboldalt és elküldi a mikroböngészőnek, mely kiírja a képernyőre.



DBIST WAP-on keresztül

7. Következtetések

- A Java/CORBA megoldás elegáns és több operációs rendszeren is megy, de lassú;
- A Web/PHP megoldás megy több operációs rendszeren, de webszervert és PHP modult kell telepíteni és beállítani;
- A WAP/WML megoldás mobil, de drága és nem minden mobil készülék/szolgáltató támogatja.

8. További információk

- BIST
  http://nic.savba.sk/~upsyvord/BIST/sld005.htm
  http://www.ece.neu.edu/info/vhdl/test/cstp.html
- CORBA
  http://www.omg.org/corba/beginners.html
  http://www.cs.wustl.edu/~schmidt/tutorials-corba.html
- HTML
  http://www.w3c.org
  http://www.w3schools.com
- PHP
  http://www.php.net
  http://www.phpbuilder.com
  http://php.resourceindex.com

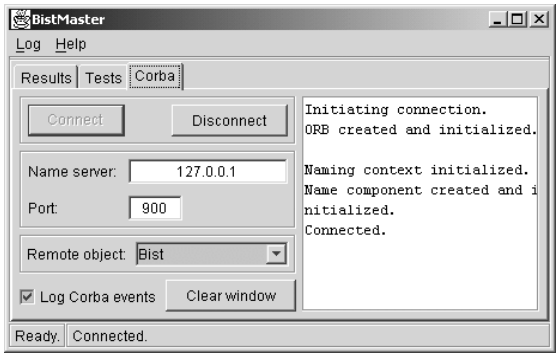
http://www.daholygoat.com
– WAP/WML
http://www.wapforum.org
http://www.gelon.net

9. Képek a modulok grafikus interfészéről

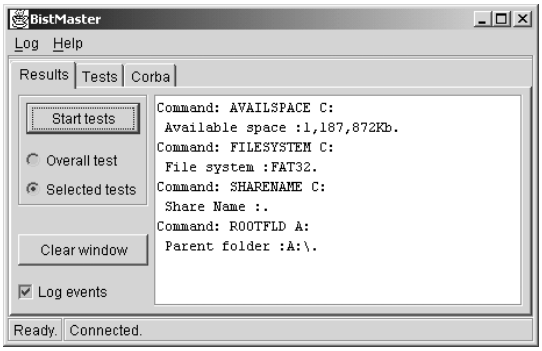
A BIST modul helyi válasza az "AVAILSPACE C:" parancsra (elküldi a „megrendelőnek", de helyileg is kiírja)

Megjegyzés: *mindhárom változat (Java/CORBA, Web/PHP és WML/PHP ugyanazzal a BIST modullal dolgozik).*

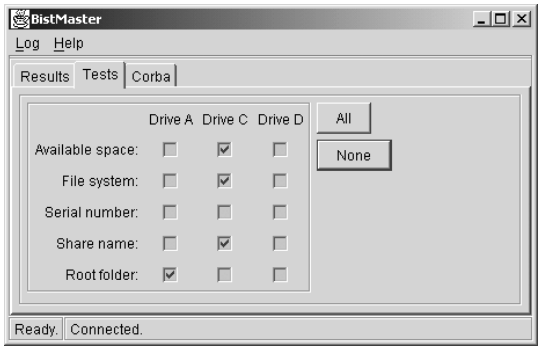
9.1 Java/CORBA változat



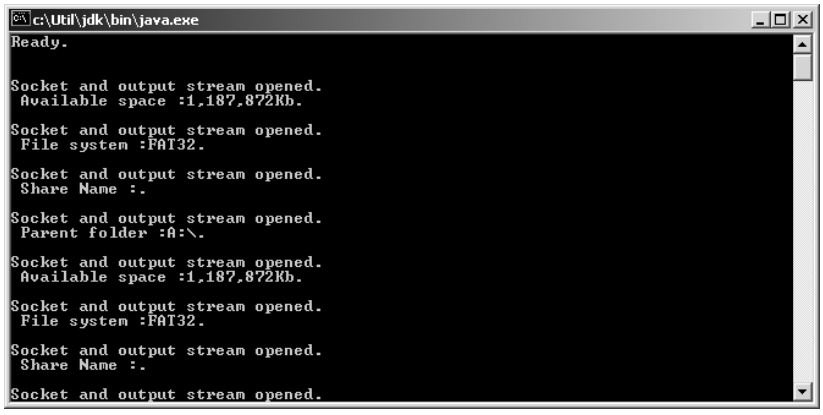
A BistMaster modul CORBA beállításai



A kért tesztek kijelölése a BistMaster modulban



A tesztek eredményei a BistMaster modulban



A BistSlave modul eredményei a JVM ablakban

9.2 Web/PHP változat

Test configuration
Top of Form

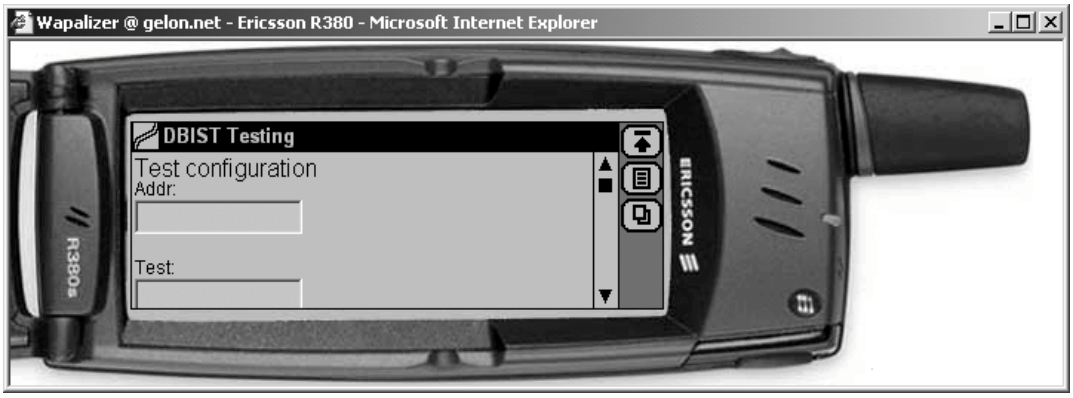
BIST module address: 172.27.36.1
Requested test (case sensitive): TESTALL
Test

Az elektronikus űrlap a Web böngészőben, amint azt a webszerver a PHP modul segítségével megalkotta

Test results
BIST module address: **127.0.0.1**
Requested test: **availspace c:**
Opening connection to BIST module...done.
Sending test command "availspace c:"...done.
Reading response from BIST module...done.
Available space :1,188,384Kb.
Closing connection to BIST module...done.

A teszt eredménye a Web böngészőben, amint azt a webszerver a PHP modul segítségével megkapta a BIST modultól

9.3 WML/PHP változat



A tesztelési űrlap egy WML böngészőben, egy Ericsson R380 telefonon. A telefon szimulált a *http://www.gelon.net* címen található Web oldalon. A böngészőben, a „Test” című szövegsor alatt egy „Send” gomb van (nem fért ki a telefon képernyőjén).

Descrierea CIP a Bibliotecii Naţionale a României
Terminológia: Magyar nyelvű szakelőadások a 2001-2002-es tanévben: Villamosmérnöki kar – Kolozsvár
[Cluj-Napoca]:
Erdélyi Magyar Műszaki Tudományos Társaság, 2002
p. ; cm.
ISBN 973-85809-1-9

004

Erdélyi Magyar Műszaki Tudományos Társaság - EMT
Kolozsvár, 1989. December 21. Sugárút (Magyar u.) 116. Szám
Postacím: 3400 Cluj, C.P. 1-140, România
Tel./fax: 0264-190825; 194042; 0744-783237
E-mail: emt@emt.ro
Honlap: http://www.emt.ro

ISBN 973-85809-1-9